12-2018

Reinforcement learning for collective multi-agent decision making

Duc Thien NGUYEN
Singapore Management University, dtnguyen.2014@phdis.smu.edu.sg

REINFORCEMENT LEARNING FOR COLLECTIVE MULTI-AGENT DECISION MAKING

NGUYEN DUC THIEN

SINGAPORE MANAGEMENT UNIVERSITY

2018

Reinforcement Learning for Collective Multi-agent Decision Making

Nguyen Duc Thien

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

LAU Hoong Chuin (Supervisor/Chair)
Professor of Information Systems
Singapore Management University

Akshat KUMAR (Co-Supervisor)
Assistant Professor of Information Systems
Singapore Management University

Qin ZHENG
Deputy Department Director
Institute of High Performance Computing, A*STAR

Pradeep VARAKANTHAM
Associate Professor
Singapore Management University

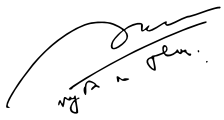
Singapore Management University

2018

I hereby declare that this PhD dissertation is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree in any university previously.



Nguyen Duc Thien

6 December 2018

Reinforcement Learning for Collective Multi-agent Decision Making

Nguyen Duc Thien

Abstract

In this thesis, we study reinforcement learning algorithms to *collectively* optimize decentralized policy in a large population of autonomous agents. We notice one of the main bottlenecks in large multi-agent system is the size of the joint trajectory of agents which quickly increases with the number of participating agents. Furthermore, the *noise* of actions concurrently executed by different agents in a large system makes it difficult for each agent to estimate the value of its own actions, which is well-known as the multi-agent credit assignment problem. We propose a compact representation for multi-agent systems using the aggregate counts to address the high complexity of joint state-action and novel reinforcement learning algorithms based on value function decomposition to address the multi-agent credit assignment problem as follows:

1. Collective Representation: In many real-world systems such as urban traffic networks, the joint-reward and environment dynamics depend on only the number of agents (the count) involved in interactions rather than agent identity. We formulate this sub-class of multi-agent systems as a Collective Decentralized Partially Observable Markov Decision Process ($\mathbb{C}$Dec-POMDP). We show that in $\mathbb{C}$Dec-POMDP, the transition counts, which summarize the numbers of agents taking different local actions and transiting from their current local states to new local states, are sufficient-statistics for learning/optimizing the decentralized policy. Furthermore, the dimensions of the count variables are not affected by the population size. This allows us to transform the original planning problems to optimize the complex joint agent trajectory into optimizing compact count variables. In addition, samples of

the counts can be efficiently obtained with multinomial distributions, which provide a faster way to simulate the multi-agent systems and evaluate the planning policy.

2. Collective Multi-agent Reinforcement Learning (MRL): Firstly, to address multi-agent credit assignment problem in $\mathbb{C}$Dec-POMDP, we propose the collective decomposition principle in designing value function approximation and decentralized policy update. Under this principle, the decentralized policy of each agent is updated using an *individualized* value instead of a joint global value. We formulate a joint update for policies of all agents using the counts, which is much more scalable than independent policy update with joint trajectory. Secondly, based on the collective decomposition principle, we design 2 classes of MRL algorithms for domains with local rewards and for domains with global rewards respectively. i) When the reward is decomposable into local rewards among agents, by exploiting exchangeability in $\mathbb{C}$Dec-POMDPs we propose a mechanism to estimate the individual value function by using the sampled values of the counts and average individual rewards. We use this count-based individual value function to derive a new actor critic algorithm called fAfC to learn effective individual policy for agents. ii) When the reward is non-decomposable, the system performance is evaluated by a single global value function instead of individual value functions. To follow the decomposition principle, we show how to estimate individual contribution value of agents using partial differentials of the joint value function with respect to the state-action counts. This is the basis for us to develop two algorithms called MCAC and CCAC to optimize individual policy under non-decomposable reward domains. Experimentally, we show the superiority of our proposed collective MRL algorithms in various testing domains: a real-world taxi supply-demand matching domain, a police patrolling game and a synthetic robot navigation domain, with population size up to 8000. They converge faster convergence and provide better solutions than other algorithms in the literature, i.e. average-flow based algorithms and standard actor critic algorithm.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I thank God for His providence throughout my life, which includes this PhD. I prayed to Him from the beginning to decide on the place to do my PhD, on my research topic, on every submission of my paper. His answers for me have never gone wrong. I worked with the best supervisors in the most interesting research topic to publish the best papers. In addition to all the academic achievements, God also allowed me to travel overseas several times and this made my PhD time more than a joy.

I would like to thank my supervisor, Lau Hoong Chuin, with whom I have been working for almost 7 years since the first day in Singapore. He gave me the opportunity to work in various projects with many collaborators. He guided me through this very important part of my career and shaped my character to become an excellent individual and researcher.

I would also like to thank my co-supervisor, Akshat Kumar, who has been working back-to-back with me over many weekend evenings to rush for submission deadlines. Akshat taught me how to develop an initial idea into an interesting research direction. We had discussed a lot of interesting concepts during coffee and lunch times and I am sure all of these discussions will be beneficial to my research later. I think by God's will, I was destined to work with him in my PhD. If I had chosen to come to a different place to do my PhD, he would also be there to help me.

I am also blessed to have many collaborators and good friends during my Ph.D. Amongst them, I would like to thank my supervisor in A*STAR, Qin Zheng. Qin

has been very supportive in all of the events in my PhD process. I also always remember William Yeoh as my additional advisor, who gave me a lot of useful tips for my PhD. Pradeep Varakantham was always available to help me go through difficult research challenges.

And the final version of my thesis would never be done without the help of my good friends John and Beverly, who have helped to read every of my nearly 200 thesis pages.

Finally, I want to dedicate my PhD to my family, especially my wife. She has been accompanying me in all the sweet and sour moments of my PhD. Without her love and support, I would never have attempted to come to Singapore and finish this PhD.

“with God all things are possible”-Matthew 19:26

Chapter 1

Introduction

Many real-world problems can be modelled as optimizing actions of autonomous agents over time to maximize some utility or reward functions. Examples include optimizing the movement of autonomous vehicles to serve passengers better, or optimizing the speed of autonomous vessels to reduce congestion in a port. We can model the decision of each autonomous agent as a decentralized policy function taking input as agent's local information about the system, such as the congestion level at its current location, and outputs an action, such as the speed or direction to move, for that agent to execute. In this thesis, we study multi-agent reinforcement learning (MRL), which is the process to learn decentralized policy function for autonomous agents from empirical reward feedback from environments or simulation engines. The objective of MRL algorithms is to adjust the policy according to the empirical feedback to maximize the expected total rewards over a planning horizon [96, 88, 124]. Reinforcement learning is shown to be efficient methodology to optimize policy in complex domains. Among the most well-studied MRL domains are distributed robot systems [5], in which autonomous robots are designed to collaborate with each other to achieve goals, for example to win a robot soccer game [9] or to maximize the number of rescued victims in disaster response tasks [72]. Multi-agent reinforcement learning is also studied in multiaccess broadcast

channel network problems, in which a decentralized controller associated to each user needs to independently make decisions about whether to transmit the packet in each specific time slot [42, 145, 85, 84]. In such problems, decentralized controllers have to cooperate to avoid the network conflict and to maximize the total throughput of the network. Other decentralized control problems are studied in power grids [100], traffic light control [32, 142], sensor network control [125, 62], or fire fighting teamwork [4]. Recently, with the availability of computing resources to train artificial neural networks, researchers are able to train neural network policies for multi-agent systems in complex environments such as Atari video games [120] or strategic video game StarCraft [30]. Learning good decentralized policy is shown to enable AI agents to even form their own language as a communicative means to achieve a goal [29, 64].

Although multi-agent reinforcement learning problems have been studied for a long time, the complexity of joint state-action spaces remains a challenge to overcome. In multi-agent systems, each agent could have separate local observations of the environment and independently take action. However their actions will jointly affect each other's transition and reward. For example, when many vehicles choose to take a specific road, the congestion could slow-down the speed of all vehicles and as a consequence, increase the cost for each vehicle. Because of the interdependence between agents, when optimizing an individual policy, we have to consider all state-actions of the related agents. In this thesis, we use MDP terminology to refer a local "*trajectory*" as a sequence of local state-actions of a agent and joint "*trajectory*" as a sequence of joint state-actions of all agents. In multi-agent planning, to estimate the individual value, an agent might need access to joint *trajectory* to know where *all* other agents are and which actions *all* other agents take over planning horizon.

In multi-agent systems, the joint state-action space grows exponentially with the number of agents. Mathematically speaking, if local state space of each agent is

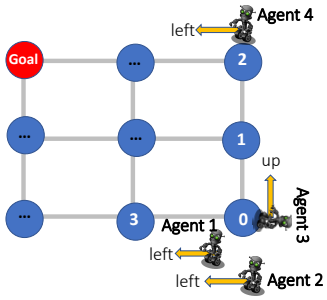
S and its local action space is A, then the size of the joint state-action space of N agents $(|S| \times |A|)^{|N|}$. This complexity of the joint space is one of main hurdles to run and evaluate contemporary multi-agent reinforcement learning algorithms. Without addressing this scalability, we are a long way from deploying MRL algorithms into real world scenarios consisting of thousands or even millions of agents. This motivates me to develop a compact representation for multi-agent planning problems and subsequently scalable reinforcement learning algorithms for large scale multi-agent systems.

As the basis for scalable multi-agent reinforcement learning algorithms, I propose to use the counts to compactly model a class of multi-agent systems and efficiently optimize individual agent policy in such systems. A count is defined as the number of agents belonging to a particular category, for example to be in a specific local state or location. Counts have been used in many graphical inference problems in machine learning as useful sufficient-statistics [49, 17, 108, 79]. In multi-agent systems, the counts can be used to express the collective behaviors of agents in population and to adjust the collective behaviors according to the feedback from the environment. For instance, the counts used in traffic domains to record the numbers of vehicles coming into each road sector at each time period. By looking at the traffic counts, one can reduce the congestion in a sector by adjusting its heavy incoming flows. In my thesis, to formulate and generalize this count-based decision making concept, I define a model called Collective Decentralized Partially Observable Markov Decision Process ($\mathbb{C}$Dec-POMDP) and then develop reinforcement learning algorithms to optimize individual policy for multi-agent planning in $\mathbb{C}$Dec-POMDPs.

1.1 Collective Decision Making Framework

Before providing further details of our collective planning framework, we give an example of multi-agent domain and compare two representations of the problem. This is followed by the general concept of multi-agent reinforcement learning that we will develop for $\mathbb{C}$Dec-POMDPs in this thesis later and a classification of reinforcement learning methods.

1.1.1 Example of Multi-agent Domain



(a) A snapshot at $t = 0$.

AgentID	s_0	a_0	...
Agent_1	0	$left$	...
Agent_2	0	$left$	...
Agent_3	0	up	...
...	...	...	...

(b) Joint state-action table

$\langle s_t, a_t \rangle_t$	$\mathrm{n}_t^{\mathrm{sa}}$
$\langle 0, left \rangle_{t=0}$	2
$\langle 0, up \rangle_{t=0}$	1
$\langle 1, up \rangle_{t=0}$	0
...	...

(c) Count table

Figure 1.1: Example of joint state-action and count table in grid navigation problem.

We consider robot grid navigation (illustrated in Figure 1.1) as an example of a multi-agent problem. This motivating example is a common testbed for multi-agent reinforcement learning algorithms [132, 66, 76]. In this example, a team of 4 robots try to move in a grid of size 3×3 toward some goal locations while avoiding conflict in narrow corridors. A goal could be the representative of a victim in a disaster rescue situation or some object to be picked-up in a transportation task. At each time period t, a robot in a location s_t has to choose an action as one of 5 movement $\{left, right, up, down, stay\}$. Consider the snapshot of the grid navigation at time $t = 0$. Agents $1, 2, 3$ are in location 0, while agent 4 is in location 2. From their local state, agent 1 and 2 take left-turn action while agent 3 goes up, agent 4 takes a left-turn. This can be summarized by the joint state-action Table 1.1b, in which we record the local state, and local action for every agent at every time step. On the

other hand, we can summarize agent movements by the count Table 1.1c, in which each entry $\mathrm{n}_t^{\mathrm{sa}}(i, j)$ is the number of agents taking a specific action j from local state i at time t. As seen in this example, when the number of agents increase, we have to create new rows to record new agents in a joint state-action Table 1.1b. Meanwhile, the number of entries in the count Table 1.1c is fixed with regard to the number of agents. When the number of agents is significantly larger than the number of local and states and actions, maintaining the count table becomes much more efficient than maintaining the joint state-action table. In MRL, using the counts as input to the decentralized policy and value functions could reduce the input dimensions of those functions in comparison to original joint state-action input. As a result, the counts could improve the scalability and convergence of MRL algorithms.

1.1.2 Multi-agent Reinforcement Learning

The overview of the MRL framework is illustrated in Figure 1.2. In general, reinforcement learning is a nature inspired principle to learn action selection function from the reward feedback provided by the environment. In this MRL framework, our goal is to learn a decentralized policy function (called *"actor"* in the literature) based on the approximation of system's value (called *"critic"* in the literature) [117].

At each time step, the decentralized policy function maps **local observation** of an agent to a local action of that agent. Notice that the policy function can be a stochastic function, which produces random actions under an action distribution. After actions are made by agents, they would jointly affect the environment. The result of interaction between joint action and environment is the **joint transition** to the new **joint state** of agents and **joint rewards**.

To optimize/learn the decentralized policy, MRL estimates and assigns values to each executed action of individuals. In single agent problems, this value can

be the empirical returns (or the total empirical rewards accumulated from the time point when that action is executed). However, due to stochasticity of the problem, empirical return is a random variable with high variance. Furthermore, in multi-agent domains where multiple individual actions are executed at the same time, the *raw* empirical return is too noisy to distinguish roles of concurrent actions. To address this, we a) resort to an **approximate value function** (*"critic"*) trained to estimate the empirical return (**train critic**), and hence resolve the high variance problem of the empirical return, b) propose an efficient policy update to train the policy function (**train actor**) by the critic.

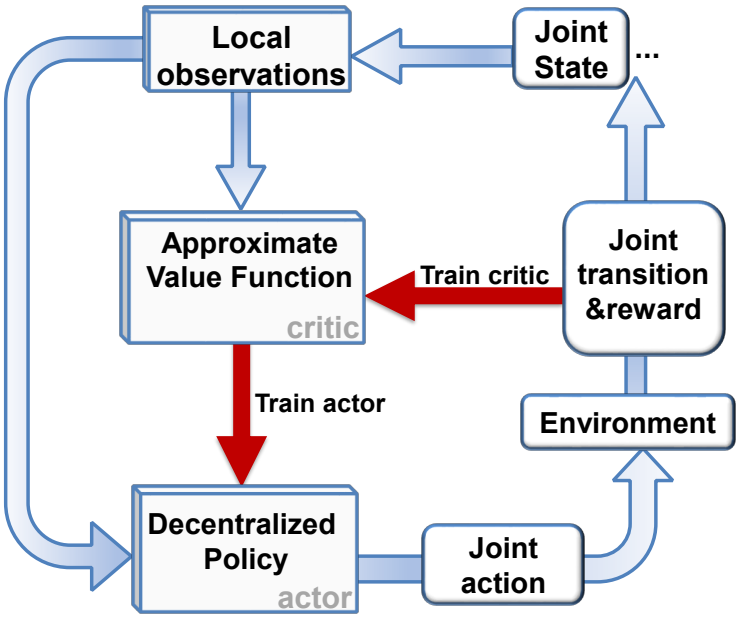


Figure 1.2: Multi-agent Reinforcement Learning framework

Shared policy

Instead of multiple decentralized policies for different agents, we consider a single decentralized policy function shared by homogeneous agents in the system. In fact, learning a *shared* (or homogeneous) *policy between agents* is a common objective in multi-agent reinforcement learning literature [121, 44, 147, 131, 40]. In large scale domains such as movement of animal flocks or a traffic network, a homogeneous behavior model of individuals is usually assumed [108, 57]. In our research

problems, by optimizing the shared policy, we can collectively *shape* behaviors of individuals in favor of system quality. To extend our model to heterogeneous agents, we can generalize shared policy by considering the type of agent as an input feature into the policy function.

Centralized Learning-Decentralized Execution

When an autonomous agent executes its decentralized policy, it only possesses a local view (or partial observation) of the systems. However, in $\mathbb{C}$Dec-POMDPs, the dynamicity and the reward of an individual is correlated with others. Local view is sometimes insufficient to learn a decentralized policy. An individual needs to know or estimate behaviors of others to act accordingly [31, 64]. An example in a navigation problem is when an agent knows the intention of another agent is to take a narrow corridor. It could plan to not take that corridor to avoid collision.

To overcome issues of partial observation in policy learning, we would learn the policy by a centralized planner off-line before deploying them into decentralized agents. This centralized planning-decentralized execution paradigm is a common practice in multi-agent reinforcement learning [56]. The centralized planner would reason on either the complete model of the domain [12, 73, 82], or samples of global states generated by a black-box simulator [120]. When neither of these is available, to approximate *global* view, the centralized planner can aggregate local observations from historic data to have a joint view of the system [31]. The global view in the learning phase provides more information for the centralized planner to better estimate value function and refine decentralized policy accordingly. In addition, we can impose the cooperativeness in decentralized policy by the centralized planner. In our collective planning framework, we assume the centralized learner has the access to the joint counts and rewards of the system.

1.1.3 Reinforcement Learning Classification

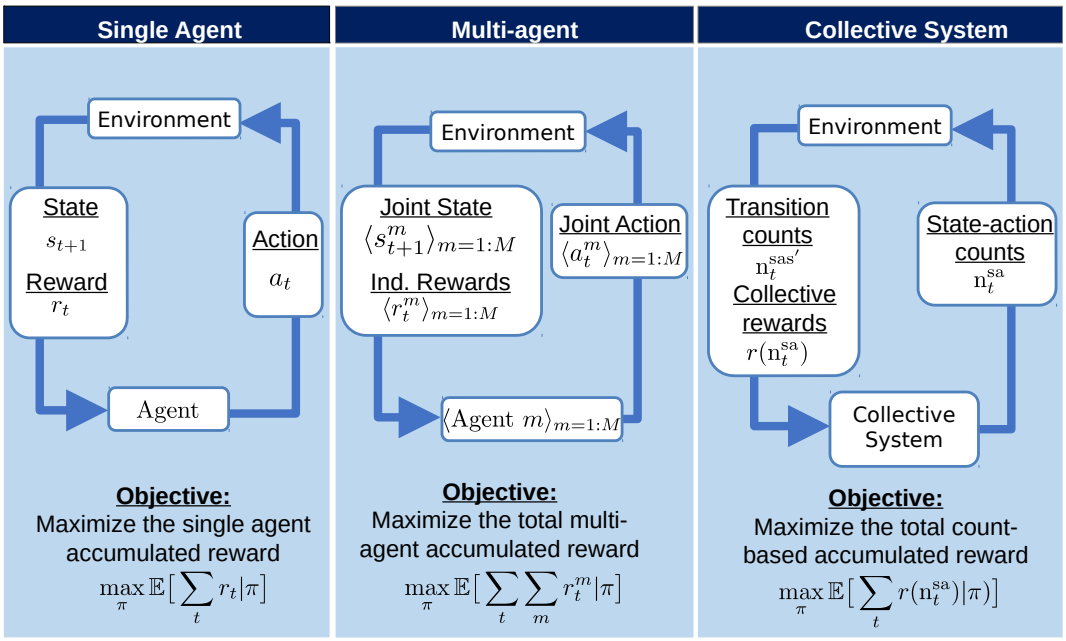


Figure 1.3: Multi-agent Reinforcement Learning Classification.

In Figure 1.3, we compare a single agent reinforcement learning system, a multi-agent reinforcement learning system and a collective learning system in a cooperative setting. The goal of reinforcement learning algorithm is to optimize the system total rewards through interacting with environment. We want to emphasize the difference in representations of state, reward, action and objective function of the three systems.

In the single agent system, the state, action and reward are singular. The objective in the single agent system is to maximize a single agent expected accumulative reward when the agent interacts with an environment.

The realization of a multi-agent system at time t is represented by samples joint state $\langle s_t^m \rangle_{m=1:M}$ of all agents $m = 1 : M$, their action $\langle a_t^m \rangle_{m=1:M}$ and corresponding joint rewards $r(\boldsymbol{s}_t = \langle s_t^m \rangle_{m=1:M}, \boldsymbol{a}_t = \langle a_t^m \rangle_{m=1:M})$ depending on the joint state-action of agents. In some case, the joint reward can be represented as a sum of

individual reward as

$$r(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{m=1:M} r^m(s_t^m, a_t^m, \boldsymbol{s}_t, \boldsymbol{a}_t).$$

Each $r^m(s_t^m, a_t^m, \boldsymbol{s}_t, \boldsymbol{a}_t)$ represents the reward of agent m receives when it takes action a_t^m in local state s_t^m and given the joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t)$.
An objective in a cooperative multi-agent system is to maximize the total expected rewards over planning horizon when multiple agents interact with an environment.

$$\max_{\pi} \mathbb{E}\big[\sum_{t} r(\boldsymbol{s}_t, \boldsymbol{a}_t|\pi\big] = \max_{\pi} \mathbb{E}\big[\sum_{t} \sum_{m=1:M} r^m(s_t^m, a_t^m, \boldsymbol{s}_t, \boldsymbol{a}_t)|\pi\big].$$

In the collective planning model, the identity of each agents is marginalized out in collective variables (containing no agent index). The joint state $\langle s_t^m \rangle_{m=1:M}$ and action $\langle a_t^m \rangle_{m=1:M}$ are summarized into the state-action counts $\mathrm{n}_t^{\mathrm{sa}}$. The joint reward is a function of state-action counts as $r(\mathrm{n}_t^{\mathrm{sa}})$. In some case, the joint reward can be presented as a sum of individual rewards as

$$r(\mathrm{n}_t^{\mathrm{sa}}) = \sum_{i \in S, a \in A} \mathrm{n}_t^{\mathrm{sa}}(i, j) r(i, j, \mathrm{n}_t^{\mathrm{sa}})$$

Each reward $r(i, j, \mathrm{n}_t^{\mathrm{sa}})$ represents the average reward of an agent (regardless of its identity) in state i taking action j when the joint state-action counts are $\mathrm{n}_t^{\mathrm{sa}}$. The joint objective value of the multi-agent reinforcement learning is re-written in the count and average reward variables as

$$\max_{\pi} \mathbb{E}\big[\sum_{t} r(\mathrm{n}_t^{\mathrm{sa}})|\pi\big] = \max_{\pi} \mathbb{E}\big[\sum_{t} \sum_{i \in S, a \in A} \mathrm{n}_t^{\mathrm{sa}}(i, j) r(i, j, \mathrm{n}_t^{\mathrm{sa}})|\pi\big].$$

Later we would show that in a collective system, instead of sampling joint trajectory $\langle s_{1:H}^m, a_{1:H}^m \rangle_m$, we only need to sample the state-action counts $\mathrm{n}_{1:H}^{\mathrm{sa}}$. In

large populations, the action count sampling process of collective distribution is generally much cheaper than joint trajectory sampling. More details of collective distribution and sampling process are provided in Chapter 2.

1.2 Summary of Contributions

Our main contributions to the multi-agent collective decision making are two-fold. Firstly, we propose a novel representation for the collective planning problems in $\mathbb{C}$Dec-POMDP model using the count variables. Secondly, based on the new planning representation, we develop count-based reinforcement learning algorithms to efficiently optimize individual policy for collective planning problems.

1.2.1 Count-based Representation for Collective Planning

Main research challenge: The complexity of multi-agent representation grows when the number of agents in the system increases. This causes a big challenge in managing the state of large population systems for planning purpose. This complexity bottleneck is present in our domains of interest, i.e. traffic network and transportation supply-demand matching, where number of agents could vary from 10 to 8000. To address this problem, we propose a compact representation using the count variables.

Technical contributions: We are motivated by the recent advance in collective graphical model (CGM) proposed by Sheldon and Dietterich [108] in showing the counts to be a *lifted* representation of the population. Sheldon and Dietterich [108] show that the counts are sufficient-statistics for inference of collective behaviors. However, CGM is limited in domains where individual policy and transition function of each agent are independent from others. We generalize this notion of the col-

lective representation with the count to multi-agent planning problems by proposing what we term as the $\mathbb{C}$**Dec-POMDP** model, in which the transition and reward function of an individual agent depends on the collective behaviour of the population. We show that in $\mathbb{C}$Dec-POMDPs, we can marginalize joint trajectory of agents into the count variables. In addition, we show the count variables are sufficient statistics for planning in $\mathbb{C}$Dec-POMDP. This means that we can write the global value function in $\mathbb{C}$Dec-POMDPs as a function of count variables and we can optimize the objective value of $\mathbb{C}$Dec-POMDPs by changing parameters of the collective distribution of the count variables. The collective distribution of the count variables provides a fast simulation of the collective system by sampling the counts instead of sampling individual trajectories. This lays the foundation for latter development of efficient planning algorithms using this count representation.

1.2.2 Collective reinforcement learning algorithms

Main research challenge: Due to the complexity of joint trajectory in multi-agent systems, many current multi-agent reinforcement learning (MRL) algorithms are only evaluated in small domains with few agents [58, 120, 30, 31]. We want to exploit the compact representation with the count variables to develop count-based multi-agent reinforcement learning algorithms scalable to large populations. The main research challenge in designing such algorithms is how to estimate the *credit* (as numeric representation of the role) of each individual action to the total reward of the system. The *credit* provides the local feedback for each agent to update its policy accordingly. In $\mathbb{C}$Dec-POMDPs, we have to compute the credit from collective variables instead of the joint trajectories as in the MRL literature.

Technical contributions: Our algorithmic contributions are in development of count-based MRL algorithms using local reward signal and count-based MRL algorithms using global reward signal.

To use local reward signals to train decentralized policy, we propose fictitious play

based algorithms using the count representation in $\mathbb{C}$Dec-POMDPs. We show that the individual value function can be estimated by sampled values of the counts and average rewards. Then we show that we can aggregate policy updates of agents with same state-action into a count-based policy gradient computation. Similar to other fictitious play based algorithms [133], due to the properties inherited from fictitious play, our solution can be also considered as an approximation to the equilibrium in the non-cooperative setting.

When local reward is not available, we train the critic (the value function approximator) by the global reward. Then to compute the gradient of the decentralized policy, instead of directly using the global critic, we use its first order Taylor approximation. By showing that first order Taylor approximation of the critic is factored amongst agents, we propose an efficient policy gradient computation.

Area	**Contributions**	**Main techniques**
Representation for collective planning	Representing collective planning problems using the count $\mathbb{C}$Dec-POMDPs[78, 76]	Collective distribution and sufficient statistics of the count
Collective reinforcement learning algorithms	Scalable collective factored policy gradient methods [76, 77]	Fictitious play, exchangeability theorem, Taylor approximation

Table 1.1: Summary of contributions in collective multi-agent decision making

The relation between the components in our framework is demonstrated by the diagram in Figure 1.4. Two new classes of RL algorithms, one with local reward and another with global reward, are proposed based on the factorization form of actor and critic function in $\mathbb{C}$Dec-POMDP. The whole framework is developed based on the novel count representation in collective planning.

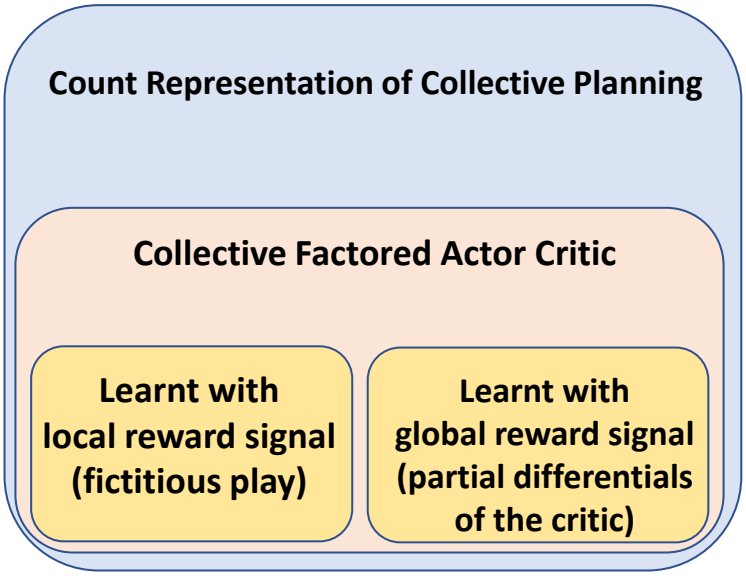


Figure 1.4: Summary of Framework

1.3 Thesis structure

The structure of my thesis is shown in Figure 1.5. After the introduction, in Chapter 2, we formulate the $\mathbb{C}$Dec-POMDP model and propose the new representation of $\mathbb{C}$Dec-POMDP planning problems using the count variables. In Chapter 3, we introduce collective graphical model (CGM) as a predecessor of our $\mathbb{C}$Dec-POMDP. We discuss the connection between collective planning in $\mathbb{C}$Dec-POMDP and collective inference in CGM. In Chapter 4, we define collective multi-agent reinforcement learning problems and the related works in multi-agent reinforcement learning literature. We highlight a major challenge in multi-agent reinforcement learning which is the *credit-assignment* for joint action. To address credit-assignment in collective planning domain, in Chapter 4, we show the factorization of collective policy gradient. The factorization of collective policy gradient is the principle we use to derive collective reinforcement learning algorithms based on local rewards in Chapter 5 and global rewards in Chapter 6. We summarize the main ideas of the thesis and propose future directions in Chapter 7.

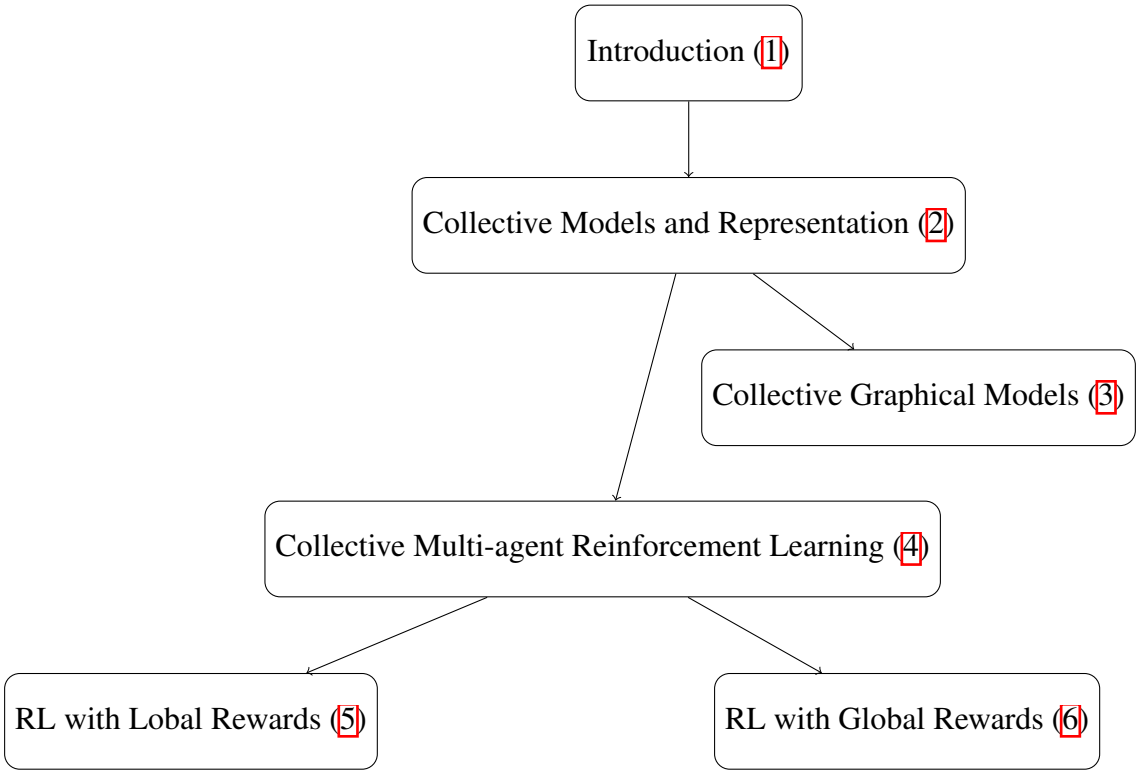


Figure 1.5: Chapter dependencies. Included in (·) are chapter numbers with hyperlink.

Chapter 2

Representation of Collective Planning

In this chapter, we introduce the collective decentralized (PO)-MDPs (ℂDec-POMDP) framework to model multi-agent systems (MAS) where the transition and reward of each individual agent depends on the number (count values) of agents in different local states. First, we show examples of ℂDec-POMDPs in different multi-agent domains, e.g. taxi supply-demand matching, grid navigation, and patrolling (in Section 2.1). Then we formally define the ℂDec-POMDP model in Section 2.2. In Section 2.3, we show that count variables are sufficient statistics for planning in ℂDec-POMDPs. It implies that we can re-write the value functions of a ℂDec-POMDP with respect to count variables instead of state-action trajectory of agents. By developing the collective distribution of the counts, we propose an efficient count sampling procedure to simulate the dynamics of the collective system in Section 2.3.1.

2.1 Motivation

2.1.1 Taxi Supply Demand problem

We now present a motivating application for $\mathbb{C}$Dec-POMDPs based on the taxi supply demand problem in a zonal city. This is based on the problem introduced in [133]. Figure 2.1 shows the map of Singapore divided into different zones. Our objective is to optimize taxi agent policies to maximize the total profit of the taxi fleet. Such a setting is useful in the case of autonomous taxi fleet operations for revenue maximization. We next describe a taxi agent's decision making process. At time t, a taxi agent observes its current location in a zone z and also the count of other taxis in zone z. The agent has two actions: decide to *stay* in the zone to look for passengers or *move* to another zone (one of 80 other zones). If the agent stays in the current zone, its probability of picking up a passenger is dictated by the ratio between the current demand and the count of other taxis in the zone. If the demand is higher than the number of taxis, then the agent picks up a passenger with a probability close to 1, else the probability is smaller than 1 (based on the ratio of taxis and the current demand). If the agent picks up a passenger, it moves to the passenger's intended destination. Such transition probabilities can be encoded into the transition function of the $\mathbb{C}$Dec-POMDP. The reward an agent gets upon picking a passenger is the total profit of the trip (trip payment minus the fuel cost of moving). If the drive moves to another zone (without a passenger), it incurs the fuel cost for moving.

In this domain, the reward and transition function of each taxi agent is defined by the aggregate count value rather than some specific identity.

Figure 2.1: Taxi navigation in zonal map.

2.1.2 Goal oriented robot navigation

Another domain studied in this thesis is the goal oriented robot navigation. This domain is known as Grid World Problem [118] and has been a testbed for many reinforcement learning algorithms including MAS planning [132, 66]. In this domain, a team of robots try to move in a grid map toward some goal locations. A goal can be representative of a victim in a disaster rescue scenario or some object to be picked-up in a transportation task. Figure 2.2 shows an example of 3 robots trying to reach a single goal in a 4×4 grid. A robot receives a constant reward whenever it reaches the goal. Corridors in the grid are narrow, so when there are many robots crossing the same corridor, the transition probability for each robot being able to reach the next location drops dramatically. In other words, the transition function of each robot depends on the number of robots taking the same corridor. In this goal oriented domain, the objective is to maximize the number of times robots reach the goal.

2.1.3 Police Patrolling

We consider the police patrolling problem introduced in [21]. A team of homogeneous police personnel are stationed in predefined geographic regions to be ready to respond to incidents. A central communications command unit is responsible for receiving emergency calls and dispatching police cars. Responding cars are cho-

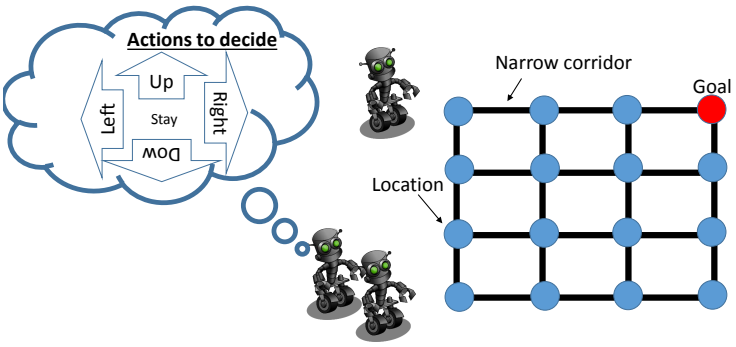


Figure 2.2: Robot navigation toward single goal (red location).

sen to minimize the travel time to the incident location. A police car dispatched to attend an incident would only come back to their locations after a certain of time including traveling and engagement time. When some car leave their stations in critical zones to go to incidents, it is necessary for *free* police cars in nearby stations to re-allocate to be able to quickly respond to impending emergency. Instead of a centralized police re-allocation, we consider autonomous police agents with a decentralized policy to make decisions on their stations in each decision epoch. An urgent incident is required to be attended within 10 minutes and a non-urgent incident is needed to be attended within 20 minutes. We want to optimize the decentralized patrolling policy to minimize number of unsatisfied incidents. To model this objective, we consider the penalty -10 whenever the response time requirement of an incident is not met and 0 otherwise.

Similar to the zonal setting in the taxi domain, we also consider a finite number of locations where police presence can be. However, different from the taxi and robot navigation domains, the number of active agents to make decision is not constant. In other words, only free agents are able to move to new locations. Busy agents would have to come back from all assigned incidents before changing stations. To model this problem with $\mathbb{C}$Dec-POMDPs, we extend the local state space of an agent by including the time it would take to become free.

2.2 Collective Decentralized POMDP (ℂDec-POMDP) framework

Table 2.1: Table of Notation

M	$\triangleq$	Number of agents
m	$\triangleq$	Agent m
H	$\triangleq$	Planning horizon
S	$\triangleq$	State space of an agent
A	$\triangleq$	Action space of an agent
$s_t^m \in S$	$\triangleq$	State of agent m at time period t
$\boldsymbol{s}_t = \langle s_t^m \rangle_m$	$\triangleq$	Joint state of all agents at time period t
$a_t^m \in S$	$\triangleq$	Action of agent m at time period t
$\boldsymbol{a}_t = \langle a_t^m \rangle_m$	$\triangleq$	Joint action of all agents at time period t
$d_t \in D$	$\triangleq$	Global state component at time period t, e.g to model taxi demand or victim location
$o_t^m \in S$	$\triangleq$	Local observation of agent m at time period t including current local state $s_t^m = i$ of the agent and its local view of the global state
$\mathbb{I}_t^m(i) \in \{0,1\}$	$\triangleq$	if agent m is at state i at time t or $s_t^m = i$
$\mathbb{I}_t^m(i,j) \in \{0,1\}$	$\triangleq$	if agent m takes action j in state i at time t or $(s_t^m, a_t^m) = (i,j)$
$\mathbb{I}_t^m(i,j,i') \in \{0,1\}$	$\triangleq$	if agent m takes action j in state i at time t and transitions to state i' or $(s_t^m, a_t^m, s_{t+1}^m) = (i,j,i')$
$\mathrm{n}_t^{\mathrm{s}}(i) \in [0;M]$	$\triangleq$	Number of agents at state i at time t
$\mathrm{n}_t^{\mathrm{sa}}(i,j) \in [0;M]$	$\triangleq$	Number of agents at state i taking action j at time t
$\mathrm{n}_t^{\mathrm{sas'}}(i,j,i') \in [0;M]$	$\triangleq$	Number of agents at state i taking action j at time t and transitioning to state i' at time $t+1$
$\mathrm{n}_t^{\mathrm{s}}$	$\triangleq$	Count table $(\mathrm{n}_t^{\mathrm{s}}(i)\ \forall i \in S)$
$\mathrm{n}_t^{\mathrm{sa}}$	$\triangleq$	Count table $(\mathrm{n}_t^{\mathrm{sa}}(i,j)\ \forall i \in S, j \in A)$
$\mathrm{n}_t^{\mathrm{sa}}(i)$	$\triangleq$	Count table $(\mathrm{n}_t^{\mathrm{sa}}(i,j)\ \forall j \in A)$
$\mathrm{n}_t^{\mathrm{sas'}}$	$\triangleq$	Count table $(\mathrm{n}_t^{\mathrm{sas'}}(i,j,i')\ \forall i,i' \in S, j \in A)$
$\mathrm{n}_t^{\mathrm{sas'}}(i,j)$	$\triangleq$	Count table $(\mathrm{n}_t^{\mathrm{sas'}}(i,j,i')\ \forall i' \in S)$
$o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)$	$\triangleq$	Local observation of agent m regard to global count $\mathrm{n}_t^{\mathrm{s}}$ when it is at state i at time period t
$\pi_t^m(j \mid i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$	$\triangleq$	Probability of agent m taking action $j \in A$ at state $i \in S$, time t given local observation $o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)$.
$\phi_t^m\left(i' \mid i, j, \mathrm{n}_t^{\mathrm{s}}\right)$	$\triangleq$	Transition probability of agent m from state i after taking action j to state i' conditioning on global count $\mathrm{n}_t^{\mathrm{s}}$.
$r(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r(\mathrm{n}_t^{\mathrm{sa}}, d_t)$	$\triangleq$	Global reward of system given the joint state-action input $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$.

We now formally define $\mathbb{C}$Dec-POMDP as a class of decentralized multi-agent model where agent transition and reward functions are dependent on only the aggregate variables. In $\mathbb{C}$Dec-POMDP, the identity of an agent is not important and can be marginalized out with the counts. The framework of $\mathbb{C}$Dec-POMDP consists of the following:

- A finite planning horizon H.
- The number of agents M. An agent m can be in one of the states in the state space S. We denote a single state as $i \in S$.
- A set of action A for each agent m. We denote an individual action as $j \in A$.
- $\boldsymbol{s}_t, \boldsymbol{a}_t$ denote the joint state and joint action of agents at time t
- We consider a global state component $d \in D$. The joint state space is $\times_{m=1}^{M} S \times D$.
- Let $(s_{1:H}, a_{1:H})^m = (s_1^m, a_1^m, s_2^m \ldots, s_H^m, a_H^m)$ denote the complete state-action trajectory of an agent m. We denote the state and action of agent m at time t using random variables s_t^m, a_t^m. Different indicator functions $\mathbb{I}_t(\cdot)$ are defined in Table 2.1. We define the following count given the trajectory of each agent $m \in M$:
  - $\mathrm{n}_t^{\mathrm{sas}'}(i,j,i') = \sum_{m=1}^{M} \mathbb{I}_t^m(i,j,i')\ \forall i, i' \in S, j \in A$
  - $\mathrm{n}_t^{\mathrm{sa}}(i,j) \ = \sum_{m=1}^{M} \mathbb{I}_t^m(i,j) \qquad \forall i \in S, j \in A$
  - $\mathrm{n}_t^{\mathrm{s}}(i) \ \ = \sum_{m=1}^{M} \mathbb{I}_t^m(i)\ \forall i \in S$

  As noted in Table 2.1, count $\mathrm{n}_t^{\mathrm{sa}}(i,j)$ denotes the number of agents in state i taking action j at time step t; other counts are interpreted analogously. We denote count tables as $\mathrm{n}_t^s = (\mathrm{n}_t^{\mathrm{s}}(i)\ \forall i \in S)$ and $\mathrm{n}_t^{\mathrm{sa}} = (\mathrm{n}_t^{\mathrm{sa}}(i,j)\ \forall i \in S, j \in A)$; table $\mathrm{n}_t^{\mathrm{sas}'}$ is defined analogously for the time step t as shown in Table 2.1.

- We assume a general partially observable setting wherein agents can have different observations based on the collective influence of other agents. An agent observes its local state s_t^m. In addition, it also observes o_t^m at time t based on its local state s_t^m, the count table $\mathrm{n}_t^{\mathrm{s}}$, and the global component d_t. E.g., an agent m in state i at time t can observe the count of other agents also in state i (=$\mathrm{n}_t^{\mathrm{s}}(i)$) or other agents in some neighborhood of the state i (=$\{\mathrm{n}_t^{\mathrm{s}}(j)\ \forall j \in \mathrm{Nb}(i)\}$). Without loss of generality, we consider the deterministic observation function $o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)$ outputting the same local observation for all agents in the same state i. To handle stochastic case with different possible observations in the same state, we can extend the state space to include the observation o^i which agent receives in a state. The state count is extended to record the number $\mathrm{n}_t(i, o^i)$ of agents in a specific state i and receiving a same observation o^i.

- The local transition function of an agent m is $P_l\big(s_{t+1}^m | s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t\big)$. The transition function is the same for all the agents. Notice that it is affected by $\mathrm{n}_t^{\mathrm{sa}}$, which depends on the collective behavior of the agent population.

- The transition function of the global component d is $P_g(d_{t+1}|d_t, \mathrm{n}_t^{\mathrm{sa}})$. Notice that the global component is also affected by state-action count table.

- Each agent m has a non-stationary policy $\pi_t^m(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ denoting the probability of agent m to take action j given its observation $(i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ at time t. We denote the policy over planning horizon of an agent m to be $\pi^m = (\pi_1^m, \ldots, \pi_H^m)$. When agents have the same policy, we denote the common policy with π.

- A reward $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r(\mathrm{n}_t^{\mathrm{sa}}, d_t)$ is produced for each joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$. Notice that the reward function depends on the collective variables n^{sa}.

- Initial state distribution, $b_o = (P(i) \forall i \in S)$, is same for all agents.

- Initial distribution over global component is $b_o^g(d) \forall d$.

In the $\mathbb{C}$Dec-POMDP model, agent identities do not matter; different model components are only affected by agent's local state-action, and a statistic of other agents' states-actions. We define the global component d to model the external variable besides agents' local states. In taxi domain, d can be used to model passenger demands. In patrolling domain, d can be used to model the victim or incident occurrence.

The joint-state transition probability is:

$$P(\boldsymbol{s}_{t+1}, d_{t+1} | \boldsymbol{s}_t, d_t, \boldsymbol{a}_t) = P_g(d_{t+1} | d_t, \mathrm{n}_t^{\mathrm{sa}}) \cdot \prod_{m=1}^{M} P_l\big(s_{t+1}^m | s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) \tag{2.1}$$

Such an expression conveys that only the statistic $\mathrm{n}_t^{\mathrm{sa}}$ of the joint state-action, the global value d and an agent's local state-action are sufficient to predict the agent's next state.

We assume a decentralized and partially observable setting in which each agent receives only a partial observation about the environment. Let the current joint-state be $(\boldsymbol{s}_t, d_t)$ after the last join-action, then the local observation for agent m is given using the function $o_t(\boldsymbol{s}_t^m, d_t, \mathrm{n}_t^{\mathrm{s}})$. Agents in different states will get different partial observation about the environment. An agent decides its action a^m based on its local observation as π

We consider a general definition of the function $r(\boldsymbol{s}_t, d_t, \boldsymbol{a}_t)$. In domains like taxi navigation, this reward can be decomposed into a sum of local rewards of agents $r(\boldsymbol{s}_t, d_t, \boldsymbol{a}_t) = \sum_m r(\boldsymbol{s}_t^m, \boldsymbol{a}_t^m, d_t, \mathrm{n}_t^{\mathrm{sa}})$ where $r(\boldsymbol{s}_t^m, \boldsymbol{a}_t^m, d_t, \mathrm{n}_t^{\mathrm{sa}}$ is the local reward for individual agent m, which depends on the agent's local state-action and collective variables. Given that the reward function is the same for all the agents, we can further simplify it as $\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) r(i,j,d_t,\mathrm{n}_t^{\mathrm{sa}})$, where $\mathrm{n}_t^{\mathrm{sa}}(i,j)$ is the number of agents in state i and taking action j given the joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$. The algorithms

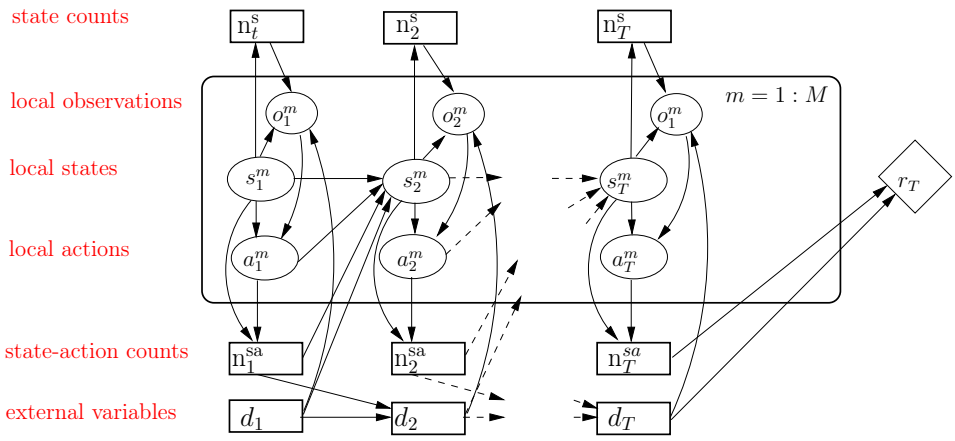


Figure 2.3: DBN for T-step reward for $\mathbb{C}$Dec-POMDP with external variables

for decomposable rewards are studied in Chapter 5 and ones for non-decomposable rewards are studied in Chapter 6 later.

The dynamic Bayesian Network (DBN) for reward collected at T^{th}step in $\mathbb{C}$Dec-POMDP is demonstrated with plate notation in Figure 2.3.

Agent type: The above defined model components can also differentiate among agents by using the notion of *agent types*, which can be included in the state space. In the extreme case, each agent would be of a different type representing a fairly general multiagent planning problem. However, the main benefit of the model lies in settings when agent types are much smaller than the number of agents.

Policy and value function: We consider a finite-horizon problem with H time steps. Each agent has a non-stationary reactive policy that takes as input agent's current state i and the observation o, and outputs the probability of the next action j as $\pi_t^m(j|i,o)$. Let $\pi = \langle \pi^1, \ldots, \pi^M \rangle$ denote the joint-policy.

In $\mathbb{C}$Dec-POMDPs, we consider the goal to find the homogeneous policy π to maximize the total rewards over planning horizon H

$$\max_{\pi} V(\pi) = \sum_{t=1}^{H} \mathbb{E}[r_t | b_o, b_o^g, \pi] \tag{2.2}$$

Average flow approximation

Our model is motivated by the decentralized stochastic planning model (D-SPAIT) for anonymous agents proposed in [131], and the framework of congestion games [68]. In our work, we explicitly model the distribution over counts $n(\cdot)$ of individuals and use this distribution as the basis for planning. In contrast, the D-SPAIT model is based on the concept of approximating the planning problem using *expected counts* of agents. Intuitively, if $\mathbb{E}[f(\boldsymbol{n})]$ denotes the planning objective over counts $\boldsymbol{n}$, then D-SPAIT model approximates this objective as $f\big(\mathbb{E}[\boldsymbol{n}]\big)$. Table 2.4a show the computation of such average flow; $x_{s_t}(i)$ denotes the expected number of agents in state i at time t and Figure 2.4b shows DBNs for D-SPAIT model. Computing policies based on such average flow leads to inaccurate estimation of the true objective function and lower quality policies, as we also demonstrate empirically.

$$
\begin{aligned}
x_{s_1}(i) &= M \times P(i), && \forall i \in S \\
x_{s_t a_t}(i,j) &= x_{s_t}(i) \times \pi(j|i, x_{s_t}(i)) && \forall i \in S, j \in A \\
x_{s_{t+1}}(i') &= \sum_{i,j} x_{s_t a_t}(i,j) \phi_t(i'|i,j,x_{s_t}(i)) && \forall i' \in S
\end{aligned}
$$

(a) Average approximation of agent flow in D-SPAIT model

(b) Deterministic Markov chain for T-step reward

Figure 2.4: D-SPAIT model

2.2.1 Policy representation

The benefit of models such as D-SPAIT and $\mathbb{C}$Dec-POMDPs lies when the agent population is large, and the agent identity does not affect the reward or the transition function. E.g., in the taxi fleet operation optimization problem discussed earlier

such aggregate interactions occur. Given a large number of taxis ($\approx$ 8000), it is infeasible to compute a unique policy for each taxi. Therefore, similar to the D-SPAIT model, our goal is to compute a homogenous policy π for all the agents. As the policy is dependent on counts n_t, it allows for an expressive class of policies.

We define an *open loop* policy as a policy where action selection only depends on the current local state of the agent without any dependence on the count information. In a *closed loop* policy, action selection depends on counts also in addition to the agent's local state. Our proposed model free algorithm developed in the following sections can train both open and closed loop policies, whereas previous average flow based approaches are limited to open loop policy optimization.

Neural network policy

The complexity of closed loop policy would quickly increase when agent observation $o_t^m(i, \boldsymbol{n}_t^{\mathrm{s}}, d_t)$ include not only the count of its current location $\mathrm{n}_t^{\mathrm{s}}(i)$ but neighboring locations $\{\mathrm{n}_t^{\mathrm{s}}(j)\}_{j\in N(i)}$. In this case, we can consider the policy function to be neural network $\pi_t^m : O_t^m \to \Omega(A)$ which takes the input to be possible observation vector $o_t^m \in O_t^m$ and output the action probability $\pi_t^m(o_t^m) = \langle \pi_t^m(j|o_t^m)\rangle_{j\in A} \in \Omega(A)$ with $\Omega(A)$ to be the probability space where $\sum_{j\in A} \pi_t^m(j|o_t^m) = 1$. To ensure the output of policy function to be valid probabilities, we consider the common technique to apply the soft-max normalization for output $\boldsymbol{z} = \langle z_j\rangle_{j\in A}$

$$\sigma(\boldsymbol{z})_j = \frac{\exp(z_a)}{\sum_{j'\in A}\exp(z_{j'})}$$

An example of a simple policy function is illustrated in Figure 2.5. .

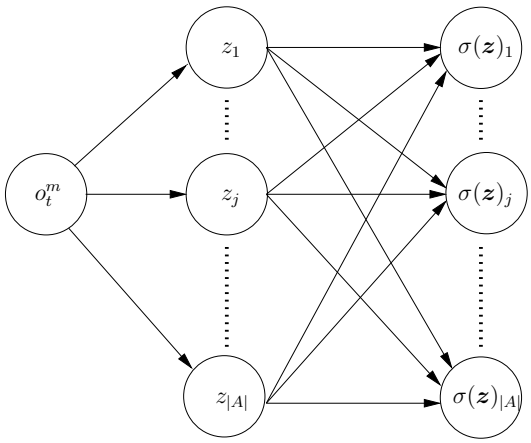


Figure 2.5: Simple policy function in which each $z_j = \theta_j \times o_t^m$ is a linear transformation of the input o_t^m and the output is the soft-max normalization. This is known as shadow or no-hidden layer neural network.

2.3 Count-based representation of $\mathbb{C}$Dec-POMDP

We now establish several basic properties of the $\mathbb{C}$Dec-POMDP model. For a fixed population M, let $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) = \{(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})^m \ \forall m\}$ denote the state-action trajectories of different agents sampled from the DBN in Figure 2.3. Let $\mathbf{n}_{1:T}=\{(\mathrm{n}_t^{\mathrm{s}}, \mathrm{n}_t^{\mathrm{sa}}, \mathrm{n}_t^{\mathrm{sas'}} \ \forall t = 1:T\}$ be the combined vector of the resulting count tables for each time step t. We first show that this combined vector is sufficient statistics in $\mathbb{C}$Dec-POMDP.

Theorem 2.1. *Count tables $\mathbf{n}_{1:T}$ are the sufficient statistic for a sample of M state-action trajectories from the $\mathbb{C}$Dec-POMDP graphical model in Figure 2.3.*

Proof. Let $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T})$ denote the joint trajectory. The joint-distribution $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}; \pi)$ is defined as:

$$= b_o^g(d) \prod_{t=1}^{T-1} P_g(d_{t+1}|d_t, \mathrm{n}_t^{\mathrm{sa}}) \prod_{m=1}^{M} \Bigg[\prod_{i \in S} P(i)^{\mathbb{I}_t^m(i)} \prod_{t=1}^{T-1} \prod_{i,j,i'} \Big[\pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))^{\mathbb{I}_t^m(i,j)} P_l(i'|i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t)^{\mathbb{I}_t^m(i,j,i')} \Big] \prod_{i,j} \pi_T(j|i, o(i, \mathrm{n}_T^{\mathrm{s}}, d_T))^{\mathbb{I}_t^m(i,j)} \Bigg]$$

We can simplify the above expression by grouping together terms from all the

agents. The resulting expression $f(\mathbf{n}_{1:T}, d_{1:T}; \pi)$ depends only on counts $\mathbf{n}_{1:T}$ as:

$$
\begin{aligned}
&f(\mathbf{n}_{1:T}, d_{1:T}; \pi) \\
&= b_o^g(d) \prod_{t=1}^{T-1} P_g(d_{t+1}|d_t, \mathrm{n}_t^{\mathrm{sa}}) \prod_{i \in S} P(i)^{\mathrm{n}_1^{\mathrm{s}}(i)} \prod_{t=1}^{T-1} \prod_{i,j,i'} \Big[\pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \\
&P_l(i'|i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t)^{\mathrm{n}_t^{\mathrm{sas'}}(i,j,i')} \Big] \prod_{i,j} \pi_T(j|i, o(i, \mathrm{n}_T^{\mathrm{s}}, d_T))^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \qquad (2.3)
\end{aligned}
$$

Thus, count tables $\mathbf{n}_{1:T}$ are the sufficient statistic for the population sample as the joint-probability $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}; \pi)$ is a function of counts $\mathbf{n}_{1:T}$. □

We next define a distribution directly over the count tables $\mathbf{n}_{1:T}$ as below:

Theorem 2.2. *The distribution $P(\mathbf{n}_{1:T}, d_{1:T}; \pi)$ is defined as:1*

$$P(\mathbf{n}_{1:T}, d_{1:T}; \pi) = h(\mathbf{n}_{1:T}) f(\mathbf{n}_{1:T}, d_{1:T}; \pi) \qquad (2.4)$$

where $f(\mathbf{n}_{1:T}, d_{1:T}; \pi)$ is given in (2.3). The function $h(\mathbf{n}_{1:T})$ counts the total number of ordered M state-action trajectories with sufficient statistic equal to $\mathbf{n}$, given as:

$$
\begin{aligned}
h(\mathbf{n}_{1:T}) = &\frac{M!}{\prod_i \mathrm{n}_1^{\mathrm{s}}(i)!} \prod_{t=1}^{T-1} \prod_{i \in S} \left[\frac{\mathrm{n}_t^{\mathrm{s}}(i)!}{\prod_{j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)!} \frac{\mathrm{n}_t^{\mathrm{sa}}(i,j)!}{\prod_{i' \in S} \mathrm{n}_t^{\mathrm{sas'}}(i,j,i')!} \right] \\
&\times \prod_{i \in S} \left[\frac{\mathrm{n}_T^{\mathrm{s}}(i)!}{\prod_{j \in A} \mathrm{n}_T^{\mathrm{sa}}(i,j)!} \right] \mathbb{I}[\mathbf{n}_{1:T} \in \Omega_{1:T}] \qquad (2.5) \\
&\qquad (2.6)
\end{aligned}
$$

Set $\Omega_{1:T}$ is the set of all allowed consistent count tables as:

$$\sum_{i\in S} \mathrm{n}_t^{\mathrm{s}}(i) = M \ \forall t \ ; \sum_{j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) = \mathrm{n}_t^{\mathrm{s}}(i) \ \forall j, \forall t \tag{2.7}$$

$$\sum_{i'} \mathrm{n}_t^{\mathrm{sas}'}(i,j,i') = \mathrm{n}_t^{\mathrm{sa}}(i,j) \ \forall i \in S, j \in A, \forall t$$

$$\sum_{i,j} \mathrm{n}_t^{\mathrm{sas}'}(i,j,i') = \mathrm{n}_{t+1}^{\mathrm{s}}(i') \ \forall i' \in S, \forall t \tag{2.8}$$

Proof. For any *invalid* count values $\mathbf{n}_{1:T} \notin \Omega_{1:T}$, there is no realization of joint trajectory possessing the invalid count.

We prove the expression for $\mathbf{n}_{1:T} \in \Omega_{1:T}$ as follows :

$$\begin{aligned} P(\mathbf{n}_{1:T}, d_{1:T}; \pi) &= \sum_{\langle \boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T} \rangle \sim \mathbf{n}_{1:T}} P(\langle \boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T} \rangle) \\ &= \sum_{\langle \boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T} \rangle \sim \mathbf{n}_{1:T}} f(\mathbf{n}_{1:T}, d_{1:T}; \pi) \qquad (2.9) \\ &= h(\mathbf{n}_{1:T}) f(\mathbf{n}_{1:T}, d_{1:T}; \pi). \qquad (2.10) \end{aligned}$$

We prove the expression (2.5) for $h(\mathbf{n}_{1:T})$ by induction:

- When $T = 1$, (2.5) holds as $h(\mathbf{n}_1) = \frac{M!}{\prod_{i,j} \mathrm{n}_1^{\mathrm{sa}}(i,j)}$ is the total number of combinations to assign M individuals to $|S| \times |A|$ possibilities of state-action. The $h(\mathbf{n}_1)$ is equivalent to multinomial coefficient of distribution of M individuals to $|S| \times |A|$ possibilities.
- Assume that (2.5) holds for T. Given a joint trajectory $\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}$ satisfying the count table $\mathbf{n}_{1:T}$, the total number of possible joint value $\boldsymbol{s}_{T+1}\boldsymbol{a}_{T+1}$ satisfying the count table $\mathbf{n}_{T+1}$ is

$$\Big(\prod_{i\in S, j\in A} \frac{\mathrm{n}_T^{\mathrm{sa}}(i,j)!}{\prod_{i'\in S} \mathrm{n}_T^{\mathrm{sas}'}(i,j,i')!} \Big) \Big(\prod_{i\in S} \frac{\mathrm{n}_{T+1}^{\mathrm{s}}(i)!}{\prod_{i'\in S, j\in A} \mathrm{n}_{T+1}^{\mathrm{sa}}(i,j)!} \Big) \tag{2.11}$$

Notice that in (2.11), each expression of $i \in S$ in first term is a multinomial coefficient of distribution of $\mathrm{n}_T^{\mathrm{sa}}(i,j)$ individuals into $|S|$ possibilities of next state i' to satisfying the count $\left(\mathrm{n}_T^{\mathrm{sas}'}(i,j,i'), \forall i'\right)$; similarly, each expression of $i \in S$ in second term is a mutinomial coefficient of distribution of $\mathrm{n}_{T+1}^{\mathrm{s}}(i)$ individuals in state i at time step $T+1$ into $|A|$ possibilities of action j. Multiplying (2.11) with $h(\mathbf{n}_{1:T})$ shows that the expression of $h(\mathbf{n}_{1:T+1})$ as in (2.5) holds for $T+1$, which completes the proof.

□

One corollary of theorem 2.2 is we can decompose the collective distribution of the count variables as

Corollary 2.1. *The collective distribution can be represented by*

$$P(\mathbf{n}_{1:T}, d_{1:T}; \pi) = P(\mathrm{n}_T^{\mathrm{sa}} \mid \mathrm{n}_T^{\mathrm{s}}, d_T) b_o^g(d) P(\mathrm{n}_1^{\mathrm{s}})$$
$$\prod_{t=1:T-1} P_g(d_{t+1}|d_t, \mathrm{n}_t^{\mathrm{sa}}) P(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, d_t) P(\mathrm{n}_t^{\mathrm{sas}'} \mid \mathrm{n}_t^{\mathrm{sa}}, d_t) \mathbb{I}[\mathbf{n}_{1:T} \in \Omega_{1:T}],$$

in which

$$P(\mathrm{n}_1^{\mathrm{s}}) = \frac{M!}{\prod_i \mathrm{n}_1^{\mathrm{s}}(i)!} \prod_{i \in S} b_o(i)^{\mathrm{n}_1^{\mathrm{s}}(i)} \tag{2.12}$$

$$P(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, d_t) = \prod_{i \in S} \left(\frac{\mathrm{n}_t^{\mathrm{s}}(i)!}{\prod_{j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)!} \prod_{j \in A} \pi_t(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}}))^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \right) \tag{2.13}$$

$$P(\mathrm{n}_t^{\mathrm{sas}'} \mid \mathrm{n}_t^{\mathrm{sa}}, d_t) = \prod_{i \in S, j \in A} \left(\frac{\mathrm{n}_t^{\mathrm{sa}}(i,j)!}{\prod_{i' \in S} \mathrm{n}_t^{\mathrm{sas}'}(i,j,i')!} \prod_{i' \in S} P_l(i'|i,j,\mathrm{n}_t^{\mathrm{sa}}, d_t)^{\mathrm{n}_t^{\mathrm{sas}'}(i,j,i')} \right) \tag{2.14}$$

The Bayesian graphical model of the collective distribution is shown in Figure 2.6.

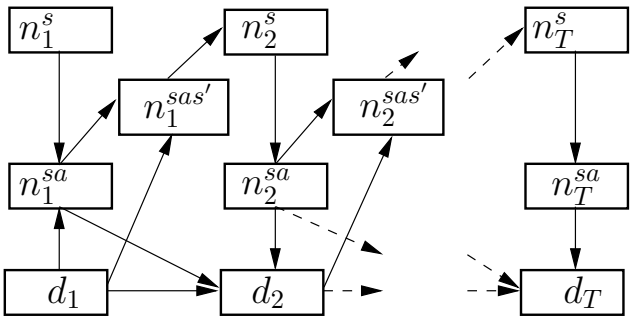


Figure 2.6: Generative model of the counts in $\mathbb{C}$Dec-POMDP

2.3.1 Count Sampling Process

Originally, as a summary of joint trajectory, the count variables are obtained by aggregating values of individual variables. However, sampling individual values would be computationally expensive in large populations. Fortunately, the collective distribution of the counts shown in corollary 2.1 can provide us a way to *directly* sample the count values instead of aggregating the individual variables. As generative model of the count is a Bayesian network (in Figure 2.6), we can generate the values of the counts by forward sampling from state-count to action count and then transition count.

Algorithm 1 provides the pseudo code to generate the count samples for H time periods. The state count in the first period is sampled by the multinomial distribution with a population size to be M and probabilities to be the initial distribution b_o (line 2). At each time period t, we sample action counts for agents at each local state i by multinomial distribution with population size $\mathrm{n}_t^{\mathrm{s}}(i)$ and probabilities $\pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ (line 5). Analogously, to simulate the effect of the joint action counts into the environment, we can sample the transition count for agents taking action j from the local state i by multinomial distribution with population size $\mathrm{n}_t^{\mathrm{sa}}(i, j)$ and probabilities $P_l(i'|i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t)$ (line 6).

Algorithm 1: Collective Sampling Algorithm

Algorithm C-SAMPLING()

- Sampling $\mathrm{n}^{\mathrm{s}}_1 \sim \mathrm{Mul}(M, b_o)$
- Sampling $d_1 \sim b^g_o$
- **for** $t \leftarrow 1$ ***to*** H **do**
  - Sampling state-action counts: $\mathrm{n}^{\mathrm{sa}}_t(i, \bullet) \sim \mathrm{Mul}(\mathrm{n}^{\mathrm{s}}_t(i), \pi_t(\bullet|i, o(i, d_t, \mathrm{n}^{\mathrm{s}}_t))), \forall i \in S$
  - Sampling transition counts: $\mathrm{n}^{\mathrm{sas}'}_t(i, j, \bullet) \sim \mathrm{Mul}(\mathrm{n}^{\mathrm{sa}}_t(i, j), P_l(\bullet|i, j, \mathrm{n}^{\mathrm{sa}}_t, d_t)), \forall i \in S, j \in A$
  - Sampling external variables: $d_{t+1} \sim P_g(\bullet|d_t, \mathrm{n}^{\mathrm{sa}}_t)$
  - Aggregate: $\mathrm{n}^{\mathrm{s}}_{t+1}(i') = \sum_{i,j} \mathrm{n}^{\mathrm{sas}'}_t(i, j, i'), \forall i' \in S$
- **return** $\mathbf{n}_{1:H}$

2.3.2 Joint-Value Function

We next show that the joint-value for a given policy π also depends on the count vector $\mathbf{n}$. Thus, making counts as the sufficient statistic for planning in $\mathbb{C}$Dec-POMDPs.

Theorem 2.3. *The joint-value function of a policy π over horizon H given by the expectation of total rewards, $V(\pi) = \sum_{T=1}^{H} \mathbb{E}[r_T]$, can be computed by the expectation over counts as:*

$$\sum_{\mathbf{n} \in \Omega_{1:H}} P(\mathbf{n}_{1:H}, d_{1:H}; \pi) \left[\sum_{T=1}^{H} r_T(\mathrm{n}^{\mathrm{sa}}_T, d_T) \right] \tag{2.15}$$

Proof. Let $\boldsymbol{s}_T$ and $\boldsymbol{a}_T$ represent the joint-state and joint-action of all the agents at the time step T; $\mathrm{n}^{\mathrm{s}}_T$ and $\mathrm{n}^{\mathrm{sa}}_T$ represent the count vectors corresponding to $(\boldsymbol{s}_T, \boldsymbol{a}_T)$. The *immediate* reward received for this joint-state and action is $r(\boldsymbol{s}_T, \boldsymbol{a}_T, d_T) = r_T(\mathrm{n}^{\mathrm{sa}}_T, d_T)$.

We can compute the expectation of immediate reward received at time T as:

$$\mathbb{E}[r_T(\pi)]$$

$$= \sum_{(\boldsymbol{s}_{1:T-1}, \boldsymbol{a}_{1:T-1}, d_{1:T-1}), \boldsymbol{s}_T} P(\boldsymbol{s}_{1:T-1}, \boldsymbol{a}_{1:T-1}, d_{T-1}) P(\boldsymbol{s}_T | \boldsymbol{s}_{T-1}, \boldsymbol{a}_{T-1}) r_T(\boldsymbol{s}_T, d_T; \pi) \tag{2.16}$$

using $\mathbb{C}$Dec-POMDP distribution, we have:

$$= \sum_{(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}) \sim \mathbf{n}_{1:T}, d_{1:T}} f(\mathbf{n}_{1:T}, d_{1:T}) r_T(\mathrm{n}_T^{\mathrm{sa}}, d_T) \tag{2.17}$$

Notice that in the above expression, the expected immediate reward at time step T only depends on the counts $\mathrm{n}_T^{\mathrm{sa}}$ that arise from the joint state and action $(\boldsymbol{s}_T, \boldsymbol{a}_T)$. Similar to equations (2.9) and (2.10), instead of summing over all the joint state-action trajectories $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T})$, we can sum over the space of all possible counts vectors $\mathbf{n}_{1:T}$ multiplied by the total number of joint trajectories satisfying the corresponding counts vector $\mathbf{n}_{1:T}$, which results in the following expression:

$$\mathbb{E}[r_T(\pi)] = \sum_{d_{1:T}} \sum_{\mathbf{n}_{1:T} \in \Omega_{1:T}} h(\mathbf{n}_{1:T}) f(\mathbf{n}_{1:T}, d_{1:T}) \left[\sum_{\mathrm{n}_T^{\mathrm{sa}} \sim \mathrm{n}_T^{\mathrm{s}}} P(\mathrm{n}_T^{\mathrm{sa}} \mid \mathrm{n}_T^{\mathrm{sa}}, d_T) r_T(\mathrm{n}_T^{\mathrm{sa}}, d_T) \right] \tag{2.18}$$

Using the above expression, the value function can be computed as:

$$V(\pi) = \sum_{T=1}^{H} \mathbb{E}[r_T] = \sum_{T=1}^{H} \sum_{\mathbf{n}_{1:T} \in \Omega_{1:T}, d_{1:T}} P(\mathbf{n}_{1:T}, d_{1:T}) r_T(\mathrm{n}_T^{\mathrm{sa}}, d_T) \tag{2.19}$$

□

Our goal in $\mathbb{C}$Dec-POMDP is to compute the policy π that maximizes (2.15). Notice that the set of all the allowed counts $\Omega_{1:H}$ is combinatorially large, making the exact policy evaluation infeasible. Therefore, our approach would be to use a sampling based approach that can evaluate, and also optimize the policy π.

Furthermore, we can define the joint state-action value function as a function of

joint state-action counts $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{T=t}^{H} \mathbb{E}[r_T | \, \mathrm{n}_t^{\mathrm{sa}}, d_t]$:

Theorem 2.4. *The joint state-action value function in $\mathbb{C}$Dec-POMDP is defined by*

$$Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t) = r_t(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \sum_{\mathrm{n}_t^{\mathrm{sas}'}, \mathrm{n}_{t+1}^{\mathrm{sa}} \in \Omega_{t+1}, d_{t+1}} P(d_{t+1} | \, \mathrm{n}_t^{\mathrm{sa}}, d_t) \times P(\mathrm{n}_t^{\mathrm{sas}'} \, | \, \mathrm{n}_t^{\mathrm{sa}}, d_t) P(\mathrm{n}_{t+1}^{\mathrm{sa}} \, | \, \mathrm{n}_{t+1}^{\mathrm{s}} \sim \mathrm{n}_t^{\mathrm{sas}'}, d_{t+1}; \pi) Q_{t+1}^\pi(\mathrm{n}_{t+1}^{\mathrm{sa}}, d_{t+1}) \tag{2.20}$$

in which $P(\mathrm{n}_t^{\mathrm{sa}} \, | \, \mathrm{n}_t^{\mathrm{s}}, d_t), P(\mathrm{n}_t^{\mathrm{sas}'} \, | \, \mathrm{n}_t^{\mathrm{sa}}, d_t)$ are defined in corollary 2.1. Ω_{t+1} is the subset of consistency constraints (2.7) linking counts for time t and $t+1$

Proof. We start by the general dynamic programming equation for MDP [117] with notice $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r(\mathrm{n}_t^{\mathrm{sa}}, d_t)$

$$Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \sum_{\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1}} P(\boldsymbol{s}_{t+1}, d_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t, d_t) P(\boldsymbol{s}_{t+1}, \boldsymbol{a}_t | \boldsymbol{s}_{t+1}, d_{t+1}; \pi) Q_{t+1}^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1}) \tag{2.21}$$

Using similar arguments as in the proof of theorem 2.2, we can aggregate similar $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1} \rangle$ by $\langle \mathrm{n}_t^{\mathrm{s}}, \mathrm{n}_t^{\mathrm{sa}}, \mathrm{n}_t^{\mathrm{sas}'} \rangle$ and consider induction hypothesis $Q_{t+1}^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1}) = Q_{t+1}^\pi(\mathrm{n}_{t+1}^{\mathrm{sa}}, d_{t+1})$

$$Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r_t(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \sum_{\mathrm{n}_t^{\mathrm{sas}'}, \mathrm{n}_{t+1}^{\mathrm{sa}} \in \Omega_{t+1}, d_{t+1}} P(d_{t+1} | \, \mathrm{n}_t^{\mathrm{sa}}, d_t) \times P(\mathrm{n}_t^{\mathrm{sas}'} \, | \, \mathrm{n}_t^{\mathrm{sa}}, d_t) P(\mathrm{n}_{t+1}^{\mathrm{sa}} \, | \, \mathrm{n}_{t+1}^{\mathrm{s}} \sim \mathrm{n}_t^{\mathrm{sas}'}, d_{t+1}; \pi) Q_{t+1}^\pi(\mathrm{n}_{t+1}^{\mathrm{sa}}, d_{t+1})$$

The right hand side of above equation is an expression over counts, which defines $Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t)$. □

2.4 Related works

2.4.1 Count-based models

In many multi-agent systems (MAS), the transition and reward of each individual in the population is affected by aggregate values rather than the identity of agents. Among the most well-studied domains is the class of congestion games [68, 80], in which the pay-off function of each agent is defined only by the number of other agents traveling on the same edges. The congestion game has a wide range of applications in modeling the delay cost of road traffic flow [112, 136], and latency in package routing in communication network such as the Internet [95]. Besides the aggregate-variable pay-off function, the state transition of each agent also can be modeled as a function of aggregate values [131, 97, 113] . For example, in routing problem, after executing a moving action, high congestion level (or high number of agents presenting) in the same area could reduce the probability of agent arriving the next zone [132]. Apart from the congestion game, aggregate-variable transition functions are also defined in action graph games[47], in which the transition of an agent depends on the aggregate value of its neighbors. In the disease control domain, Robbel et al. [97] modelled the vulnerability to of a geological zone by the number of its disease-infected neighbors . In riot control, Sonu et al. [113] studied the protest intensity depending on the number of protestors and the number of police troops presenting in a location.

Our application domains in goal-oriented robot navigation and taxi supply-demand matching were studied previously by Varakantham et al. [131, 133]. We consider our domain setting similar to the ones in [133, 131]. However, our count variables provide the exact representation of $\mathbb{C}$Dec-POMDP while the average flow is an approximate representation.

2.4.2 Mean-field game theory and average flow estimations

To deal with planning problems in large population of agents, researchers in the current literature have been trying to estimate the planning problems with tractable representation. One amongst these well-studied directions is the mean-field estimation of population [146, 140] in continuous state-action space. Rooted in mean-field theory in physics to estimate the distribution of individual particles in systems, the mean-field methods quantify population behavior by the density function of distribution of agents over continuous spaces. It assumes each individual in a large population has a very small impact on the global distribution. As a result, the dynamic of the population can be represented in the form of a differential equation of continuous (flow of agents) variables. Examples of mean-field systems include fish school, ant colonies or flocks of birds [15] in nature or swarm robotics in AI [11, 103]. In domain of discrete state-action space, Varakantham et al. [133, 131] explored a similar idea with mean-field by estimating average flows of agents into each discrete state-action in a finite Markov Decision Process (MDP). By using the average flow estimation, multi-agent planning problems are re-formulated as network flow problems with non-linear flow splitting constraints induced from the MDP transition function. Based on the average flow variables, Varakantham et al. [133] proposed a individual value function estimation, which facilitated fictitious play computation of the policy. In addition, as the average flow MDP has Varakantham et al. [131] showed that the average flow MDP problem can be modeled and solved by a mathematics programming. Average flow and mean-field methods are empirically shown to achieve good results in some domains. However, it is worth noting that these methods provide only an estimation of the original problem, and this estimation can incur a high approximation error when the transition function is highly non-linear.

2.4.3 Lifted inference

The sufficient statistics of the counts are exploited in probabilistic inference. Poole [92] and Braz et al. [17] showed that the probabilistic inference problem in the Markov Logic Network graphical model can be solved by reasoning on the count variables instead of joint individual variables. This can be done by defining the count distribution for each factor of the corresponding graphical model and applying a variable elimination to the count variables under the count distributions. As the state space of count variables is much smaller than the original joint variables', performing exact inference with count variables is more tractable. The idea to "lift" problems from joint variable representation to count representation is also considered to improve dynamic programming planning as shown in [97]. Specifically, Robbel et al. [97] showed that if the Dynamic Bayesian Network is factorizable and the state transition is function of the count. The value function can be re-written as function of the count and the Bellman equation can be also re-written accordingly. The count transition function is considered in many network diffusion domains [57] such as disease outbreak where the probability for a node to be infected is dependent on the number of its infected neighbors [97]. Although exact inference and planning operators have lower complexity under the count space, these operators are no longer tractable in our $\mathbb{C}$Dec-POMDP. This is due to the dense connection between local states and large population size in $\mathbb{C}$Dec-POMDP. Therefore, we approach the collective planning problems in $\mathbb{C}$Dec-POMDP by reinforcement learning and use a sampling based method rather than the exact lifted dynamic programming method in the current research literature.

A closely related work to our $\mathbb{C}$Dec-POMDP model is the collective graphical model (CGM) introduced by Sheldon and Dietterich [108]. CGM is used to model the count distribution when agent behavior is modelled with an undirected tree-structured graphical model. In the special case of $\mathbb{C}$Dec-POMDP where agents move with open-loop policy and the state transition of each individual is inde-

pendent of other agents, the dynamics of $\mathbb{C}$Dec-POMDP can be modelled by a chain-structured CGM. In general, a $\mathbb{C}$Dec-POMDP with closed-loop policy and the transition-dependency induced by interaction amongst agents can not be modelled by a CGM. Although the dynamics of $\mathbb{C}$Dec-POMDP is more complex than CGM, the sampling process in $\mathbb{C}$Dec-POMDP is simpler than CGM because we can sample the count by a forward sampling process based on $\mathbb{C}$Dec-POMDP's Bayesian network. More details of the relation between $\mathbb{C}$Dec-POMDP and CGM are discussed in the next chapter.

2.5 Summary

We introduced $\mathbb{C}$Dec-POMDP as a framework to model the large population systems in which the transition and reward functions are represented by the counts. The $\mathbb{C}$Dec-POMDP model has a wide range of applications in congestion domains and logistics supply-demand matching domains. We showed one of the most important properties of the $\mathbb{C}$Dec-POMDP model is that count variables are sufficient statistics for planning. This allows us to lift the original planning model into a new planning representation with the counts. In particular, we showed that both the global value function and individual value function are functions of the counts. Moreover, by showing the Dynamic Bayesian Network over the count, we proposed an efficient method to directly sample the count from its collective distribution.

Chapter 3

Collective Graphical Model

In previous chapters, we have shown collective distribution over the counts of agents in different states as a basis for multi-agent planning in $\mathbb{C}$Dec-POMDPs. In this chapter, we introduce the collective graphical model proposed in [108] and discuss the relationship between planning problem in a $\mathbb{C}$Dec-POMDP and inference problem in a collective graphical model.

Collective graphical model (CGM) proposed by Sheldon and Dietterich [108] model the count statistics of a homogeneous population whose individual behavior is modeled by a standard graphical model [108]. CGM is developed on the assumption that agents behave independently from each other. This is more restrictive than our $\mathbb{C}$Dec-POMDP model where agents influence each other by their collective behavior. However, CGM is more general than $\mathbb{C}$Dec-POMDP because it can have tree-structureed representation. The $\mathbb{C}$Dec-POMDP can only model chain-structured and directed graphical model. To unify our $\mathbb{C}$Dec-POMDP model with its predecessor CGM, we define a generalized collective graphical model with count-based potential function.

CGM is mainly used to study collective inference problems [108, 63, 78]. In particular, given noisy observations of the node counts or edge counts for a CGM, it is required to infer the underlying count table or estimate parameters of the CGM

model. Count inference and parameter estimation are co-related problems. Sheldon et al. [107] showed that the parameter in a CGM can be estimated by a modified Expectation Maximization algorithm which repeatedly finds the maximum likelihood counts and uses such counts to update model parameters. As shown in the literature, probabilistic inference and planning problem are co-related problems [123, 101, 138], we discuss the relationship between the parameter estimation in CGM with $\mathbb{C}$Dec-POMDP planning problems.

3.1 Collective Graphical Models

3.1.1 Motivation

Collective graphical models (CGMs) are a recently introduced formalism for inference and learning about a population of independent and identically distributed individuals when only *noisy* and *aggregate* observations are given [108]. In many settings, such as in ecology, social sciences and transportation, data about each individual is rarely available due to privacy concerns or the difficulty of tracking each individual over time. As an example, the eBird database[1] contains observations about the count of birds at different locations and time across the North American region [106]. The data released by Census Bureau may contain count-based aggregate information for privacy reasons. Similarly, the traffic data typically contains noisy aggregate count of vehicles at different locations [71, 57]. In such scenarios, CGMs can be used to model the individual-level behavior by doing inference and learning based on the available *noisy* and *aggregate* data.

[1] http://ebird.org/

3.1.2 Background

Collective graphical models (CGMs) describe the distribution of the aggregate statistics of a population of individuals sampled from a discrete graphical model (also known as the *individual* model). Let $G = (V, E)$ denote a pairwise Markov random field describing the individual model. Let $\boldsymbol{X} = (X_1, \ldots, X_{|V|})$ denote the random variables associated with each node in G. The joint-probability is:

$$p(\boldsymbol{x}; \boldsymbol{\theta}) = \Pr(\boldsymbol{X} = \boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{(i,j) \in E} \phi_{ij}(x_i, x_j; \boldsymbol{\theta}) \tag{3.1}$$

where $\phi_{ij}(\cdot, \cdot; \boldsymbol{\theta})$ is the potential function for the edge (i, j) in G defined as per the parameters $\boldsymbol{\theta}$; $Z(\boldsymbol{\theta})$ denotes the partition function. Let the domain of each variable be denoted using $\mathcal{X}$.

Consider i.i.d. samples $\{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$ drawn from the model G representing a population of M individuals. We can define the counts or contingency tables for this population as follows. Let $\mathbf{n}_i = (n_i(x_i) : x_i \in \mathcal{X})$ represent the node counts, and $\mathbf{n}_{ij} = \big(n_{ij}(x_i, x_j) : x_i, x_j \in \mathcal{X}\big)$ represent the edge counts for different edges. The counts $n_i(x_i)$ and $n_{ij}(x_i, x_j)$ are defined as:

$$n_i(x_i) = \sum_{m=1}^{M} \mathbb{I}\big(X_i^m = x_i\big) \tag{3.2}$$

$$n_{ij}(x_i, x_j) = \sum_{m=1}^{M} \mathbb{I}\big(X_i^m = x_i, X_j^m = x_j\big) \tag{3.3}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function.

3.1.3 CGM Distribution

We first describe the structure of the CGM distribution $p(\mathbf{n}; \boldsymbol{\theta})$. The CGM distribution is defined over the junction tree $\mathcal{T}$ corresponding to the graph G. Each node t of this tree is associated with a clique $C_t \subseteq V$. Let $\mathcal{C}$ denote the set of all the cliques

for the tree $\mathcal{T}$. If C and C' denote two adjacent cliques in $\mathcal{T}$, then $S = C \cap C'$ denotes a *separator*. Let $\mathcal{S}$ denote the set of all the separators for this junction tree. For any subset $C \subseteq V$, and a particular assignment $x_C \in \mathcal{X}^{|C|}$, we can define the counts $n_C(x_C)$ analogous to the node and edge counts as:

$$n_C(x_C) = \sum_{m=1}^{M} \mathbb{I}\left(X_C^m = x_C\right) \tag{3.4}$$

Using counts $n_C(x_C)$, we can define the contingency table $\mathbf{n}_C$ similar to tables $\mathbf{n}_i$, $\mathbf{n}_{ij}$ for nodes and edges of G. Let $\mathbf{n} = \{\mathbf{n}_A : A \in \mathcal{C} \cup \mathcal{S}\}$ denote the combined vector of clique and separator counts. The vector $\mathbf{n}$ is sufficient statistic of the population [63]. The distribution over this vector is denoted as the *CGM distribution*.

As shown in [63], the CGM distribution is given as $p(\mathbf{n};\boldsymbol{\theta}) = f(\mathbf{n};\boldsymbol{\theta})g(\mathbf{n})$ where we have:

$$f(\mathbf{n};\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})^M} \prod_{(i,j)\in E} \prod_{x_i,x_j} \phi_{ij}(x_i, x_j;\boldsymbol{\theta})^{n_{ij}(x_i,x_j)} \tag{3.5}$$

$$g(\mathbf{n}) = M! \frac{\prod_{S\in\mathcal{S}} \prod_{x_S\in\mathcal{X}^{|S|}} \left(\mathbf{n}_S(x_S)!\right)^{\nu(S)}}{\prod_{C\in\mathcal{C}} \prod_{x_C\in\mathcal{X}^{|C|}} \mathbf{n}_C(x_C)!} \tag{3.6}$$

where $\nu(S)$ denotes the number of times S appears as a separator or the number of junction tree edges (t, t') for which $S = C_t \cap C_{t'}$. The distribution $p(\mathbf{n};\boldsymbol{\theta})$ is defined over the following set of constraints:

$$\sum_{x_C\in\mathcal{X}^{|C|}} \mathbf{n}_C(x_C) = M \;\forall C \in \mathcal{C} \tag{3.7}$$

$$\mathbf{n}_S(x_S) = \sum_{x_{C\setminus S}} \mathbf{n}_C(x_S, x_{C\setminus S}) \;\forall x_S; \forall S \sim C \in \mathcal{T} \tag{3.8}$$

where $S \sim C \in \mathcal{T}$ implies that S is adjacent to C in the junction tree $\mathcal{T}$. We also have the constraint that $\mathbf{n}$ must be integer valued. Notice that the above two sets of constraints are similar to the constraints defining the marginal polytope for

a graphical model [135]. The only difference being that the counts must sum to M instead of 1, and counts must be integers.

A diagram of CGM of a tree model with 4 nodes is shown in Figure 3.1 using plate notation. The undirected sub-graph inside in the plate m represents the relation between variables $X_1^m, X_2^m, X_3^m, X_4^m$ of individual m. The separator counts and clique counts are computed by aggregating individual values.

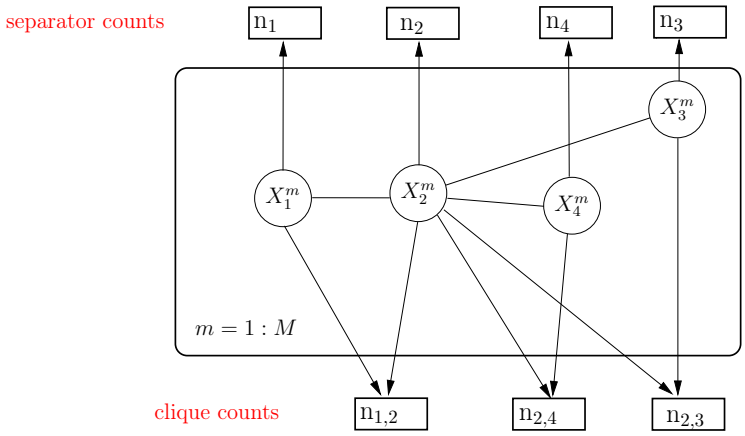


Figure 3.1: Example of collective graphical model in a tree model with 4 nodes

3.1.4 Relation between CGM and ℂDec-POMDP

One of common application of CGM is to model animal migration or human movement [108, 63, 78]. In such a domain, a geographical map is divided into sub-areas and scientists would try to measure the counts of individuals in each sub-areas over time. As a motivation, we consider the concrete domain of bird migration [108, 63, 78] where a map is divided into a set Z of grid cells. We can model the location of a bird m at time period t by node variable X_t^m taking value x_t in Z. The relation between locations of the bird from time period t to $t+1$ is usually assumed to follow the Markov transition as $\phi_{t,t+1}(x_t, x_{t+1}) = P(x_{t+1}|x_t)$. We can model a special instance of ℂDec-POMDP using a CGM as follows:

Proposition 3.1. *In a ℂDec-POMDP with the individual transition function dependent on only the previous local state and action as $P_l(s_{t+1}^m|s_t^m, a_t^m)$ and agents*

follow open-loop policy $\pi(a_t^m|s_t^m)$, the $\mathbb{C}$Dec-POMDP can be modeled by a CGM.

Proof. We construct the CGM for this special case of $\mathbb{C}$Dec-POMDP by defining CGM nodes to be s_t^m for agent m's local state at time t and a_t^m for agent's local action at time $t, \forall t$. The separators in this CGM are $\langle s_t^m \rangle_t$ and the cliques are $\langle s_t^m, a_t^m, s_{t+1}^m \rangle_t$. The clique potential is defined by

$$\phi_t(s_t^m, a_t^m, s_{t+1}^m) = \pi(a_t^m|s_t^m) P_l(s_{t+1}^m|s_t^m, a_t^m).$$

The plate notation diagram of this CGM is shown in Figure 3.2. □

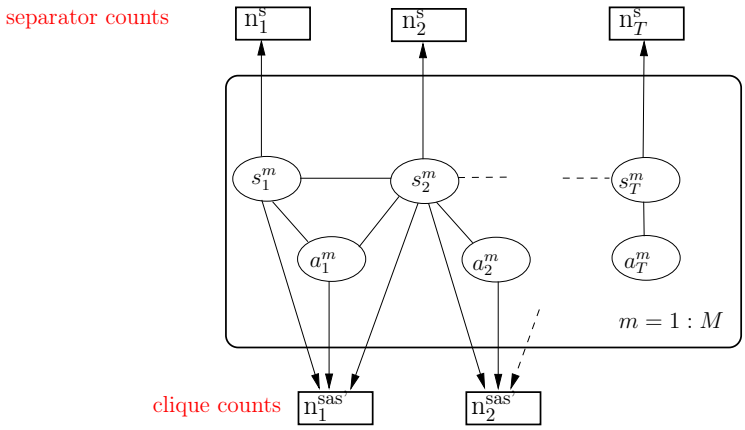


Figure 3.2: Collective graphical model of a independent-transition and open-loop policy $\mathbb{C}$Dec-POMDP.

Although a special instance of a $\mathbb{C}$Dec-POMDP can be modeled with CGM, its CGM representation has not much benefit besides showing the relation between the CGM model and the $\mathbb{C}$Dec-POMDP model. Advantages of $\mathbb{C}$Dec-POMDP over CGM include: 1) $\mathbb{C}$Dec-POMDP can model the count-based transition and closed-loop policy while CGM assumes independence between agents; 2) Multinomial sampling process in $\mathbb{C}$Dec-POMDP with the directed graph is much easier than the rejection sampling algorithm with Dobra Markov basis proposed by Sheldon and Dietterich [108].

3.2 Collective inference in CGM

3.2.1 Noisy observation models

In application domains of CGM, the sufficient-statistics table $\mathbf{n} = \{\mathbf{n}_A : A \in \mathcal{C} \cup \mathcal{S}\}$ are not fully observable but there are only some noisy observations about some subset of the sufficient statistic $\mathbf{n}$ are provided. The most common noisy observations in CGM are the node count observation table $\mathbf{y}_i$ of the node counts $\mathbf{n}_i$ and the edge count observation table $\mathbf{y}_{ij}$ or the edge counts $\mathbf{n}_{ij}$ [78].

The probability $p\big(y(\cdot)|n(\cdot)\big)$ is referred to as the *noise model* for the CGMs. The typical noise model used for CGMs include the Poisson and the Gaussian noise. It is usually assumed that $p(\boldsymbol{y}|\mathbf{n})$ is log-concave in $\mathbf{n}$, which makes the negative log-likelihood convex in $\mathbf{n}$ [78].

Given the count observations $\mathbf{y}$, the main 2 inference problems in CGM are 1) *aggregate MAP inference* to find the maximum likelihood $\mathbf{n}$ corresponding to $\mathbf{y}$ and 2) estimation of parameters $\boldsymbol{\theta}$ of the potential function ϕ.

3.2.2 Aggregate MAP inference

In the aggregate MAP inference, we are given noisy observations $\boldsymbol{y}$ about some subset of sufficient statistic $\mathbf{n}$. Our goal in this work is to find the best count vector $\mathbf{n}$ that maximizes $p(\mathbf{n}|\boldsymbol{y};\boldsymbol{\theta}) \propto p(\mathbf{n};\boldsymbol{\theta})p(\boldsymbol{y}|\mathbf{n})$. That is:

$$\mathbf{n}^\star = \arg\max_{\mathbf{n}} \big[\log p(\mathbf{n};\boldsymbol{\theta}) + \log p(\boldsymbol{y}|\mathbf{n})\big] \tag{3.9}$$

To solve the above optimization problem, we first analyze the structure of $\log p(\mathbf{n};\boldsymbol{\theta}) = \log f(\mathbf{n};\boldsymbol{\theta}) + \log g(\mathbf{n})$, where $p(\cdot)$ is the CGM distribution. Using definitions (3.5), (3.6), we have:

$$\log f(\mathbf{n};\boldsymbol{\theta}) \propto \sum_{(i,j)} \sum_{x_i,x_j} \mathbf{n}_{ij}(x_i,x_j) \log \phi_{ij}(x_i,x_j) \tag{3.10}$$

where we have ignored terms that are independent of $\mathbf{n}$ such as $M \log Z(\boldsymbol{\theta})$. We further have:

$$\log g(\mathbf{n}) \propto \sum_{S\in\mathcal{S}} \sum_{x_S} \nu(S) \log \left(n_S(x_S)!\right) - \sum_{C\in\mathcal{C}} \sum_{x_C} \log \left(n_C(x_C)!\right)$$

As addressing integer counts $\mathbf{n}$ is challenging within an optimization framework directly, we make an approximation by making counts $\mathbf{n}$ continuous, as also used previously [107]. For continuous $\mathbf{n}$, we can further use the Stirling's approximation as $\log n! \approx n \ln n - n$. Using these approximations, we can simplify $\log g(\mathbf{n})$ further as:

$$\begin{aligned}
&\log g(\mathbf{n}) \propto\!\!\!\!\approx \sum_{S\in\mathcal{S}} \sum_{x_S} \nu(S) \left[n_S(x_S) \log n_S(x_S) - n_S(x_S)\right] - \\
&\sum_{C\in\mathcal{C}} \sum_{x_C} \left[n_C(x_C) \log n_C(x_C) - n_C(x_C)\right] \\
&= \sum_{S\in\mathcal{S}} \sum_{x_S} \nu(S) n_S(x_S) \log n_S(x_S) - \sum_{S\in\mathcal{S}} \sum_{x_S} \nu(S) n_S(x_S) \\
&- \sum_{C\in\mathcal{C}} \sum_{x_C} n_C(x_C) \log n_C(x_C) + \sum_{C\in\mathcal{C}} \sum_{x_C} n_C(x_C)
\end{aligned}$$

We can simplify the above expression by observing that as per constraints (3.7) we have $\sum_{x_C} n_C(x_C) = M$, and from constraints (2.8), we have $\sum_S n_S(x_S) = \sum_{x_C} n_C(x_C) = M$. Thus, we have the final simplified expression for $\log g(\mathbf{n})$ after ignoring terms independent of $\mathbf{n}$ as below:

$$\log g(\mathbf{n}) \propto \sum_{S \in \mathcal{S}} \sum_{x_S} \nu(S) n_S(x_S) \log n_S(x_S) - \sum_{C \in \mathcal{C}} \sum_{x_C} n_C(x_C) \log n_C(x_C)$$

Notice that the above expression subject to the constraints (3.7) and (3.8) is analogous to the entropy of a graphical model, the only difference being that counts sum up to M, rather than 1. We can now use the Bethe entropy [154] to approximate this term, which is nicely decomposable along the nodes and edges of the individual model G. We have the following approximation:

$$\log g(\mathbf{n}) \propto \sum_{i \in V} \sum_{x_i \in \mathcal{X}} \big(\nu(i) - 1\big) n_i(x_i) \log n_i(x_i) - \sum_{(i,j) \in E} \sum_{x_i, x_j} n_{ij}(x_i, x_j) \log n_{ij}(x_i, x_j) \tag{3.11}$$

where $\nu(i)$ denotes the degree of the node i in the graph G. The above approximation represents a significantly more tractable form of $\log g(\mathbf{n})$ as all the terms are defined over the pairwise graph G, rather than the junction tree. Finally combining (3.10) and (3.11) and considering a factorized likelihood function $p(\mathbf{y}|\mathbf{n}) = \prod_i \prod_{x_i} p(y_i(x_i)|n_i(x_i)) \prod_{ij} p(y_{ij}(x_i, x_j)|n_{ij}(x_i, x_j))$, we have the approximate objective function of aggregate MAP inference as

$$APPROX(\boldsymbol{\theta}, \mathbf{y}) : \max_{\mathbf{n}} \sum_{i \in V} \Big[(1 - \nu(i)) \sum_{x_i} n_i(x_i) \log n_i(x_i) + p(y_i(x_i)|n_i(x_i))\Big]$$
$$\sum_{(i,j) \in E} \sum_{x_i, x_j} \Big[n_{ij}(x_i, x_j)[\phi_{ij}(x_i, x_j) - \log n_{ij}(x_i, x_j)] + p(y_{ij}(x_i, x_j)|n_{ij}(x_i, x_j))\Big] \tag{3.12}$$

The constraint set that each valid $\mathbf{n}$ must satisfy is given as:

$$n_i(x_i) = \sum_{x_j} n_{ij}(x_i, x_j) \; \forall j \in \text{Nb}_i, \forall x_i, \forall i \in V \tag{3.13}$$

$$\sum_{x_i} n_i(x_i) = M, \forall i \in V \tag{3.14}$$

The above set of constraints for CGMs are similar to the constraints that define the local polytope for graphical models [135].

3.2.3 Parameter estimation

Given the observations $\mathbf{y}$ for a CGM, we can estimate the parameter $\boldsymbol{\theta}$ by maximizing the likelihood function

$$\max_{\boldsymbol{\theta}} p(\mathbf{y}|\boldsymbol{\theta}) = \sum_{\mathbf{n} \in \Omega} p(\mathbf{y}|\mathbf{n}, \boldsymbol{\theta}) = \sum_{\mathbf{n} \in \Omega} p(\mathbf{y}|\mathbf{n}) p(\mathbf{n}|\boldsymbol{\theta}), \tag{3.15}$$

in which the summation at the RHS is over the space of all feasible values of the counts $\mathbf{n}$ satisfying count consistency constraints (3.7),(3.8).

The count variables have exponential-size space [107], which makes it infeasible to compute the exact expectation in equation (3.15). Instead, an useful alternative method is to use Expectation Maximization (EM) algorithm [24] to repeatedly maximize a surrogate objective

$$\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) = \max_{\boldsymbol{\theta}} p(\mathbf{y}|\boldsymbol{\theta}^\star) \log p(\mathbf{y}|\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \sum_{\mathbf{n} \in \Omega} p(\mathbf{y}|\mathbf{n}, \boldsymbol{\theta}^\star) \log p(\mathbf{y}|\mathbf{n}, \boldsymbol{\theta}) \tag{3.16}$$

in which $\boldsymbol{\theta}$ is the parameters to optimized, and $\boldsymbol{\theta}^\star$ is the parameters optimized in the previous iteration and fixed in this current iteration. EM maximizes a lowerbound of the likelihood function hence converges to a local optimum [75]. As noted in [107], because the joint distribution $p(\cdot)$ of CGM is from an exponential family, this EM objective function is simplified into maximization of $\log p(\mathbf{y}|\bar{\mathbf{n}}, \boldsymbol{\theta})$ with

$\bar{\mathbf{n}} = \mathbb{E}_{\boldsymbol{\theta}^\star}[\mathbf{n}|\mathbf{y}]$. In general, this conditional expectation $\mathbb{E}_{\boldsymbol{\theta}^\star}[\mathbf{n}|\mathbf{y}]$ is difficult to estimate, Sheldon et al. [107] proposed to estimate it by solution $\mathbf{n}^\star_{APPROX}$ of the aggregate MAP inference problem from solving the objective function (3.12).

3.2.4 Relation between CGM inference and ℂDec-POMDP planning

Researchers have shown the relation between reinforcement learning problems and probabilistic inference problems [138, 101]. In fact, MDP planning problems can be cast as probabilistic inference problems and solved using probabilistic inference algorithms such expectation maximization [123, 58]. We have shown in proposition 3.1 that CGM and ℂDec-POMDP in fact overlap in a special case of independent transition. In this special case, ℂDec-POMDP problem can be modeled as a collective inference problem in CGMs, consequently it can be solved by CGM inference solvers.

Proposition 3.2. *When the individual transition is only dependent on the previous local state and action as $P_l(s^m_{t+1}|s^m_t, a^m_t)$ and agents follow the same open-loop policy $\pi(a^m_t|s^m_t)$, the ℂDec-POMDP planning problem can be re-cast as an CGM parameter inference.*

Proof. Using proposition 3.1, we can represent the dynamic in ℂDec-POMDP by a CGM nodes with a set of nodes $\langle s^m_t, a^m_t\rangle_{m,t}$ for all agents over planning horizon. Similar to Toussaint et al. [123], Kumar et al. [58], we can define a likelihood function by an auxiliary binary variables $\hat{\mathbf{y}} \in \{0; 1\}$ to represent the reward received at each time T by

$$p(\hat{\mathbf{y}}_T = 1|\mathbf{n}_T, T) = \frac{r(\mathrm{n}^{sa}_T) - r_{min}}{r_{max} - r_{min}},$$

in which r_{max}, r_{min} are the maximum and minimum values of the immediate rewards respectively.

Then we can re-interpret the planning objective function $\mathbb{E}[\sum_{T=0}^{H} r_T]$ as a maximization of a mixture of likelihood over time as

$$\max p(\hat{\mathbf{y}}|\boldsymbol{\theta}) = \sum_{T=0}^{H} p(T) \sum_{\mathbf{n}_{1:T} \in \Omega} p(\hat{\mathbf{y}}_T = 1|\mathbf{n}_T, T) p(\mathbf{n}_{1:T}|\boldsymbol{\theta}),$$

in which mixture weight $p(T) = 1/H$. □

3.3 Related works

The count inference was first proposed as the MAP inference for CGMs by Sheldon et al. [107] as a sub-step for parameters learning [107] within the EM framework [24]. Since then, there are a number of approaches proposed for inference in CGMs [108, 107, 63, 114]. Sheldon et al. develop a continuous convex relaxation of the MAP inference problem formulated over the junction tree derived from the individual model, and solve it using a generic optimization solver. Liu et al. develop a Gaussian approximation for CGMs and use Expectation-Propagation for inference. Sun et al. generalize the well known belief propagation algorithm [89] to *nonlinear* belief propagation (NLBP) for CGMs.

There is a close relation between MAP inference in CGMs and probabilistic inference in standard graphical model [49, 135]. In particular, the marginal count constraint and likelihood function of the count variables are equivalent to marginal probability constraint and posteriori probability function in probability inference [49]. This relation between count inference and probability inference was also noticed previously when Liu et al. [63] and Sun et al. [114] adopted belief propagation methods into count inference problem. This motivates us to consider other techniques from standard probabilistic inference, namely Bethe entropy approximation [154] and the concave-convex procedure (CCCP) [156, 155], to develop approximate solution for high tree-width MAP inference in CGMs.

3.4 Summary

In this chapter, we showed the relation between CGM inference problems and $\mathbb{C}$Dec-POMDP planning problems.

The collective inference problem was introduced by Sheldon and Dietterich [108] to infer the underlying counts from the noisy observation of the counts. To solve the collective inference problem, Sheldon and Dietterich [108] constructed a collective graphical model (CGM) of the counts as a *lifted* representation of the population. The CGM model considers agents having transition function independent from each other. In our $\mathbb{C}$Dec-POMDP model, agents are interacting with each other and their transition functions are interdependent through the collective behavior (the counts). However, we showed that our planning model and CGM overlap in the case of independent transition and open-loop policy. Furthermore, in this special case, the objective function in collective planning and can be re-cast as a likelihood function in CGM.

Although we can model a special instance of $\mathbb{C}$Dec-POMDP planning as CGM inference problem, this is shown only for demonstrating the relation between $\mathbb{C}$Dec-POMDP and CGM. In general, sampling process for the directed graphical model in $\mathbb{C}$Dec-POMDP is more efficient than rejected sampling procedure in CGM.

Chapter 4

Collective Multi-agent Reinforcement Learning Framework

In this chapter, we present general frameworks to optimize agent policies individual policy in $\mathbb{C}$Dec-POMDP model. First, we study the model-based approach by showing the dynamic program for $\mathbb{C}$Dec-POMDPs as a special case of DEC-POMDPs. We show that we can reformulate the dynamic programming in $\mathbb{C}$Dec-POMDP by using the counts which have lower complexity than the dynamic programming over joint state-action. Unfortunately, the *lifted* dynamic programming algorithm in $\mathbb{C}$Dec-POMDP still has exponential time complexity with respect to the number of states and actions. This motivates us to develop sampling-based planning algorithms using reinforcement learning in $\mathbb{C}$Dec-POMDPs. To establish the basis for efficient RL algorithms, we study decomposition of the critic and the decomposition of policy gradient in $\mathbb{C}$Dec-POMDPs. For the critic decomposition, we show that the compatible value function approximation (or critic) in $\mathbb{C}$Dec-POMDP is decomposable amongst agents. For the actor decomposition, we show that if the critic is a linear function of the action counts, the policy gradient is decomposable. We show that the decomposition of actor and critic functions in $\mathbb{C}$Dec-POMDP also addresses the credit-assignment in MRL problems, therefore it can be used to design effective

$\mathbb{C}$Dec-POMDP algorithms with fast convergence to high quality solutions.

4.1 Multi-agent Planning Model

4.1.1 Multi-agent Dec-POMDP

Similar to [12, 58], we define a standard multi-agent decentralized partially observation Markov Decision Process (Dec-POMDP) by

- A finite planning horizon H.
- $\boldsymbol{S}$ denotes a finite set of states with initial distribution b_o.
- The number of agents M.
- A set of action A^m for each agent m. We denote an individual action as $j^m \in A^m$. The joint action space is $\boldsymbol{A} = \otimes_{m=1}^{M} A^m$.
- A finite set of observation Y^m for each agent m.
- The observation model for each agent m is defined by the probability $o^m(y_{t+1}^m|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ agent m observes y_{t+1}^m given the global states $\boldsymbol{s}_t, \boldsymbol{s}_{t+1}$ and joint action $\boldsymbol{a}_t$.
- The transition function $\phi_t\big(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t\big)$ to be the probability the system transits to new state $\boldsymbol{s}_{t+1}$ from state $\boldsymbol{s}_{t+1}$ and joint action $\boldsymbol{a}_t$, $\forall \boldsymbol{s}_t, \boldsymbol{s}_{t+1} \in \boldsymbol{S}, \boldsymbol{a}_t \in \boldsymbol{A}$.
- The whole system receives the reward $r = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ dependent on global state $\boldsymbol{s}_t$ and joint action $\boldsymbol{a}_t$.

Reactive policy

The reactive policy of an agent m is defined by a function $\pi_t^m(j^m|y^m) \in [0,1]$ to be the probability of agent m to take action j^m given its observation y^m at time t. The reactive policy is only dependent on the last observation.

Finite state controller (FSC) policy

As shown for Dec-POMDPs in [12], the optimal action of an agent m at time t depends on its observation and action history $(a_1^m, o_1^m, \ldots, a_{t-1}^m, o_{t-1}^m)$. However, this would require agent to exhaustively maintain a large memory space to store optimal actions for all possible history. To overcome this, we can summarize history with a finite internal memory state $q^m \in Q^m$. To use the finite state controller, policy function is extended to consist of 2 components: an action distribution $\pi^m(j^m|q^m)$ to generate action based on the local memory state, and a memory state transition function $\lambda^m(q_{t+1}^m|y_{t+1}^m, q_t^m)$ to specify the next memory state given the last memory state and local observation. Reactive policy could be considered as a special FSC policy with memory state to be identical with the local observation.

We denote the joint memory state to be $q \in Q = \otimes Q^m$, the joint action to be $\langle j_t^m \rangle_m$, the joint memory state to be $\langle q^m \rangle_m$. The value function of FSC policy can be defined by the dynamic programming equation as in [58]

$$
\begin{aligned}
&V_t(q_t, \boldsymbol{s}_t) \\
&= \sum_{\langle j_t^m \rangle_m \in A} \Big(\prod_m \pi^m(j_t^m|q_t^m) \Big) \Big[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \sum_{\boldsymbol{s}_{t+1}} P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \langle j_t^m \rangle_m) \sum_{y_{t+1}} \\
&\qquad \sum_{<q_{t+1}^m>_m} \Big(\prod_m o^m(y_{t+1}^m|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \lambda^m(q_{t+1}^m|y_{t+1}^m, q_t^m) \Big) V(\langle q_{t+1}^m \rangle_m, \boldsymbol{s}_{t+1}) \Big]
\end{aligned}
\tag{4.1}
$$

4.1.2 ℂDec-POMDP as Lifted DEC-POMDP

Based on the definition of the ℂDec-POMDP model in previous Section 2.2, we can consider ℂDec-POMDP as a special case of Dec-POMDP with following properties:

- The global state $\boldsymbol{s}$ is the joint state $\langle s^m \rangle_m$.
- All agent have the same local state space S and local action space A.
- A single policy function π is shared amongst homogeneous agents.
- The local observation is a local view of the global state count $\mathrm{n}_t^{\mathrm{s}}$ corresponding to the joint state $\langle s^m \rangle_m$.

In this thesis, we consider the reactive policy function $\pi(j^m|i^m, o^m(i^m, \mathrm{n}_t^{\mathrm{s}}))$ associating with the local state and local observation of agent. This allows us to efficiently train policy function using collective variables. Under this policy, the value function in a ℂDec-POMDP can be defined by

$$\begin{aligned} & V_t(\boldsymbol{s}_t) \\ &= \sum_{\langle j_t^m \rangle_m \in A} \Big(\prod_m \pi(j_t^m | i^m, o^m(i^m, \mathrm{n}_t^{\mathrm{s}})) \Big) \Big[r(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ &\qquad + \sum_{s_{t+1}} P(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \langle j_t^m \rangle_m) V(\boldsymbol{s}_{t+1}) \Big] \end{aligned} \tag{4.2}$$

Using similar manipulation as in theorem 2.3, we can aggregate agents in the same local state-action to have the value function defined in term of the state-action count

Proposition 4.1. *For ℂDec-POMDPs, the value function can be represented with respect to the counts as follows:*

$$\begin{aligned} & V_t^{\pi}(\mathrm{n}_t^{\mathrm{s}}) \\ &= \sum_{\mathrm{n}_t^{\mathrm{sa}}} P^{\pi}(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, \pi) \Big[r(\mathrm{n}_t^{\mathrm{sa}}) + \sum_{\mathrm{n}_t^{\mathrm{sas}'}} P^{\pi}(\mathrm{n}_t^{\mathrm{sas}'} \mid \mathrm{n}_t^{\mathrm{sa}}) V(\mathrm{n}_{t+1}^{\mathrm{s}}) \Big], \forall \mathbf{n}_t \in \Omega_t \end{aligned} \tag{4.3}$$

in which Ω_t *is the set of feasible count values and* $P(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, \pi), P(\mathrm{n}_t^{\mathrm{sas}'} \mid \mathrm{n}_t^{\mathrm{sa}})$ *are the collective distributions defined in corollary 2.1 as*

$$P(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, \pi) = \prod_{i \in S} \Big(\frac{n_{s_t}(i)!}{\prod_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)!} \prod_{j \in A} \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \Big)$$

$$P(\mathrm{n}_t^{\mathrm{sas}'} \mid \mathrm{n}_t^{\mathrm{sa}}) = \prod_{i \in S, j \in A} \Big(\frac{\mathrm{n}_t^{\mathrm{sa}}(i,j)!}{\prod_{i' \in S} \mathrm{n}_t^{\mathrm{sas}'}(i,j,i')!} \prod_{i' \in S} \phi_t(i'|i,j,\mathrm{n}_t^{\mathrm{s}}(i))^{\mathrm{n}_t^{\mathrm{sas}'}(i,j,i')} \Big)$$

Proof. The derivation to $V_t^{\pi}(\mathrm{n}_t^{\mathrm{s}})$ is similar to the proof in theorem 2.4. □

When the number of agents is greater than 1, there could be more than one (permutable) joint state-action $\boldsymbol{s}_t \boldsymbol{a}_t$ corresponding to each value of the state count $\mathrm{n}_t^{\mathrm{sa}}$. As a consequence, the space of feasible counts is than the space of all possible joint state-actions. Therefore, the value function defined by the counts as in equation 4.3 is more compact than the joint value function defined by equation 4.2. Assume the number of agents is larger than number of the local states, we can quantify the complexity of planning space by:

Proposition 4.2. *The size of joint state space is* $|S|^M$ *while the size of state count space is* $\binom{M}{|S|-1}$.

Proof. As each agent can be in one i in S, the total number of combination of states of M agents is $|S|^M$. To form a feasible state count of M agents over $|S|$ states, we can firstly choose $|S|-1$ numbers in range $[0; M]$ and rank them in the increasing order $\{x_1, \ldots, x_{|S|-1}\}$. Then together with $x_0 = 0, x_{|S|} = M$, we can specify state count $n(i)$ at each local state i to be the number of unique integer in range $[x_i, x_{i+1})$. The number of ways choose such $|S|-1$ number is $\binom{M}{|S|-1}$, which is equivalent to the size of state count space. □

The asymptotic complexity of the state count space is $\Theta\binom{M}{|S|-1} = M^{|S|-1}$. Therefore, when the size $|S|$ of state space is much smaller than number of agents, the state count space has much lower complexity than the joint state space.

Although the count-based value function is more tractable than the joint state-action value function, its $O\binom{M}{|S|-1}$ complexity is exponentially large. This motivates us to a develop sampling-based approaches for $\mathbb{C}$Dec-POMDP planning problems.

Notice: We can extend our model to FSC based policy by adding additional counts for the number of agents moving from one memory state to another. However, this requires a comprehensive study of the trade-off between the increment of the count dimensions and the richness of FSC policy. This is beyond the scope of this thesis. We leave it as a future research direction.

4.2 Reinforcement Learning

In this section, we introduce the general reinforcement learning (RL) framework for the Markov Decision Process with full observation. This is the basis for us to derive the RL framework for Dec-POMDP and $\mathbb{C}$Dec-POMDP problems later.

We consider the general $\mathbb{C}$Dec-POMDP model with global state component d_t as introduced in Chapter 2. The value function and policy function are defined as $Q(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ and $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t)$ respectively. $\boldsymbol{s}_t, \boldsymbol{a}_t$ denote the joint state and joint action in Dec-POMDP or single state and single action in standard MDP. If there is no global component, we can set $d_t = null$.

To extend MDP results to POMDP, the partial observation o about the environment state can be included later by directly replacing $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t)$ with $\pi(\boldsymbol{a}_t|o)$.

Consider state-action value function in a MDP under a policy function π and transition function P

$$Q_t^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \mathbb{E}[\sum_{t' \geq t} r_{t'}|\boldsymbol{s}_t, \boldsymbol{a}_t, d_t, \pi] \tag{4.4}$$

Q value can be computed exactly by a dynamic program as: [117]

$$Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) + \sum_{\boldsymbol{s}_{t+1}, d_{t+1}} P(\boldsymbol{s}_{t+1}, d_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t, d_t) V_t^\pi(\boldsymbol{s}_{t+1}, d_{t+1}), \forall \boldsymbol{s}_t \in \boldsymbol{S}, \boldsymbol{a}_t \in \boldsymbol{A} \quad (4.5)$$

$$V_t^\pi(\boldsymbol{s}_t, d_t) = \sum_{\boldsymbol{a}_t} \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t), \forall \boldsymbol{s}_t \in \boldsymbol{S} \quad (4.6)$$

Theoretically, after computing the value for all states and actions based on this dynamic program equations (4.5), (4.6), we can optimize policy for time t by:

$$\max_\pi \sum_{\boldsymbol{s}_t, d_t} P(\boldsymbol{s}_t, d_t) V_t^\pi(\boldsymbol{s}_t, d_t) = \max_\pi \sum_{\boldsymbol{s}_t} P(\boldsymbol{s}_t, d_t) \sum_{\boldsymbol{a}_t} \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t),$$

in which $P(\boldsymbol{s}_t, d_t)$ is the frequency when the global state $(\boldsymbol{s}_t, d_t)$ appears at time t.

Dynamic programming approach requires the enumeration of all possible states and actions, hence is not scalable to problems with large state and action spaces, e.g. the joint state space in a $\mathbb{C}$Dec-POMDP. Instead of computing the exact value function $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$, reinforcement learning (RL) methods estimate an approximate value function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \approx Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ and update the policy based on this approximate $\tilde{Q}$ by Monte Carlo sampling. The approximate value function $\tilde{Q}_w$ can be learnt by using regression to fit empirical returns obtained from historical data or simulation. To update policy, a policy gradient is estimated from $\tilde{Q}_w$ to specify the direction to adjust policy parameters.

The reinforcement learning with value function approximation is also known as the actor-critic [117, 54] with actor referring to the policy and critic referring to the approximate value function [117].

4.2.1 Reinforcement Learning Outline

In each learning iteration, to estimate $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$, we generate samples of $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, d_t, \boldsymbol{r}_t \rangle$ and fit $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to the corresponding empirical returns form samples by using regression method. The policy is updated to increase the expected value of $\tilde{Q}_w$ at the samples $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_t \rangle$.

Following a convention in RL community, we consider the approximate value function $\tilde{Q}$ to be a function parameterized by parameters $\boldsymbol{w}$ and the policy π to be a function parameterized by parameters $\boldsymbol{\theta}$.

4.2.2 Policy Gradient

When the policy π is parameterized with $\boldsymbol{\theta}$, we can optimize π by applying a policy gradient estimation computed with the critic function $\tilde{Q}_w$. Following similar derivation with [118], we now show how to compute policy gradient $\nabla_\theta V(\pi)$.

We denote the value function of a given policy π in an expanded form is given as:

$$V_t(\pi) = \sum_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t} P(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g; \pi) Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \tag{4.7}$$

where $P(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g; \pi) = \sum_{\boldsymbol{s}_{1:t-1}, \boldsymbol{a}_{1:t-1}} P(\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, d_{1:t} | b_o, b_o^g; \pi)$ is the distribution of the joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ under the policy π.

Recall the Bellman equation of value function $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to be

$$\begin{aligned} Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) =& r_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \\ &+ \sum_{\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1}} P(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t, d_t; \pi) Q_{t+1}^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, d_{t+1}) \end{aligned} \tag{4.8}$$

The policy gradient is computed as follows:

$$\frac{\partial V_0}{\partial \theta} = \sum_{\boldsymbol{s}_0, \boldsymbol{a}_0, d_0} \nabla_\theta \Big(P(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0 | b_o, b_o^g; \pi) Q_0^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0) \Big) \tag{4.9}$$

$$= \sum_{\boldsymbol{s}_0, \boldsymbol{a}_0, d_0} Q_0^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0) \nabla_\theta P(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0 | b_o, b_o^g; \pi)$$

$$+ \sum_{\boldsymbol{s}_0, \boldsymbol{a}_0, d_0} P(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0 | b_o, b_o^g; \pi) \nabla_\theta Q_0^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0) \tag{4.10}$$

using the Q function definition for $\mathbb{C}$Dec-POMDPs and taking the derivative we get

$$= \sum_{\boldsymbol{s}_0, \boldsymbol{a}_0, d_0} Q_0^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0) \nabla_\theta P(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0 | b_o, b_o^g; \pi)$$

$$+ \sum_{\boldsymbol{s}_0, \boldsymbol{a}_0, d_0} P(\boldsymbol{s}_0, \boldsymbol{a}_0, d_0 | b_o, b_o^g; \pi) \nabla_\theta \Big[\sum_{\boldsymbol{s}_1, \boldsymbol{a}_1} P(\boldsymbol{s}_1, \boldsymbol{a}_1, d_1 | \boldsymbol{s}_0, \boldsymbol{a}_0, d_0; \pi) Q_1^\pi(\boldsymbol{s}_1, \boldsymbol{a}_1, d_1) \Big]$$

If we continue unrolling out the terms in the above expression, we get

$$= \sum_t \sum_{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}} Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) P(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, d_{t-1} | b_o, b_o^g; \pi)$$

$$\times \nabla_\theta P(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, d_{t-1}; \pi) \tag{4.11}$$

this can be re-written use the log trick

$$= \sum_t \sum_{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}} Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) P(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, d_{t-1} | b_o, b_o^g; \pi) P(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g; \pi)$$

$$\times \nabla_\theta \log P(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, d_{t-1}; \pi) \tag{4.12}$$

$$= \sum_t \sum_{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, d_{1:t}} E_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g, \pi} \Big[Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \nabla_\theta \log P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t; \pi) \Big] \tag{4.13}$$

Compatible Value Function Approximation - Unbiased Gradients

One of desirable properties of the critic function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is that it should give the same policy gradient as $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ as

$$\sum_{t} \sum_{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, d_{1:t}} E_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g, \pi} \Big[Q_t^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \nabla_{\theta} \log P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t; \pi) \Big]$$

$$= \sum_{t} \sum_{\boldsymbol{s}_{1:t}, \boldsymbol{a}_{1:t}, d_{1:t}} E_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t | b_o, b_o^g, \pi} \Big[\tilde{Q}_t^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \nabla_{\theta} \log P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t; \pi) \Big] \tag{4.14}$$

Sutton et al. [118] showed a class of critic function with this property, namely compatible value function approximation with $\tilde{Q}_t^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \boldsymbol{w} \nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t; \pi)$. In other words, the compatible critic function to be a linear function of the derivative vector of policy function is able to provide non-bias policy gradient.

In practice, $\tilde{Q}$ is considered as a neural network function instead of linear function. Nevertheless, we will show later, the compatible value function approximation provides us some hints on designing structure of the neural network based policy in Dec-POMDP and $\mathbb{C}$Dec-POMDP domains.

Actor-critic approaches

Denote α_w and α_θ to be learning rates of critic and actor respectively. Based on trajectory sample $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$, we can update the parameterized critic and the

actor using standard approaches [117] as in algorithm 2.

Algorithm 2: Actor Critic Framework

1 Initialize network parameter θ for actor π and and w for critic $\tilde{Q}_w$

2 **repeat**

3 Sample trajectory $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$

4 Computing empirical k-step prediction

$$R_t^k(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \begin{cases} \sum_{t'=t}^{t+k-1} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, d_{t'}) + \tilde{Q}_w(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, d_{t+k}) & \text{if } t+k < H \\ \sum_{t'=t}^{H-1} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, d_{t'}) + \tilde{Q}_w(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, d_{t+k}) & \text{otherwise} \end{cases} \tag{4.15}$$

5 **Critic update:** Minimize the TD difference $\sum_t \left(\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - R_t^k(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)\right)^2$ by gradient descent

$$\boldsymbol{w} = \boldsymbol{w} - \alpha_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \sum_t \left(\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - R_t^k(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)\right)^2 \tag{4.16}$$

6 **Actor update:** Maximize the surrogate objective $\sum_t \tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t)$ by gradient ascent

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \sum_t \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) \tag{4.17}$$

7 **until** ***convergence***

Notice: The surrogate objective $\sum_t \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t)$ in actor update equation (4.17) comes from re-writing the expected value of policy using the log

derivative trick

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{a}} \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d)\pi(\boldsymbol{a}|\boldsymbol{s}, d; \boldsymbol{\theta}) &= \sum_{\boldsymbol{a}} \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d)\nabla_{\boldsymbol{\theta}}\pi(\boldsymbol{a}|\boldsymbol{s}, d; \boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{a}} \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d)\pi(\boldsymbol{a}|\boldsymbol{s}, d; \boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}|\boldsymbol{s}, d; \boldsymbol{\theta}) \\ &\overset{\text{sample}}{\approx} \sum_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s},d;\boldsymbol{\theta})} \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d)\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{a}|\boldsymbol{s}, d; \boldsymbol{\theta}) \end{aligned} \tag{4.18}$$

Variants

TD(λ) : As proposed by Sutton [116], in TD critic update, instead of computing the return value estimation with a fixed k, we can consider a mixture of different values k as:

$$R_t^{\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{k=1}^{H-t-1} (1-\lambda)\lambda^{k-1} R_t^k(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) + \lambda^{H-t-1} R_t^{H-t}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \tag{4.19}$$

and update the critic to minimize TD difference with these values as $\min_{\boldsymbol{w}} \sum_t \left(\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - R_t^{\lambda}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)\right)^2$.
From now on, we refer $R_t(\bullet)$ to be either k-step prediction or λ prediction, unless specified.

REINFORCE: An earlier method used to train the actor function is the REINFORCE algorithm proposed by Williams [143]. Instead of using the critic in actor update, we can directly use the empirical returns and maximize $\sum_t R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t)$ as follows:

$$\theta = \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t)$$

REINFORCE is still among the most popular RL algorithms in many single agent domains because of its advantage in providing no-bias estimation of policy gradient.

State value function: Another common method in reinforcement learning literature

is to use *state* value function $\tilde{V}_w(\boldsymbol{s}_t, d_t)$ instead of the state-action value function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ as follows:

- Critic update:

$$G_t^k(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \begin{cases} \sum_{t'=t}^{t+k-1} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, d_{t'}) + \tilde{V}(\boldsymbol{s}_{t+k}, d_{t+k}) & \text{if } t+k < H \\ \sum_{t'=t}^{H-1} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, d_{t'}) + \tilde{V}_w(\boldsymbol{s}_{t+k}, d_{t+k}) & \text{otherwise} \end{cases}$$

$$\boldsymbol{w} = \boldsymbol{w} - \alpha_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} \sum_t \left(\tilde{V}_w(\boldsymbol{s}_t, d_t) - G_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \right)^2$$

- Actor update:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \sum_t G_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t)$$

Notice: A main problem with REINFORCE and state value functions is that they are not decomposable in a multi-agent setting. As shown later, the decomposability of policy gradient is the important and desirable property in multi-agent RL, and it is achievable with the use of Q-value in policy gradient computation. Therefore, in this thesis, we focus more on designing an efficient policy update in $\mathbb{C}$Dec-POMDP for $\tilde{Q}$ in equation (4.17).

4.2.3 Baseline subtraction

The policy gradient in the equation (4.17) is estimated by samples of state-action values. Theoretically, the sampling based estimation of the policy gradient converges to the true policy gradient when the number of samples is sufficient to approximate the state-action space. However, in practical problems like $\mathbb{C}$Dec-POMDPs, the state-action space can be exponentially large, and the convergence of policy gradient is slow. In fact, the sampling-based estimation of the policy gradient is usually a stochastic variable with high variance. To reduce the high variance

of the stochastic policy gradient estimation, a baseline $b(\boldsymbol{s}_t, d_t)$ is used as a control variate in policy gradient estimation [137, 35]. In particular, $\tilde{Q}$ is subtracted by the baseline as follows:

$$\theta = \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t [\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)] \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) \tag{4.20}$$

The baseline function is only dependent on state, therefore does not introduce an bias into the expectation of policy gradient:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{a}} b(\boldsymbol{s}, d) \pi(\boldsymbol{a} | \boldsymbol{s}, d; \boldsymbol{\theta}) &= b(\boldsymbol{s}, d) \nabla_{\boldsymbol{\theta}} \underbrace{\sum_{\boldsymbol{a}} \pi(\boldsymbol{a} | \boldsymbol{s}, d; \boldsymbol{\theta})} \\ &= b(\boldsymbol{s}, d) \nabla_{\boldsymbol{\theta}} \qquad (1) \\ &= b(\boldsymbol{s}, d) \times 0 \\ &= 0 \end{aligned}$$

Bhatnagar et al. [14] showed that the optimal baseline in actor-critic, which results into minimum variance of stochastic policy, is:

$$b^*(\boldsymbol{s}, d) = \sum_{\boldsymbol{a}} \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d) \pi(\boldsymbol{a} | \boldsymbol{s}, d; \boldsymbol{\theta}) \tag{4.21}$$

Notice: In reinforcement learning, we can unify different ways to compute the policy gradient by considering a generic update formula:

$$\theta = \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t A(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t), \tag{4.22}$$

in which the value $A(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ used to update policy is called the advantage function. $A(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ can be set to be $[R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)]$, $[G_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)]$ or $[\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)]$.

4.3 Multi-agent Reinforcement Learning

The main challenge to extend single agent reinforcement learning to multi-agent problems like Dec-POMDP or $\mathbb{C}$Dec-POMDP is the large joint state and action space. In $\mathbb{C}$Dec-POMDPs, the number of possible joint actions of M agents at each decision epoch is $|A|^M$. The exponential complexity of joint space requires our study to find out suitable form of critic functions as well as efficient policy update formulas in this section.

4.3.1 Factorization of policy in decentralized execution

Recall that by $\mathbb{C}$Dec-POMDP definition in Chapter 2, the joint policy function has the form of $P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) = \prod_m \pi^m(a_t^m|o^m(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))$.

We consider the same observation function $o^m(i, \mathrm{n}_t^{\mathrm{s}}, d_t) = o(i, \mathrm{n}_t^{\mathrm{s}}, d_t), \forall m$ and same policy $\pi^m = \pi, \forall m$ for all the agents. Therefore we would drop the superscript index m in the policy and observation function π^m, o^m, and the joint policy becomes $P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) = \prod_m \pi(a_t^m|o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))$.
Under this factored form of the joint policy, we have:

Proposition 4.3.

$$\nabla_\theta \log P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) = \sum_m \nabla_\theta \log \Big(\pi(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))\Big) \tag{4.23}$$

Proof. We simplify the above gradient as following:

$$\begin{aligned}\nabla_\theta \log P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) &= \nabla_\theta \log \Big(\prod_m \pi(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))\Big) \\ &= \sum_m \nabla_\theta \log \Big(\pi(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))\Big) \end{aligned} \tag{4.24}$$

□

Replace (4.24) into (4.20), we have a general policy gradient in $\mathbb{C}$Dec-POMDP as follows:

$$\theta = \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t A(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \sum_m \nabla_\theta \log \Big(\pi(a_t^m | s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)) \Big) \tag{4.25}$$

$$= \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t [\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)] \sum_m \nabla_\theta \log \Big(\sum_m \pi(a_t^m | s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)) \Big) \tag{4.26}$$

As shown by Peshkin et al. [90], the decomposition form (4.26) allows the decentralized policy to be efficiently updated by an individual action instead of a joint action. This reduces the exponential complexity $|A|^M$ into a linear complexity $M \times |A|$.

In the formula (4.26), although the log of the policy function is decomposed under the decentralized execution, the advantage function $[\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - b(\boldsymbol{s}_t, d_t)]$ is not decomposed amongst agents. A single number representing the non-decomposable value of the value function does not tell much about the effect of each individual action in the system. Consequently, as shown in experimental section in later chapters, when the number of agents is large, the non-decomposed global value is insufficient to update individual policy and often leads to a poor reinforcement learning solution.

To see why using a non-decomposed global value is problematic in multi-agent RL, let us consider a trivial example to optimize routing policy for decentralized vehicles in a traffic network with 2 main sectors A and B. An agent can choose to go through either sector A or B. Agents with optimized policy should be able to avoid moving into congested sector and balance the traffic in the network. Assume at a time step t, sector A has no congestion and sector B has heavily congested. As a result, the system incurs a high penalty cost, saying a reward value $\boldsymbol{r}_t = -1000$, due to the congestion at B. Without decomposing the value, we can not update the policy to mitigate the right cause of congestion. By feeding this singleton penalty

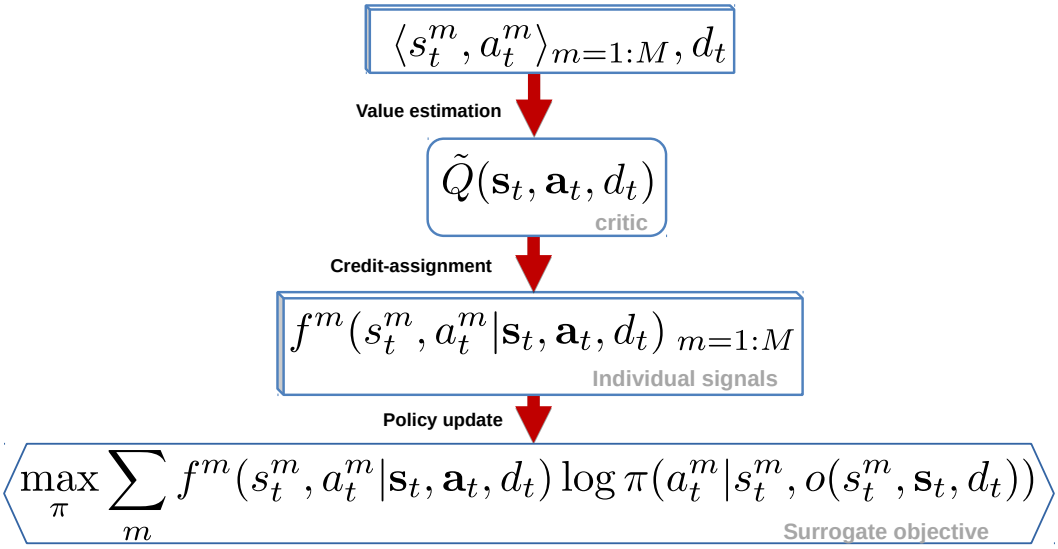


Figure 4.1: Credit-assignment in multi-agent RL.

to *all* agents, the individual policy would be updated to reduce the probability to neither move into B sector (which is desirable) nor move into A sector (which is undesirable). Ideally, we should decompose the global value into a high congestion penalty for agents in sector B and a low (or zero) congestion penalty for agents in sector A. The process of decomposing the global value to individual signals is called credit-assignment, which we study in the following section.

4.3.2 Credit-assignment

To effectively optimize individual policy with RL , Wolpert and Tumer [148] showed that we have to "personalize" the value signal for each individual agent. Specifically, the multi-agent reinforcement learning (MRL) policy update (4.25) is re-written as

$$\theta = \theta + \alpha_\theta \nabla_{\boldsymbol{\theta}} \sum_t \sum_m f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \nabla_\theta \log \Big(\pi(a_t^m | s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)) \Big) \tag{4.27}$$

in which $f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is the credit value for individual m when it participates with action a_t^m into the joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$. Credit value f^m is usually computed from the joint value function estimator $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$. The credit assignment

procedure for MRL is demonstrated by the diagram in Figure 4.1. Given the joint state action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$, we would use a global critic $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to estimate the value. Then we compute the credit value f^m for each individual m using this global critic. Finally, we use f^m to update each individual policy $\pi(a_t^m|\bullet)$.

Desirable Properties: The credit value is designed to reflect the contribution of an individual agent into the global value function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$. In expectation, the policy gradient computed by individual credit values should be the same (or close) to the one computed by the global value function. In other words, credit assignment should not introduce any bias (or a low bias) into the policy gradient. At the same time, credit assignment should be able to distinguish the roles of different agents in the system, and hence reduce the *noise* in the gradient estimation of each agent.

Notice: As our objective is to learn a homogeneous policy function π instead of multiple policy functions, we can combine policy update of all agents by a sum $\sum_t \sum_m f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \nabla_\theta \log \Big(\pi(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))$.

Credit assignment methods

We now present the most common credit assignment methods from the literature.

Vanilla: The vanilla policy update in equation (4.25) can be considered as a dummy credit assignment with $f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$.

Expected value of individual: To estimate the value of an individual action a^m, Claus and Boutilier [23] proposed that m can marginalize over all possible actions $\boldsymbol{a}^{-m}$ of other agents $m' \neq m$

$$
\begin{aligned}
& f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \\
&= \sum_{\boldsymbol{a}_t^{-m} \in A^{-m}} Q(\boldsymbol{s}_t, \boldsymbol{a}_t^{-m} \cup a_t^m, d_t) \prod_{m' \neq m} \pi(\boldsymbol{a}_t^{-m}|s_t^{m'}, o(s_t^{m'}, \boldsymbol{s}_t, d_t))
\end{aligned} \tag{4.28}
$$

This type of credit assignment is applicable to multi-agent domains with small ac-

tion space. A recent application is shown in multi-agent patrolling by Gupta et al. [41]. However, as the action space increases, computing exact marginalization in (4.28) becomes infeasible.

Counterfactual value function: Wolpert and Tumer [148] proposed the credit assignment function to be the difference reward for each agent

$$\begin{aligned} & f^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \\ & = \tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - \sum_{a'^m \in A^m} \pi(a'^m | s_t^m, o(s_t^m, \boldsymbol{s}_t, d_t)) \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t^{-m} \cup a'^m, d_t) \end{aligned} \quad (4.29)$$

In general, this formula quantifies the contribution of action a_t^m of agent m by the difference between the realized value $\tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ and average value when agent m takes alternative (counterfactual) actions a'^m. Foerster et al. [30] showed that the counterfactual baseline $\sum_{a'^m \in A^m} \pi(a'^m | s_t^m, o(s_t^m, \boldsymbol{s}_t, d_t)) \tilde{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t^{-m} \cup a'^m, d_t)$ is zero in expectation, therefore it does not introduce any bias into the policy gradient.

One of main problems with the counterfactual RL is we have to compute the counterfactual values for all possible counterfactual actions a'^m of all agents m. In other words, in the system of M agent and the action space to be A, to perform credit assignment for each sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$, we have to compute the value function $|M| \times |A|$ times. We would address this bottleneck in the $\mathbb{C}$Dec-POMDP domain later in Chapter 6.

Local reward: In a domain where the global reward $\boldsymbol{r}_t$ is the sum of local rewards $\boldsymbol{r}_t = \sum_m r_t^m$, we can use the local reward r_t^m to learn policy for an individual m. In particular, instead of a single joint value function $\tilde{Q}$, [121] proposed independent learner framework in which each agent m would maintain a separate value function $Q^m(s_t^m, a_t^m)$ to estimate the accumulative reward $\mathbb{E}[\sum_{t'=t}^{H} r_{t'}^m | s_t^m, a_t^m]$ it could collect. Bagnell and Ng [8] showed that learning with local rewards can improve the convergence compared with global reward signals. Claus and Boutilier [23] showed that learning with a local reward signal, in fact, follows the fictitious play rule [60],

hence can achieve a high quality solution.

However, in large populations, the independent learner method would require a large number of learning processes (one for an agent), which is not scalable. In Chapter 5, we address this problem by a count-based local reward RL approach.

4.3.3 Factored critic function

In a decentralized execution setting, we consider the following special form of the approximate value function f_w [115, 37, 53]:

$$Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \approx \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{m=1}^{M} f_w^m(s_t^m, a_t^m, \boldsymbol{s}_t, d_t) \tag{4.30}$$

where each f_w^m is defined for each agent m and takes as input the agent's local state, action and the observation. Notice that different components f_w^m are correlated as they depend on the global state $\boldsymbol{s}_t, d_t$. Such a decomposable form is useful as it leads to efficient policy gradient computation. Furthermore, an important class of approximate value function having this form for $\mathbb{C}$Dec-POMDPs is the *compatible value function* [118] which results in an unbiased policy gradient.

Proposition 4.4. *Compatible value function for $\mathbb{C}$Dec-POMDPs can be factorized as:*

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{m} f_w^m(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)$$

Proof. As we showed previously the result from [118], the compatible value function approximates the value function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ with linear value $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = w^T \phi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$, where w denotes function parameter vector and $\phi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is compatible feature vector computed from the policy π as

$$\phi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \nabla_\theta \log P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) \tag{4.31}$$

Applying this for $\mathbb{C}$Dec-POMDPs and using the result from proposition 4.3, we

have the linear compatible feature in a $\mathbb{C}$Dec-POMDP to be:

$$\phi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \nabla_\theta \log P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) = \sum_m \nabla_\theta \log \pi_t^m(a^m|s_t^m, o(s_t^m, \mathrm{n}_t^\mathrm{s}, d_t)) \quad (4.32)$$

We can rearrange $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ as follows:

$$\begin{aligned}\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = w^T\phi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) &= w^T\Big[\sum_m \nabla_\theta \log \pi_t(a^m|s_t^m, o(s_t^m, \mathrm{n}_t^\mathrm{s}, d_t))\Big] \\ &= \sum_m w^T \nabla_\theta \log \pi_t(a^m|s_t^m, o(s_t^m, \mathrm{n}_t^\mathrm{s}, d_t)) \quad (4.33)\end{aligned}$$

If we set $f_w^m(s_t^m, a_t^m, \mathrm{n}_t^\mathrm{s}, d_t) = w^T\nabla_\theta \log \pi_t(a^m|s_t^m, o(s_t^m, \mathrm{n}_t^\mathrm{s}, d_t))$, the theorem is proved. □

We can directly replace $\tilde{Q}(\cdot)$ in the policy gradient (4.26), which is equivalent to a naive credit assignment of local signal to global signal. Empirically, we found that variance using this estimator was high. We exploit the structure of $\tilde{Q}_w$ and show further factorization of the policy gradient, which suggests us to use directly f_w^m as credit function.

Theorem 4.1. *For any value function having the decomposition as:*

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_m f_w^m\big(s_t^m, a_t^m, \mathrm{n}_t^\mathrm{s}, d_t\big), \quad (4.34)$$

the policy gradient can be computed as

$$\nabla_\theta V_1(\pi) = \sum_{t=1}^H \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\sum_m \nabla_\theta \log \pi\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^\mathrm{s}, d_t)\big) f_w^m\big(s_t^m, a_t^m, \mathrm{n}_t^\mathrm{s}, d_t\big)\Big] \quad (4.35)$$

Proof. Substitute the approximate value function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ in the policy gradient formula (4.13), we have the policy gradient computed by

approximate value function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to be

$$
\begin{aligned}
&\nabla_\theta V_1(\pi) \\
&= \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\nabla_\theta \log P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t; \theta)\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)\Big] \\
&= \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\frac{\partial \log \prod_m \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)}{\partial\theta}\big(\sum_{m'} f_w^{m'}(s_t^{m'}, a_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big] \\
&= \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\sum_m \nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\big(\sum_{m'} f_w^{m'}(s_t^{m'}, a_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big] \\
&= \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\sum_m \nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\big(\sum_{m'\neq m} f_w^{m'}(s_t^{m'}, a_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big] \\
&+ \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\sum_m \nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\big(f_w^m(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big] \quad (4.36)
\end{aligned}
$$

Let us simplify the first term in (4.36) as follows:

$$
\mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t}\Big[\nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\big(\sum_{m'\neq m} f_w^{m'}(s_t^{m'}, a_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big]
$$

Given the independence of value functions of other agents $m' \neq m$ w.r.t. the action a_t^m of agent m, we have:

$$
\begin{aligned}
&= \mathbb{E}_{\boldsymbol{s}_t}\Bigg[\mathbb{E}_{a_t^m|\boldsymbol{s}_t}\Bigg(\nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big) \\
&\qquad \sum_{m'\neq m} \mathbb{E}_{a_t^{m'}|\boldsymbol{s}_t} f_w^{m'}(s_t^{m'}, a_t^{m'}, o(s_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t))\Bigg)\Bigg] \\
&= \mathbb{E}_{\boldsymbol{s}_t}\Bigg[\mathbb{E}_{a_t^m|\boldsymbol{s}_t}\Bigg(\nabla_\theta \log \pi^m\big(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t) \times constant\_to\_a_t^m\Bigg)\Bigg] \\
&= 0
\end{aligned}
$$

Applying this to (4.36), we can dismiss all the term of $m' \neq m$ to simplify (4.36) into (4.35). □

Optimal baseline for decentralized execution

The baseline $b(\boldsymbol{s}, d)$ is used to reduce the variance of policy gradient, with the baseline, the policy gradient would be computed in term of advantage function $A(\boldsymbol{s}, \boldsymbol{a}, d) = Q(\boldsymbol{s}, \boldsymbol{a}, d) - b(\boldsymbol{s}, d)$ instead of $Q(\boldsymbol{s}, \boldsymbol{a}, d)$. In decentralized planning case, this is computed as:

Proposition 4.5. *For any value function having the decomposition as:*

$$Q(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \approx \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_m f_w^m\big(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\big), \tag{4.37}$$

the optimal baseline can be computed as:

$$b_w(\mathrm{n}_t^s, d_t) = \sum_m b_w^m(s_t^m, \mathrm{n}_t^s, d_t), \tag{4.38}$$

in which each

$$b_w^m(s_t^m, \mathrm{n}_t^s, d_t) = \sum_{a_t^m} \pi_t(a_t^m | s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)) f_w^m\big(s_t^m, a_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)$$

Proof. Recall from [14], optimal baseline of any critic $Q(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ to reduce variance of policy gradient is:

$$b^*(\mathrm{n}_t^s, d_t) = \sum_{\boldsymbol{a}_t} P(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) Q(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t). \tag{4.39}$$

Now, substitute (4.37) into this we have:

$$\begin{aligned}
& b^*(\mathrm{n}_t^s, d_t) \\
&= \sum_{\boldsymbol{a}_t} \prod_{m'} \pi_t(a_t^{m'} | o(s_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)) [\sum_m f_w^m\big(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\big)] \\
&= \sum_{a_t^0, \ldots, a_t^M} \prod_{m'} \pi_t(a_t^{m'} | o(s_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)) [\sum_m f_w^m\big(s_t^m, a_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)]
\end{aligned} \tag{4.40}$$

For a specific m, denote $\boldsymbol{a}^{-m}$ to be the joint action of all $m' \neq m$, we have:

$$\begin{aligned}
&\sum_{a_t^0,\ldots,a_t^M} \prod_{m'} \pi_t(a_t^{m'}|o(s_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)) f_w^m\left(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right) \\
=&\sum_{a^m} f_w^m\left(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right) \times \sum_{\boldsymbol{a}^{-m}} \prod_{m' \neq m} \pi_t(a_t^{m'}|o(s_t^{m'}, \mathrm{n}_t^{\mathrm{s}}, d_t)) \\
=&\sum_{a^m} f_w^m\left(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right) \times 1
\end{aligned}$$

Notice the reason the last equation holds is due to the independence of decentralized individual policy. Apply this simplification for each m, we can simplify (4.40) into (4.38). □

We summarize theorem 4.1 and proposition 4.5 by

Corollary 4.1. *When the critic function is decomposed as:*

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_m f_w^m\left(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right)$$

the policy gradient can be computed by:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \sum_t \sum_m \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t} \Big[&\left(f_w^m\left(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right) - b^m(s_t^m, \mathrm{n}_t^s, d_t)\right) \\
&\times \nabla_\theta \log\left(\pi(a_t^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t))\right)\Big]
\end{aligned} \tag{4.41}$$

in which $b^m(s_t^m, \mathrm{n}_t^s, d_t) = \sum_{a_t'^m} \pi_t(a_t'^m|s_t^m, o(s_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t)) f_w^m\left(s_t^m, a_t'^m, \mathrm{n}_t^{\mathrm{s}}, d_t\right)$

Notice that computing the policy gradient using the above result is not practical for multiple reasons. The space of join-state action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is combinatorial. Given that the agent population size can be large, sampling each agent's trajectory is not computationally tractable. To remedy this, we later show how to compute the gradient by directly sampling counts $\mathbf{n} \sim P(\mathbf{n}; \pi)$ similar to policy evaluation in equation (2.15). Similarly, one can estimate the action-value function $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ using empirical returns as an approximation. This would be the analogue of the

standard REINFORCE algorithm [144] for $\mathbb{C}$Dec-POMDPs. It is well known that REINFORCE may learn more slowly than other methods that use a learned action-value function [118]. Therefore, we next present a function approximator for Q_t^π, and show the computation of policy gradient by directly sampling counts $\mathbf{n}$.

4.4 Collective Reinforcement Learning

In general, when dealing with homogeneous agents, we can aggregate agents in the same state by the state-actions count. The log-likelihood of the joint action probability in proposition 4.3 can be simplified as follows:

$$\nabla_\theta \log P(\boldsymbol{a}_t|\boldsymbol{s}_t, d_t) = \sum_{i\in S, j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) \nabla_\theta \log \pi_t(j|o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) \tag{4.42}$$

Using the above results, the final policy gradient expression for $\mathbb{C}$Dec-POMDPs is readily proved.

Theorem 4.2. *For $\mathbb{C}$Dec-POMDPs, the policy gradient is given as:*

$$\nabla_\theta V_1(\pi) = \sum_{t=1}^{H} E_{\mathrm{n}_t^{\mathrm{s}}, \mathrm{n}_t^{\mathrm{sa}}, d_t | b_o, b_o^g; \pi} \Big[Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t) \sum_{i\in S, j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) \nabla_\theta \log \pi_t(j|o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) \tag{4.43}$$

Proof. This result is directly implied by substitute (4.42) into (4.13) and notice that $\mathrm{n}_t^{\mathrm{sa}}, d_t$ are sufficient statistics in $\mathbb{C}$Dec-POMDP. □

4.4.1 Policy Gradient with Factored Collective Critic

In a population of homogeneous agents, we have the same function f_w^m for each agent, and deduce the following:

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w\big(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t\big) \tag{4.44}$$

Theorem 4.3. *For any value function having the form:* $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w\big(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t\big)$*, the policy gradient can be computed as:*

$$\mathbb{E}_{\mathbf{n}_{1:H} \in \Omega_{1:H}, d_{1:H}} \Bigg[\sum_{t=1}^{H} \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) \nabla_\theta \log \pi\big(j | i, o(i, \mathrm{n}_t^s, d_t)\big) f_w(i, j, \mathrm{n}_t^s, d_t) \Bigg] \tag{4.45}$$

Proof. By aggregating agents in similar state-action in theorem 4.1 and (4.44), we have:

$$\nabla_\theta V_1(\pi) = \sum_t \mathbb{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t, d_t} \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j | i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)}{\partial \theta} f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \Big] \tag{4.46}$$

We can expand the above expression as:

$$\nabla_\theta V_1(\pi) = \sum_{\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H}} P(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H}) \Big[\sum_{t=1}^{H} \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j | i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)}{\partial \theta} f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \Big]$$

From Chapter 2, we know that the probability $P(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$ depends only on counts $\mathbf{n}$ generated by the joint-state and action trajectory $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$ and is equal to the corresponding joint probability $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T})$ in (2.3). Using this result, we have:

$$\begin{aligned} &\nabla_\theta V_1(\pi) \\ &= \sum_{\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H}} f(\mathbf{n}_{1:H}) \Big[\sum_{t=1}^{H} \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j | i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)}{\partial \theta} f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \Big] \end{aligned}$$

Notice that the entire expression inside the summation above depends only on the

resulting counts $\mathbf{n}_{1:H}$. We also know from Chapter 2 that the multinomial coefficient $h(\mathbf{n}_{1:H})$ in (2.5) counts the total number of ordered M state-action trajectories with sufficient statistic equal to $\mathbf{n}_{1:H}$ as was stated previously. Therefore, we can replace the summation over $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$ by summation over all the possible valid counts $\mathbf{n}_{1:H} \in \Omega_{1:H}$ and multiply the inner expression by $h(\cdot)$ to get $\nabla_\theta V_1(\pi)$:

$$= \sum_{\mathbf{n}_{1:H} \in \Omega_{1:H}, d_{1:H}} h(\mathbf{n}_{1:H}) f(\mathbf{n}_{1:H}) \Big[\sum_{t=1}^{H} \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j|i, o(i, \mathrm{n}_t^s, d_t)\big)}{\partial \theta} f_w(i, j, \mathrm{n}_t^s, d_t) \Big]$$

$$= \sum_{\mathbf{n}_{1:H} \in \Omega_{1:H}, d_{1:H}} P(\mathbf{n}_{1:H}) \Big[\sum_{t=1}^{H} \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j|i, o(i, \mathrm{n}_t^s, d_t)\big)}{\partial \theta} f_w(i, j, \mathrm{n}_t^s, d_t) \Big]$$

The above equation proves the theorem. □

The above result shows that the policy gradient can be computed by sampling count table vectors $\mathbf{n}_{1:H}$ from the underlying distribution $P(\cdot)$ analogous to computing the value function of the policy in (2.15), which is tractable even for large population sizes. As a result of theorem 4.3, we have

Corollary 4.2. *The count-based policy gradient can be estimated by a count* $\mathbf{n}_t$ *as follows:*

$$\mathbb{E}_{\mathbf{n}_t \in \Omega_t, d_t} \Big[\nabla_{\boldsymbol{\theta}} \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \frac{\partial \log \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)}{\partial \theta} [f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) - b(i, \mathrm{n}_t^{\mathrm{s}}, d_t)] \Big] \tag{4.47}$$

in which $b(i, \mathrm{n}_t^{\mathrm{s}}, d_t) = \sum_{j'} \pi_t(j'|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$

We find that the factorization of linear collective critic $\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ implies the credit value $f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ for an agent in i taking action j. We demonstrate the credit-assignment process in collective planning by the diagram in Figure 4.2. By comparing the stochastic policy gradient in collective formulas (4.47) and (4.41), we can consider the collective credit-assignment as obtained by aggregating similar credit values of agents in the same local state-action.

One of the main advantages of the count-based update over the individual update is that the complexity of our RL is not dependent on the population size. This is the basis for us to derive efficient count-based RL algorithms in next chapters.

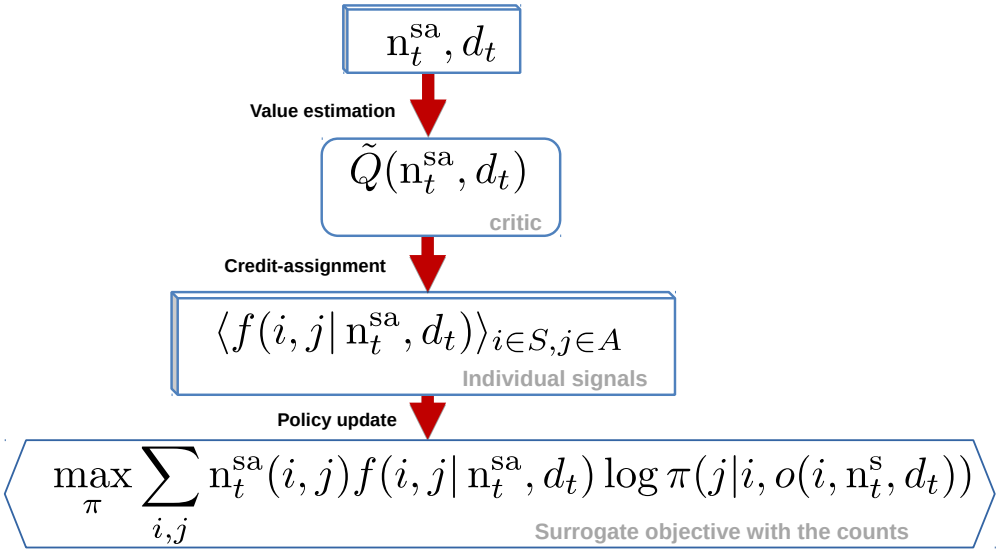


Figure 4.2: Credit-assignment in Collective RL.

4.5 Related Works

4.5.1 Model-based planning

The most common technique to solve single agent (PO)MDPs is dynamic programming [13, 43], which is a backward induction based on the Bellman equation to compute the optimal accumulated reward $Q^t(s,a)$ of each state-action pair in the current time step. However, because the exact dynamic programming requires the enumeration over all possible state-action pairs, it is not scalable in Dec-(PO)MDPs with exponential joint space of states and actions [12]. In a special case, when the closed-form formula for joint value function is available, another way to scale up the dynamic update is to search for local optimum by letting agents take turn to perform *best response* to improve their policy [74]. In general, to *tame* the exponential enumeration in Dec-(PO)MDPs dynamic programming, there has been

substantial research focusing on approximating Dec-(PO)MDPs with point-based backup approximation. In point-based backup, instead of enumerating over all belief states, the dynamic programming update is only performed over a smaller set of belief states (points). The points can be generated randomly by an online DEC-(PO)MDPs policy [119] or using a limited number of most likely observations [105] or basing on Bellman inequality [151]. All of these approximation approaches are limited to small number (less than 10) of agents and are difficult to scale up to realistic domains.

Toussaint et al. [123] proposed a novel method to recast MDP planning problems into inference problems by re-writing the reward into an auxiliary likelihood function $P(\hat{r} = 1|s_t, a_t) \propto r(s_t, a_t)$. One advantage of this probabilistic inference representation is that we can apply an advanced inference technique called Expectation Maximization [25] (EM) to solve it. Standard EM algorithm involves a backward value computation similar to dynamic programming and a forward state frequency computation. When EM is applied to solve planning problem, Schulman et al. [101] proved the objective of EM is equivalent to a surrogate policy loss function in reinforcement learning. Therefore, EM and RL both optimize closely related objectives.

Kumar et al. [58] extended the approach of Toussaint et al. [123] to multi-agent settings. They proposed an efficient EM algorithm to solve independent transition Dec-(PO)MDPs problem in MAS, i.e. the transition of an agent is independent of others but agents are related by some pairwise reward functions. By exploiting the sparse graphical structure of Dec-(PO)MDPs, the EM algorithm in [58] could be implemented as a parallelizable message-passing procedure. For general multi-agent problems with no independent transition, due to high coupling constraints between agents' variables, EM updates are computationally expensive.

4.5.2 Reinforcement Learning

In [143], Williams derived one of the first policy gradient algorithms to train neural network based policy called REINFORCE. The REINFORCE algorithm computes the policy gradient directly by the empirical return values. Training with REINFORCE has low bias but high variance [91], which implies a slow convergence rate of the algorithm. One of well-known remedies for this situation is to use an approximate value function in place of empirical returns to estimate value functions. This is known as the actor-critic method [55], in which "actor" stands for policy function and "critic" stands for approximate value function. In [118], Sutton et al. provided the general formulas for critic-update to train the critic with the empirical returns and actor-update to train the actor with critic values. These two works [143] and [118] lay the foundation for development of later advanced policy gradient methods such as natural policy gradient [14], trust region policy optimization [102], etc. The derivation of our collective policy gradient algorithms for $\mathbb{C}$Dec-POMDP is also based these two seminal works.

The biologically inspired neural network model has been known for a long time to be able to solve difficult planning problems [10]. Recently, AI systems with neural network model are shown to achieve human-level in complex video game [70] or even defeat humans in the challenging game of Go [109]. This success of neural network model is due to the progress of reinforcement learning to train neural network based policy. Two main classes of reinforcement learning algorithms are Q-learning, in which the system learns a value function and executes the greedy action according to that model; and policy gradient, in which the system adjusts a policy function by a policy gradient with regard to system value function [117]. In congestion related domains like our goal navigation or taxi matching scenarios, we prefer the policy gradient method to find stochastic policy rather than Q-learning whose greedy nature could cause high levels of congestion. Hence, in this work, we focus on developing policy gradient methods for $\mathbb{C}$Dec-POMDP.

4.5.3 Multi-agent reinforcement learning

Directly applying single agent algorithm into multi-agent systems could lead to poor solutions because the individual could not distinguish its contribution in the entangled policy and value functions [23, 121, 148]. Therefore, much effort has been spent to derive efficient reinforcement learning algorithms for multi-agent systems with neural network policy based policies.

Peshkin et al. [90] showed that using decentralized policy in MAS could help to *untangle* the policy gradient for each individual policy. More specifically, because the action probability function π^m of each agent m is independent of the parameters of other agents, the derivative of the joint action probability could be factorized into a sum of disjoint derivatives of independent policies. Peshkin et al. [90] exploited this property to propose a efficient algorithm for MAS using the REINFORCE algorithm [144]. Later on, Aberdeen [1] showed that learning decentralized policy using Peshkin et al. [90]'s algorithm could outperform centralized real-time reinforcement algorithm in job scheduling problems and also worked well in Dec-(PO)MDPs. Our decentralized planning research inherits the policy factorization property from [90], however we study actor-critic with a policy gradient decomposed by structural value function approximation. When studying MAS reinforcement learning with coordination graphs, Guestrin et al. [39] proposed a factorized policy gradient method to fine-tune the Q-learning. However, Guestrin et al. [39] used REINFORCE based method with global reward signal and their policy method was limited mainly to coordination graph domains where agents can freely communicate.

In large MAS, it becomes difficult for agent to discern the effect of its actions on the global utility. As pointed out by Wolpert and Tumer [148], the difficulty of *extracting* individual contributions from a global reward signal comes from the "noise" of the activity of other concurrently *active* agents. In particular, because policies could be stochastic or all agents could concurrently change their policies in learning, the empirical global utility values could become high variance random

variables in the perspective of each agent. The common solution for this problem is to let each individual agent learn an individual value function and optimize its policy based on its own individual estimation of reward. This solution can be considered as a hybrid of fictitious play in game-theory and reinforcement learning [23]. One of the earliest methods of this class is the independent learner proposed for Q-learning in [121], in which each agent learns a Q-function for its local observation and action based on local reward signals and acts greedily according to its individual Q-function. Chang et al. [20] showed that the local value function learning with local reward signals converge faster to good solutions as opposed to learning with global reward signal. Decentralized reinforcement learning with local reward is successfully applied to multiple complex domains such as traffic control [142] or coordination graph problem [53] etc. The advantage of the local reward signal is also exploited in one of our proposed actor-critic algorithm called fAfC showed later.

When multiple agents concurrently update their value function estimations and are consequently changing their behaviors, the environment could become non-stationary in the learning process of each agent. Several methods have been proposed to stabilize the individual value function estimation MAS reinforcement learning by avoiding abrupt change in the agent's behavior. Bowling and Veloso [16] introduced a policy hill climbing called WoLF (or Win or Learn Fast) algorithm to adaptively reduce the learning rate when the expected utility is improved (win case) and vice versa. The intuitive purpose of slowing down the learning rate is to let agents wait on each other to arrive at a good solution. When studying stateless MAS, Panait et al. [87] proposed an annealing learning rate scheme called Lenient Reinforcement learning to allow agent to tolerate each other in the exploration phase. Later, lenient RL is extended to sequential MAS in [139] and deep MAS in [86]. Similar to this, we can avoid abrupt behavioral change by a setting sufficiently small learning rate in actor-critic algorithm.

4.5.4 Credit Assignment And Value Function Decomposition

Although decentralized control system has lower execution complexity than centralized control, training individual policy is difficult due to the interaction between agents. The transition of an agent depends not only on its local state and action but also the others and the global objective involves all agents in the system. Under this entanglement, Wolpert and Tumer [148] showed that it is vital for each individual to determine its role in the system by an individual value function. Claus and Boutilier [23] showed that using the *noisy* global value to adjust policy could mislead agent into a bad behavior . The process of learning individual value function is also considered as the "credit assignment" problem in multi-agent reinforcement learning literature [2]. The complexity of the credit assignment depends on the structure of value functions to be learnt.

There is a large body of literature proposing different credit assignment schemes to improve the global value based multiagent RL training. One of first works addressed the value function decomposition in multi-agent reinforcement learning is [100]. When studying distributed control in a power grid problem in [100], the authors showed that training policy of each individual agent with local rewards could be better than training with global reward in some network topology. Interestingly, the authors in [100] found out the best choice of the local reward signal to train an agent's policy was not just its own immediate reward but the combination of its immediate reward with the average of its future reward and its related neighbors. This notion of the average over related agents shares some characteristics with our method to compute the average individual value by using the counts as in Section 5.2.2 of this thesis. The benefit of using local value function is experimentally confirmed in [20]. Bagnell and Ng [8] proved that the sampling complexity of local value function learning is substantially slower than global value function learning, which explains the faster convergence of reinforcement learning algorithm with local value functions to RL algorithm with global value function.

In many MAS domains, the global reward could be represented as a sum of local rewards, hence it is a common practice to train local value functions using the local reward signal. One popular method is independent Q-learning [121] in which each agent uses its own local state and local reward signal to train a local value function and treat other agents' actions as environmental variables. The independent Q-learning has proved to work well in many multi-agent domains [121, 65, 157]. It is also used in coordination graph algorithms, which update local value function by using only a graphically related local reward [53]. Another successful application of this local reward signal methodology is traffic light control domains, where a decentralized traffic light controlling agent in each corner can be trained by local reward signals of crossing-by vehicles[142, 59, 129].

It is shown in [147] that maximization of the global utility of MAS could be obtained by letting agents optimize some local value functions given that the local value function could well-reflect the global utility. The local reward function can be designed to evaluate the contribution of each agent's local action towards the global utility. To optimize policy in stateless MAS, Claus and Boutilier [23] proposed a method called joint action learner (JAL) in which a centralized system learns a joint value function $Q(\boldsymbol{a})$ and each agent m could access both this joint value function and policies π^{-m}. The local value function $Q^m(a^m)$ is computed as the conditional expectation of value function $Q(\boldsymbol{a})$ given the local action of the agent m to be a^m.

Another method to derive local value function from global value function is to use the difference-of-reward. Tumer et al. [128] proposed the individual pay-off value function of a local action by the difference between global utility of that action and the expected value of global utility over all possible actions of that individual agent m. Recently, Foerster et al. exploits this idea into the actor-critic framework to train MAS with neural-network policies [30].

We notice that in these existing methods, maintaining individual value functions has the complexity increasing with the number of agents, hence is not efficient with

for large populations. One of our main contributions in this work is the proposal of the count-based individual value function framework, which marginalizes individual values by the counts. Our count-based approach's complexity is not affected by the population size, therefore is effective to solve large scale multi-agent RL problems.

4.6 Summary

In this chapter, we studied the multi-agent planning framework for $\mathbb{C}$Dec-POMDP. We showed that we could *lift* the dynamic programming (DP) for MDP with the complex joint state-action variables into a more tractable DP with the count variables. However, we showed that the complexity of exact dynamic programming with the counts is still exponential, which motivated us to develop a model-free reinforcement learning approach.

To develop $\mathbb{C}$Dec-POMDP reinforcement learning, we considered policy gradient with value function approximation as in [118]. By exploiting the decentralized execution in $\mathbb{C}$Dec-POMDPs, we showed that the policy gradient for $\mathbb{C}$Dec-POMDPs can be factored when the value function approximation (the critic) is decomposable. We justified this by showing the compatible value function approximation in $\mathbb{C}$Dec-POMDPs also possesses the decomposable form. We pointed out the relation between the value function decomposition and the credit-assignment problem. Importantly, by marginalizing individual variables by the counts, we proposed a novel count-based reinforcement learning framework to efficiently solve RL in large population system. The count-based RL uses count variables instead of the joint state-action variables, therefore it is much more tractable when the number of agents is large. We also proposed an effective policy gradient update with the linear critic function of the counts. This establishment of count-based RL framework was the basis for us to derive collective RL algorithms in next chapters.

The relationship between our ℂDec-POMDP framework and normal MDP is summarized in diagram in Figure 4.3.

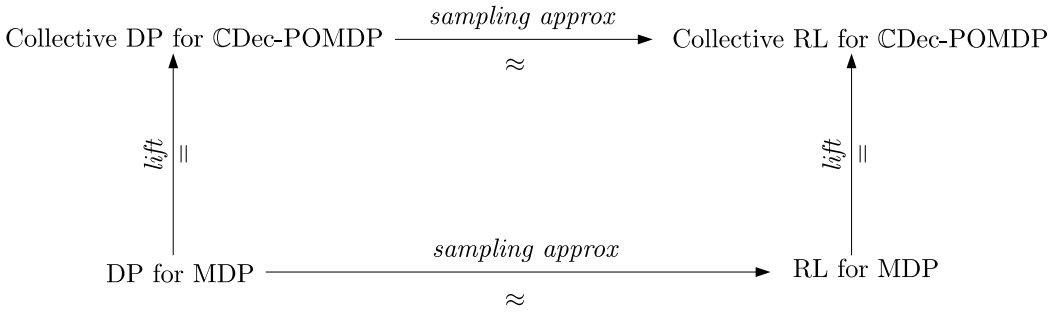


Figure 4.3: Relation between collective planning and normal MDP planning. We *lift* the original planning problems with joint state into collective planning problems with collective variables (the count).

Chapter 5

Reinforcement Learning with Local Reward Signals

In Chapter 4, we have established the basis for collective reinforcement learning by showing the sufficient statistics of the counts in planning. We have discussed the necessity of decomposing the global value functions into individual credit values to update individual policy. We showed that the credit-assignment can be done collectively in $\mathbb{C}$Dec-POMDPs by using the count and the average credit functions $f_w(i, j, \mathrm{n}^{\mathrm{s}}_t, d_t)$ for all agents in local states i taking local actions j. In this chapter, we show that $f_w(i, j, \mathrm{n}^{\mathrm{s}}_t, d_t)$ can be estimated from the individual value function. Updating individual policy with individual value function is, in fact, equivalent to a fictitious play principle in which at every iteration, each agent tries to make a small change to its policy to maximize its local reward. We show the sum of the local value functions can be an approximation of the global value function. In addition, the fictitious play based policy update can induce an evolution of population to an equilibrium.

We consider the reinforcement learning problem in which the global reward is a sum of the local rewards of individual agents. Domains with this property include traffic networks where the social welfare is computed by the sum of the delay

penalties of all the agents and taxi supply-demand matching where the objective function is the total revenue of all taxis. In this problem, individual policy can be learnt by the independent learner approach (IL) [121, 23]: each agent estimates a local value function based on its local rewards and updates its policy based on a local value function estimation. The IL is still among the most popular multi-agent RL approaches and has been applied in many multi-agent decentralized learning domains [142, 59, 8, 40]. Claus and Boutilier [23] showed that IL could be considered as a fictitious play algorithm. And as shown in the literature [60], a fictitious play principle can be applied to solve many large scale optimizations. The main problem with IL is that it scales poorly with the number of agents. In large populations of thousands of agents, maintaining thousands of individual value functions is a big challenge.

In this chapter, we show how to continue using the principle of IL in the $\mathbb{C}$Dec-POMDP model. We develop reinforcement learning by estimating the individual value function with the local reward signal and using it to optimize a decentralized policy. However, different from conventional IL, we show that individual value functions can be estimated by the sampled values of the counts instead of joint agent trajectory. For the policy update, we propose a count-based surrogate objective function to estimate the policy gradient. We justify the proposed method by showing that it is, in fact, an instance of the collective reinforcement learning framework introduced in previous chapter. Furthermore, experimentally we show that the solutions found by our proposed fictitious play based algorithm is equivalent to equilibriums found by standard evolutionary dynamics in population game theory.

5.1 Decomposable reward problems

Throughout this chapter, we assume that given the joint state-action $(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ at time t, agent m receives a local reward signal $r_t^m = r(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t)$ dependent on its local state-action (s_t^m, a_t^m) and collective variables $\mathrm{n}_t^{\mathrm{sa}}$.

The local objective for each agent m is to maximize its total expected reward $\max \sum_{t=1}^{H} \mathbb{E}[r_t^m | b_o b_o^g, \pi]$. The optimum of local objective is equivalent to an equilibrium for agents.

The global objective for the whole system is to maximize the total expected rewards of all agents over the planning horizon as

$$\max \sum_{t=1}^{H} \sum_{m} \mathbb{E}[r_t^m | b_o b_o^g, \pi]$$

Under system with homogeneous agents with the same local reward function, the global objective can be written in term of the counts

$$\max \sum_{t=1}^{H} \mathbb{E}[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) r(i,j,\mathrm{n}_t^{\mathrm{sa}}, d_t) | b_o b_o^g, \pi]$$

In this chapter, we propose algorithms to find optimum of local objective. We show that the local objective is in fact a lower bound of the global objective, hence local objective maximization can be used to produce good quality solution for global objective.

5.2 Count based Individual Value Function

In the following section, we show that the counts could be used to compute the individual value function in $\mathbb{C}$Dec-POMDP. In particular, our goal is to show the count representation of the agent m's total expected reward from time step t conditioning

on its observation $(s_t^m = i, a_t^m = j)$ and the global values $\mathrm{n}_t^{\mathrm{sa}}, d_t$:

$$Q_t(i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t) = \mathbb{E}\Big[\sum_{T=t}^{H} r_T^m | s_t^m = i, a_t^m = j, \mathrm{n}_t^{\mathrm{sa}}, d_t\Big]$$
$$= \sum_{T=t}^{H} \sum_{\mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T}} P\big(s_{1:T}^m, a_{1:T}^m, \mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T} | s_t^m = i, a_t^m = j, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) r_T\big(s_T^m, a_T^m, \mathrm{n}_T^{\mathrm{sa}}, d_T\big) \tag{5.1}$$

Notice that in equation (5.1), we need to compute the joint probability $P\big(s_{1:T}^m, a_{1:T}^m, \mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T} | s_t^m = i, a_t^m = j, \mathrm{n}_t^{\mathrm{sa}}, d_t\big)$ for each agent m. In the following section, we show how to exploit the homogeneity of agents in $\mathbb{C}$Dec-POMDPs to efficiently estimate this probability and the individual value function using only the counts.

5.2.1 Exchangeability of joint-trajectories

We start by defining *full exchangeability* [79]. A set of variables $\boldsymbol{X} = \{X_1, \ldots, X_k\}$ is fully exchangeable iff $P(X_1 = x_1, \ldots, X_k = x_k)$ equals $P(X_1 = x_{\alpha(1)}, \ldots, X_k = x_{\alpha(k)})$ for all permutations α of $\{1, \ldots, k\}$. E.g., a sequence of independent coin toss is fully exchangeable. Let $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) = \{(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})^m \ \forall m\}$ denote the T-step trajectories of all the agents. Clearly, $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ is not fully exchangeable as an agent's next state depends on its previous state. A tractable generalization of full exchangeability is partial exchangeability [27], which variables $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ would satisfy.

Definition 5.1. *Let $\mathcal{D}_i$ be the domain of X_i, and let $\mathcal{T}$ be a finite set. A set of variables $\boldsymbol{X}$ is partially exchangeable w.r.t. the statistic $T : \mathcal{D}_1 \times \cdots \times \mathcal{D}_k \to \mathcal{T}$ if and only if:*

$$T(\boldsymbol{x}) = T(\boldsymbol{x}') \text{ implies } P(\boldsymbol{x}) = P(\boldsymbol{x}')$$

We next show the following for the $\mathbb{C}$Dec-POMDP model.

Proposition 5.1. *The joint state-action trajectories of agents,* $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$*, are partially exchangeable w.r.t. the count statistic* $\mathbf{n}_{1:T} \in \Omega_{1:T}$*.*

where $\Omega_{1:T}$ is the space of allowed counts satisfying constraints (2.7)-(2.8). This result follows directly from theorem 2.1 in Section 2.3. Next we use the exchangeability theorem that relates the joint-distribution $P(\boldsymbol{X})$ over variables $\boldsymbol{X}$ with the distribution over sufficient statistic.

Proposition 5.2. *The distribution* $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T})$ *is defined as:*

$$P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}) = \sum_{\mathbf{n}_{1:T} \in \Omega_{1:T}, d_{1:T}} P(\mathbf{n}_{1:T}, d_{1:T}; \pi) \frac{\mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})}{|S_{\mathbf{n}_{1:T}}|}$$

where $\mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ *denotes if* $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ *is consistent with statistic* $\mathbf{n}_{1:T}$*;* $S_{\mathbf{n}_{1:T}}$ *is the set of all possible joint-trajectories* $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ *having sufficient statistic* $\mathbf{n}_{1:T}$*.*

This result is a direct corollary of the exchangeability theorem in [28, 79]. Notice that $|S_{\mathbf{n}_{1:T}}|$ equals to the function $h(\mathbf{n}_{1:T})$ (2.5). Let $\mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T})$ denote if agent m's trajectory $(s^m_{1:T}, a^m_{1:T})$ is consistent with the joint-trajectory $\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}$. Using this result, the joint probability $P(s^m_{1:T}, a^m_{1:T}, \mathbf{n}_{1:T})$ is:

$$\sum_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}} P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}) \mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) \mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T})$$

In the above expression, we can use proposition 5.2 to compute $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T})$. Upon further simplification, we get the following result:

Theorem 5.1. *The joint probability* $P(s^m_{1:T}, a^m_{1:T}, \mathbf{n}_{1:T}, d_{1:T})$ *is given by the following expression:*

$$= P(\mathbf{n}_{1:T}, d_{1:T}) \frac{\mathrm{n}^{\mathrm{s}}_1(s^m_1)}{M} \left[\prod_{t=1}^{T-1} \frac{\mathrm{n}^{\mathrm{sas'}}_t(s^m_t, a^m_t, s^m_{t+1})}{\mathrm{n}^{\mathrm{s}}_t(s^m_t)} \right] \frac{\mathrm{n}^{\mathrm{sa}}_T(s^m_T, a^m_T)}{\mathrm{n}^{\mathrm{s}}_T(s^m_T)} \tag{5.2}$$

Proof. According to proposition 5.2, we have

$$P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}) = \sum_{\mathbf{n}_{1:T} \in \Omega_T} P(\mathbf{n}_{1:T}, d_{1:T}) \frac{\mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})}{|S_{\mathbf{n}}|}$$

where $S_{\mathbf{n}}$ is the set of all the assignments $(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ which result in the counts $\mathbf{n}_{1:T}$. The total number of all such assignments ($|S_{\mathbf{n}}|$) is given by theorem 2.2

$$|S_{\mathbf{n}}| = h(\mathbf{n}_{1:T}) = \Big[\prod_i \frac{\mathrm{n}^{\mathrm{s}}_T(i)!}{\prod_{j \in A} \mathrm{n}^{\mathrm{sa}}_T(i,j)!}\Big]\Big[\prod_{t=1}^{T-1}\prod_i \frac{\mathrm{n}^{\mathrm{s}}_t(i)!}{\prod_{i' \in S, j \in A} \mathrm{n}^{\mathrm{sas'}}_t(i,j,i')!}\Big]\Big[\frac{M!}{\prod_{i \in S} \mathrm{n}^{\mathrm{s}}_1(i)!}\Big] \tag{5.3}$$

Let us consider the joint probability $P(s^m_{1:T}, a^m_{1:T}, \mathbf{n}_{1:T}, d_{1:T})$. It can be computed as follows:

$$\begin{aligned}
&= \sum_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}} \big[P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}) \times \mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) \mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T})\big] \\
&= \sum_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}} \big[\sum_{\mathbf{n}'_{1:T}} P(\mathbf{n}'_{1:T}, d_{1:T}) \frac{\mathbb{I}_{\mathbf{n}'_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})}{|S_{\mathbf{n}_{1:T}}|} \times \mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) \mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T})\big]
\end{aligned}$$

The terms $\mathbb{I}_{\mathbf{n}'_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ and $\mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ imply that $\mathbf{n}_{1:T} \equiv \mathbf{n}'_{1:T}$. Hence, we can further simplify the above expression as:

$$\begin{aligned}
&= P(\mathbf{n}_{1:T}, d_{1:T}) \times \sum_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}} \frac{\mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})}{|S_{\mathbf{n}_{1:T}}|} \mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T}) \\
&= P(\mathbf{n}_{1:T}, d_{1:T}) \times \frac{\sum_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, d_{1:T}} \mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}) \mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T}, a^m_{1:T})}{|S_{\mathbf{n}_{1:T}}|}
\end{aligned} \tag{5.4}$$

Notice that the numerator in the above equation counts the total number of state-action trajectories of $M-1$ individuals when combined with the given trajectory $(s^m_{1:T}, a^m_{1:T})$ of the agent m results in sufficient statistic being $\mathbf{n}_{1:T}$. This count is given as follows:

$$
\begin{aligned}
&\sum_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T},d_{1:T}} \mathbb{I}_{\mathbf{n}_{1:T}}(\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T})\mathbb{I}_{\boldsymbol{s}_{1:T}\boldsymbol{a}_{1:T}}(s^m_{1:T},a^m_{1:T}) \\
&= \frac{(M-1)!}{\prod_{i\in S}(\mathrm{n}^{\mathrm{s}}_1(i)-\mathbb{I}_{s^m_1}(i))!} \times \prod_i \frac{\big(\mathrm{n}^{\mathrm{s}}_T(i)-\mathbb{I}_{s^m_T}(i)\big)!}{\prod_{j\in A}\big(\mathrm{n}^{\mathrm{sa}}_T(i,j)-\mathbb{I}_{s^m_T a^m_T}(i,j)\big)!} \\
&\qquad \times \prod_{t=1}^{T-1}\Big[\prod_i \frac{\big(\mathrm{n}^{\mathrm{s}}_t(i)-\mathbb{I}_{s^m_t}(i)\big)!}{\prod_{i'\in S,j\in A}\big(\mathrm{n}^{\mathrm{sas}'}_t(i,j,i')-\mathbb{I}_{s^m_t a^m_t s^m_{t+1}}(i,j,i')\big)!}\Big]
\end{aligned} \tag{5.5}
$$

Substitute (5.5) and (5.3) into (5.4), we prove the theorem. □

One important corollary of theorem 5.1, which will be used to compute individual value function, is:

Corollary 5.1. *The joint conditional probability of the counts and individual trajectory can be computed as*

$$
\begin{aligned}
&P\big(s^m_{1:T},a^m_{1:T},\mathbf{n}_{1:T},d_{1:T}|s^m_1=i,a^m_1=j,\mathrm{n}^{\mathrm{sa}}_1,d_1\big) \\
&= P(\mathbf{n}_{1:T},d_{:T}|\mathrm{n}^{\mathrm{sa}}_1,d_1)\frac{\mathrm{n}^{\mathrm{sas}'}_1(s^m_1,a^m_1,s^m_2)}{\mathrm{n}^{\mathrm{sa}}_1}\Big[\prod_{t=2}^{T-1}\frac{\mathrm{n}^{\mathrm{sas}'}_t(s^m_t,a^m_t,s^m_{t+1})}{\mathrm{n}^{\mathrm{s}}_t(s^m_t)}\Big]\frac{\mathrm{n}^{\mathrm{sa}}_T(s^m_T,a^m_T)}{\mathrm{n}^{\mathrm{s}}_T(s^m_T)}
\end{aligned} \tag{5.6}
$$

Proof. Using Bayesian theorem, we have the conditional probability to be:

$$
\begin{aligned}
&P\big(s^m_{1:T},a^m_{1:T},\mathbf{n}_{1:T},d_{1:T}|s^m_1=i,a^m_1=j,\mathrm{n}^{\mathrm{sa}}_1,d_1\big) \\
&= P(\mathbf{n}_1|\mathrm{n}^{\mathrm{sa}}_1,d_1)P\big(s^m_{1:T},a^m_{1:T},\mathbf{n}_{1:T},d_{1:T}|s^m_1=i,a^m_1=j,\mathbf{n}_1,d_1\big) \\
&= P(\mathbf{n}_1|\mathrm{n}^{\mathrm{sa}}_1,d_1)\frac{P\big(s^m_{1:T},a^m_{1:T},\mathbf{n}_{1:T},d_{1:T}\big)}{P\big(s^m_1,a^m_1,\mathbf{n}_1,d_1\big)}
\end{aligned} \tag{5.7}
$$

By applying theorem 5.1 for $T=1$, the denominator in LHS of (5.7) is:

$$
P\big(s^m_1,a^m_1,\mathbf{n}_1,d_1\big) = P(\mathbf{n}_1,d_1)\frac{\mathrm{n}^{\mathrm{s}}_1(s^m_1)}{M}\frac{\mathrm{n}^{\mathrm{sa}}_1(s^m_1,a^m_1)}{\mathrm{n}^{\mathrm{s}}_1(s^m_1)} \tag{5.8}
$$

Apply (5.8) and (5.2) into (5.7) and notice

$$P\big(\mathbf{n}_{1:T}, d_{1:T}|\mathrm{n}_1^{\mathrm{sa}}, d_1\big) = P(\mathbf{n}_1|\mathrm{n}_1^{\mathrm{sa}}, d_1)P\big(\mathbf{n}_{1:T}, d_{1:T}|\mathbf{n}_1, d_1\big) = \frac{P\big(\mathbf{n}_{1:T}, d_{1:T}\big)}{P(\mathbf{n}_1, d_1)},$$

we have (5.6). □

Another corollary of theorem 5.1, which will be used later in section 5.4, is:

Corollary 5.2. *The joint probability of the counts and individual state-action can be computed as:*

$$P\big(s_T^m, a_T^m, \mathbf{n}_{1:T}, d_{1:T}\big) = P(\mathbf{n}_{1:T}, d_{:T})\frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{M} \tag{5.9}$$

Proof. We have

$$\begin{aligned}
&P\big(s_T^m, a_T^m, \mathbf{n}_T, d_T\big) \\
&= \sum_{s_{1:T}^m, a_{1:T}^m} P\big(s_{1:T}^m, a_{1:T}^m, \mathbf{n}_{1:T}, d_{1:T}\big) \\
&= \sum_{s_{1:T}^m, a_{1:T}^m} P(\mathbf{n}_{1:T}, d_{1:T})\frac{\mathrm{n}_1^{\mathrm{s}}(s_1^m)}{M}\left[\prod_{t=1}^{T-1}\frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m, a_t^m, s_{t+1}^m)}{\mathrm{n}_t^{\mathrm{s}}(s_t^m)}\right]\frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)} \\
&= P(\mathbf{n}_{1:T}, d_{1:T})\sum_{s_{1:T}^m, a_{1:T}^m}\frac{\mathrm{n}_1^{\mathrm{s}}(s_1^m)}{M}\left[\prod_{t=1}^{T-1}\frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m, a_t^m, s_{t+1}^m)}{\mathrm{n}_t^{\mathrm{s}}(s_t^m)}\right]\frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)}
\end{aligned} \tag{5.10}$$

Given $\mathbf{n}_{1:T}$, we set

$$P^{\mathbf{n}}(s_T^m, a_T^m) = \sum_{s_{1:T}^m, a_{1:T}^m}\frac{\mathrm{n}_1^{\mathrm{sas}'}(s_1^m, a_1^m, s_2^m)}{\mathrm{n}_1^{\mathrm{sa}}}\left[\prod_{t=2}^{T-1}\frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m, a_t^m, s_{t+1}^m)}{\mathrm{n}_t^{\mathrm{s}}(s_t^m)}\right]\frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)}. \tag{5.11}$$

We prove by induction that $P^{\mathbf{n}}(s_T^m, a_T^m) = \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_1^{\mathrm{sa}}}$.

With $T = 1$, $P^{\mathbf{n}}(s_1^m, a_1^m) = \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_1^{\mathrm{sa}}}$ obviously holds.

Assume $P^{\mathbf{n}}(s_T^m, a_T^m) = \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_1^{\mathrm{sa}}}$ holds for $T \geq 1$, we prove it for $T+1$ as follows:

$$
\begin{aligned}
P^{\mathbf{n}}(s_{T+1}^m, a_{T+1}^m) &= \sum_{s_T^m, a_T^m} P^{\mathbf{n}}(s_T^m, a_T^m) \frac{\mathrm{n}_T^{\mathrm{s}}(s_T^m)}{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)} \frac{\mathrm{n}_T^{\mathrm{sas}'}(s_T^m, a_T^m, s_{T+1}^m)}{\mathrm{n}_t^{\mathrm{s}}(s_T^m)} \frac{\mathrm{n}_{T+1}^{\mathrm{sa}}(s_{T+1}^m, a_{T+1}^m)}{\mathrm{n}_{T+1}^{\mathrm{s}}(s_{T+1}^m)} \\
&= \sum_{s_T^m, a_T^m} P^{\mathbf{n}}(s_T^m, a_T^m) \frac{\mathrm{n}_T^{\mathrm{sas}'}(s_T^m, a_T^m, s_{T+1}^m)}{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)} \frac{\mathrm{n}_{T+1}^{\mathrm{sa}}(s_{T+1}^m, a_{T+1}^m)}{\mathrm{n}_{T+1}^{\mathrm{s}}(s_{T+1}^m)} \\
&\quad \text{using induction hypothesis, we have} \\
&= \sum_{s_T^m, a_T^m} \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{M} \frac{\mathrm{n}_T^{\mathrm{sas}'}(s_T^m, a_T^m, s_{T+1}^m)}{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)} \frac{\mathrm{n}_{T+1}^{\mathrm{sa}}(s_{T+1}^m, a_{T+1}^m)}{\mathrm{n}_{T+1}^{\mathrm{s}}(s_{T+1}^m)} \\
&= \sum_{s_T^m, a_T^m} \frac{\mathrm{n}_T^{\mathrm{sas}'}(s_T^m, a_T^m, s_{T+1}^m)}{\mathrm{n}_{T+1}^{\mathrm{s}}(s_{T+1}^m)} \frac{\mathrm{n}_{T+1}^{\mathrm{sa}}(s_{T+1}^m, a_{T+1}^m)}{M}
\end{aligned}
\tag{5.12}
$$

Notice that $\sum_{s_T^m, a_T^m} \frac{\mathrm{n}_T^{\mathrm{sas}'}(s_T^m, a_T^m, s_{T+1}^m)}{\mathrm{n}_{T+1}^{\mathrm{s}}(s_{T+1}^m)} = 1$, we prove the induction hypothesis for $T+1$.

Now, using $P^{\mathbf{n}}(s_T^m, a_T^m) = \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{M}$ into (5.10), we have (5.9). □

5.2.2 Individual value function

Based on theorem 5.1, we now show how to compute the value function $Q_t(i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t)$. Substituting the expression for joint probability in theorem 5.1 into value function in (5.1) for $t = 1$, we have:

$$
\begin{aligned}
&Q_1(i, j, \mathrm{n}_1^{\mathrm{sa}}, d_1) \\
&= r_1\big(s_1^m, a_1^m, \mathrm{n}_1^{\mathrm{sa}}, d_1\big) + \\
&\sum_{T=2}^{H} \sum_{\mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T}} \sum_{s_{1:T}^m, a_{1:T}^m} P\big(s_{1:T}^m, a_{1:T}^m, \mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T} | s_1^m = i, a_1^m = j, \mathrm{n}_1^{\mathrm{sa}}, d_1\big) r_T\big(s_T^m, a_T^m, \mathrm{n}_T^{\mathrm{sa}}, d_T\big) \\
&= r_1\big(s_1^m, a_1^m, \mathrm{n}_1^{\mathrm{sa}}, d_1\big) + \sum_{T=2}^{H} \sum_{\mathrm{n}_{1:T}^{\mathrm{sa}}, d_{1:T}} \sum_{s_{1:T}^m, a_{1:T}^m} P(\mathbf{n}_{1:T}, d_{1:T} | \mathrm{n}_1^{\mathrm{sa}}, d_1) \\
&\quad \times \frac{\mathrm{n}_1^{\mathrm{sas}'}(s_1^m, a_1^m, s_2^m)}{\mathrm{n}_1^{\mathrm{sa}}} \left[\prod_{t=2}^{T-1} \frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m, a_t^m, s_{t+1}^m)}{\mathrm{n}_t^{\mathrm{s}}(s_t^m)} \right] \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)} r_T(s_T^m, a_T^m, \mathrm{n}_T^{\mathrm{sa}}, d_T)
\end{aligned}
\tag{5.13}
$$

For $t > 1$, by scrolling the start-time $t_0 = 1$ forward we can have the expression

$$
\begin{aligned}
&Q_t(i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t) \\
&= r_t\big(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) + \\
&\sum_{T=t+1}^{H} \sum_{\mathrm{n}_{t:T}^{\mathrm{sa}}, d_{t:T}} \sum_{s_{t:T}^m, a_{t:T}^m} P\big(s_{t:T}^m, a_{t:T}^m, \mathrm{n}_{t:T}^{\mathrm{sa}}, d_{t:T} | s_t^m = i, a_t^m = j, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) r_T\big(s_T^m, a_T^m, \mathrm{n}_T^{\mathrm{sa}}, d_T\big) \\
&= r_t\big(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) + \sum_{T=t+1}^{H} \sum_{\mathrm{n}_{t:T}^{\mathrm{sa}}, d_{t:T}} \sum_{s_{t:T}^m, a_{t:T}^m} P(\mathbf{n}_{t:T}, d_{t:T} | \mathrm{n}_t^{\mathrm{sa}}, d_t) \\
&\quad \times \frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m, a_t^m, s_2^m)}{\mathrm{n}_t^{\mathrm{sa}}} \left[\prod_{t'=t+1}^{T-1} \frac{\mathrm{n}_{t'}^{\mathrm{sas}'}(s_{t'}^m, a_{t'}^m, s_{t+1}^m)}{\mathrm{n}_{t'}^{\mathrm{s}}(s_{t'}^m)} \right] \frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m, a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)} r_T(s_T^m, a_T^m, \mathrm{n}_T^{\mathrm{sa}}, d_T)
\end{aligned}
\tag{5.14}
$$

Exactly computing the above expression is intractable due to the combinatorial space of counts $\mathbf{n}$. Therefore, we consider the Monte-Carlo approximation of the above expression by a set of samples $\{\mathbf{n}_{1:T}, d_{1:T} \sim P(\mathbf{n}_{1:T}, d_{1:T} | \mathrm{n}_1^{\mathrm{sa}}, d_1)\}$. For each

sample $\mathbf{n}$ of the counts, we compute the sampled value of (5.14) as

$$Q_t^{\mathbf{n},\boldsymbol{d}}(i,j,\mathrm{n}_t^{\mathrm{sa}},d_t) = r_t\big(s_t^m,a_t^m,\mathrm{n}_t^{\mathrm{sa}},d_t\big) + \sum_{T=t+1}^{H}\sum_{\mathrm{n}_{t:T}^{\mathrm{sa}},d_{t:T}}\sum_{s_{t:T}^m,a_{t:T}^m}\frac{\mathrm{n}_t^{\mathrm{sas}'}(s_t^m,a_t^m,s_2^m)}{\mathrm{n}_t^{\mathrm{sa}}}$$
$$\times\left[\prod_{t'=t+1}^{T-1}\frac{\mathrm{n}_{t'}^{\mathrm{sas}'}(s_{t'}^m,a_{t'}^m,s_{t+1}^m)}{\mathrm{n}_{t'}^{\mathrm{s}}(s_{t'}^m)}\right]\frac{\mathrm{n}_T^{\mathrm{sa}}(s_T^m,a_T^m)}{\mathrm{n}_T^{\mathrm{s}}(s_T^m)}r_T(s_T^m,a_T^m,\mathrm{n}_T^{\mathrm{sa}},d_T) \tag{5.15}$$

In the above result, it appears that computing the function $Q^{\mathbf{n},\boldsymbol{d}}(\cdot)$ is intractable due to the summation over $(s_{t:T}^m,a_{t:T}^m)$. Fortunately, we show that it can be computed efficiently using dynamic programming.

Theorem 5.2. *Given a sample* $\mathbf{n}$*, we can compute* $Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)$ *function by a dynamic programming as follows:*

$$Q_H^{\mathbf{n},\boldsymbol{d}}(i,j) = r_H^{\mathbf{n},\boldsymbol{d}}(i,j) \tag{5.16}$$
$$Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) = r_t^{\mathbf{n},\boldsymbol{d}}(i,j) + \sum_{i'\in S}\phi_t^{\mathbf{n},\boldsymbol{d}}(i'|i,j)V_{t+1}^{\mathbf{n},\boldsymbol{d}}(i')\forall i,j \tag{5.17}$$
$$V_t^{\mathbf{n},\boldsymbol{d}}(i') = \sum_{j'\in A}\pi_t^{\mathbf{n},\boldsymbol{d}}(j'|i')\times Q_t^{\mathbf{n},\boldsymbol{d}}(i',j'),\forall j' \tag{5.18}$$

where

$$\phi_t^{\mathbf{n},\boldsymbol{d}}(i'|i,j) = \frac{\mathrm{n}_t^{\mathrm{sas}'}(i,j,i')}{\mathrm{n}_t^{\mathrm{sa}}(i,j)};\ \pi_t^{\mathbf{n},\boldsymbol{d}}(j|i) = \frac{\mathrm{n}_t^{\mathrm{sa}}(i,j)}{\mathrm{n}_t^{\mathrm{s}}(i)} \tag{5.19}$$
$$P_1^{\mathbf{n},\boldsymbol{d}}(i) = \frac{\mathrm{n}_1^{\mathrm{s}}(i)}{M};\ r_t^{\mathbf{n},\boldsymbol{d}}(i,j) = r_t(i,j,\mathrm{n}_t^{\mathrm{sa}},d_t) \tag{5.20}$$

Proof. Given a fixed time horizon H, we prove this theorem by induction for $t = H \to 0$.

For $t = H$: As there is no future reward, $Q_H^{\mathbf{n},\boldsymbol{d}}(i,j) = r_H^{\mathbf{n},\boldsymbol{d}}(i,j)$.

Induction hypothesis: assume (5.16), (5.17), (5.18) hold for $t \le H$. We prove they

also hold for $t-1$. According to equation (5.14), we have

$$Q_{t-1}^{\mathbf{n},\boldsymbol{d}}(i,j) = r_{t-1}\big(i,j,\mathrm{n}_{t-1}^{\mathrm{sa}},d_{t-1}\big) + \sum_{T=t}^{H}\sum_{\mathrm{n}_{t-1:T}^{\mathrm{sa}},d_{t-1:T}}\sum_{s_{t-1:T}^{m},a_{t-1:T}^{m}} \frac{\mathrm{n}_{t-1}^{\mathrm{sas}'}(i,j,s_t^m)}{\mathrm{n}_{t-1}^{\mathrm{sa}}}\left[\prod_{t'=t+1}^{T-1}\frac{\mathrm{n}_{t'}^{\mathrm{sas}'}(s_{t'}^m,a_{t'}^m,s_{t+1}^m)}{\mathrm{n}_{t'}^{\mathrm{s}}(s_{t'}^m)}\right]\frac{\mathrm{n}_{T}^{\mathrm{sa}}(s_T^m,a_T^m)}{\mathrm{n}_{T}^{\mathrm{s}}(s_T^m)} r_T(s_T^m,a_T^m,\mathrm{n}_T^{\mathrm{sa}},d_T) \tag{5.21}$$

by re-arrage the sum, we have

$$\begin{aligned}
&= r_{t-1}\big(i,j,\mathrm{n}_{t-1}^{\mathrm{sa}},d_{t-1}\big) + \sum_{s_t^m,a_t^m}\frac{\mathrm{n}_{t-1}^{\mathrm{sas}'}(i,j,s_t^m)}{\mathrm{n}_{t-1}^{\mathrm{sa}}(i,j)}\frac{\mathrm{n}_{t}^{\mathrm{sa}}(s_t^m,a_t^m)}{\mathrm{n}_{t}^{\mathrm{sa}}(s_t^m)}\Bigg(r_t\big(s_t^m,a_t^m,\mathrm{n}_t^{\mathrm{sa}},d_t\big) \\
&+ \sum_{T=t+1}^{H}\sum_{\mathrm{n}_{t:T}^{\mathrm{sa}},d_{t:T}}\sum_{s_{t:T}^m,a_{t:T}^m}\frac{\mathrm{n}_{t}^{\mathrm{sas}'}(s_t^m,a_t^m,s_{t+1}^m)}{\mathrm{n}_{t}^{\mathrm{sa}}}\left[\prod_{t'=t+1}^{T-1}\frac{\mathrm{n}_{t'}^{\mathrm{sas}'}(s_{t'}^m,a_{t'}^m,s_{t+1}^m)}{\mathrm{n}_{t'}^{\mathrm{s}}(s_{t'}^m)}\right] \\
&\qquad \frac{\mathrm{n}_{T}^{\mathrm{sa}}(s_T^m,a_T^m)}{\mathrm{n}_{T}^{\mathrm{s}}(s_T^m)} r_T(s_T^m,a_T^m,\mathrm{n}_T^{\mathrm{sa}},d_T)\Bigg)
\end{aligned} \tag{5.22}$$

$$= r_{t-1}\big(i,j,\mathrm{n}_{t-1}^{\mathrm{sa}},d_{t-1}\big) + \sum_{s_t^m,a_t^m}\frac{\mathrm{n}_{t-1}^{\mathrm{sas}'}(i,j,s_t^m)}{\mathrm{n}_{t-1}^{\mathrm{sa}}(i,j)}\frac{\mathrm{n}_{t}^{\mathrm{sa}}(s_t^m,a_t^m)}{\mathrm{n}_{t}^{\mathrm{sa}}(s_t^m)} Q_t^{\mathbf{n},\boldsymbol{d}}(s_t^m,a_t^m)$$

$$= r_{t-1}\big(i,j,\mathrm{n}_{t-1}^{\mathrm{sa}},d_{t-1}\big) + \sum_{s_t^m,a_t^m}\pi_t^{\mathbf{n},\boldsymbol{d}}(a_t^m|s_t^m)\phi_t^{\mathbf{n},\boldsymbol{d}}(s_t^m|i,j)Q_t^{\mathbf{n},\boldsymbol{d}}(s_t^m,a_t^m). \tag{5.23}$$

The last expression (5.23) shows the theorem holds also for $t-1$, which concludes our induction proof. □

To summarize, we have:

Proposition 5.3. *The individual value function can be computed as:*

$$Q_t(i,j,\mathrm{n}_t^{\mathrm{sa}},d_t) = \mathbb{E}_{\mathbf{n}_{t:H},d_{t:H}}\big[Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)\big|\,\mathrm{n}_t^{\mathrm{sa}},d_t\big] \tag{5.24}$$

An example of individual value function estimated from the collective sample is given by Figure 5.1. Assume that the rewards are only collected at the planning horizon $H=2$. The red color numbers illustrate samples of the count values. Individual values at the ending period are set to be directly the immediate rewards. The values are recursively updated in a back-propagation manner based on the counts.

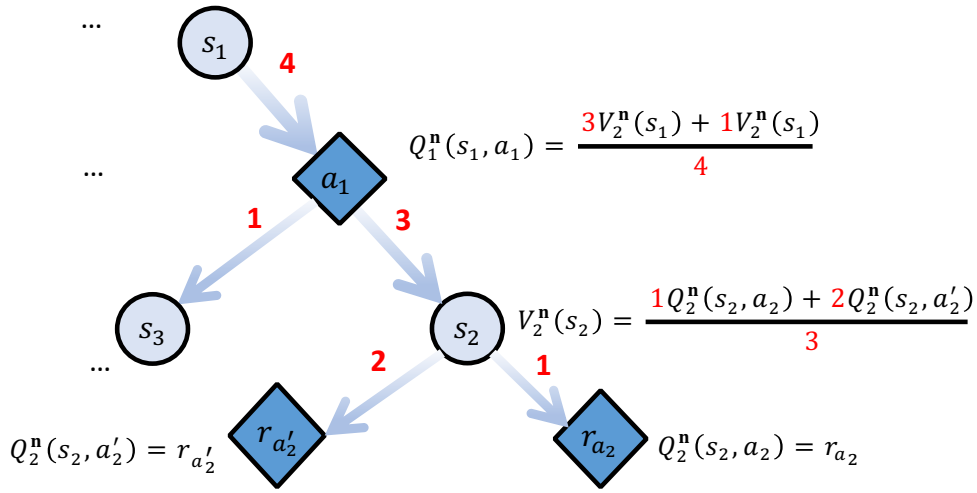


Figure 5.1: Example of individual value function estimation from collective sampling.

5.3 Policy Gradient for $\mathbb{C}$Dec-POMDPs

5.3.1 Outline

Policy gradient

Recall from Chapter 4 that for the general MDP, under a policy π, the value function $Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is defined as expected accumulative reward from time t forward given the current joint state-action to be $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, d_t \rangle$.

$$Q_t^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \mathbb{E}[\sum_{T \geq t} r_T | \boldsymbol{s}_t, \boldsymbol{a}_t, d_t; \pi] \tag{5.25}$$

The objective value is

$$V(\pi) = \mathbb{E}[\sum_{T \geq 0} r_T | b_o, b_o^g; \pi] \tag{5.26}$$

To find the policy to maximize $V(\pi)$, we consider the actor-critic method [118] to update the critic (an approximation of value function) and actor (the policy function) based on trajectory sample $(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H})$ as follows:

- **Critic update:** to refine parameters w of the value function approximation $\tilde{Q}_w$ based on the empirical return R_t:

$$R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{t'=t}^{H} r(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \tag{5.27}$$

$$w = w - \nabla_w \sum_t \Big(\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \Big)^2 \tag{5.28}$$

- **Actor update:** to refine parameters θ of the policy function π based on the sample of value function:

$$\theta = \theta + \nabla_\theta \sum_t \tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, d_t) \tag{5.29}$$

Local reward signals

To apply actor-critic method with local reward signals, we consider the factored critic function

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w\big(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t\big) \tag{5.30}$$

As shown in theorem 4.3, the *policy gradient* $\nabla_\theta V_1(\pi)$ of factored critic function can be computed as:

$$\mathbb{E}_{\mathbf{n}_{1:H} \in \Omega_{1:H}, d_{1:H}} \left[\sum_{t=1}^{H} \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) \nabla_\theta \log \pi\big(j | i, o(i, \mathrm{n}_t^s, d_t)\big) f_w(i, j, \mathrm{n}_t^s, d_t) \right]$$

In this section, we show that the individual value $Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)$ in theorem 5.2 can be used to estimate critic component $f_w\big(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t\big)$.

5.3.2 Training Action-Value Function

In our approach, after count samples $\mathbf{n}_{1:H}$ are generated to compute the policy gradient, we also need to adjust the parameters w of our critic $\tilde{Q}_w$. Notice that as per (5.30), our factored critic function $\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ depends only on the counts generated by the joint-state and action $(\mathrm{n}_t^{\mathrm{sa}}, d_t)$. Training $\tilde{Q}_w$ can be done by taking a gradient step as in (5.28) to minimize the following loss function:

$$\min_{w} \sum_{\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, d_{1:H}} \sum_{t=1}^{H} \Big(\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) - R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \Big)^2 \tag{5.31}$$

where $R_t(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t)$ is the total empirical return for time step t computed using (5.27).

However, we found that the loss in (5.31) did not work well for training the critic $\tilde{Q}_w$ for larger problems. Several count samples were required to reliably train $\tilde{Q}_w$ which adversely affects scalability for large problems with many agents. It is already known in multiagent RL that algorithms that solely rely on the *global* reward signal (e.g. R_t in our case) may require several more samples than approaches that take advantage of local reward signals [8]. Motivated by this observation, we next develop a local reward signal based strategy to train the critic f_w.

Individual Value Function: Let $\mathbf{n}_{1:H}$ be a count sample and the individual value $Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)$ as shown in theorem 5.2. Based on this value function, we next show an alternative reparameterization of the global empirical reward R_t in (5.27):

Lemma 5.1. *The empirical return R_t for the time step t given the count sample $\mathbf{n}_{1:H}$ can be re-parameterized as: $R_t = \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)$.*

Proof. We know from theorem 5.2 that the individual value function $Q_t^{\mathbf{n},\boldsymbol{d}}$ for a count sample $\mathbf{n}$ is given by the following expectation:

$$Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) = \mathbb{E}\Big[\sum_{t'=t}^{H} r_{t'}^m | s_t^m = i, a_t^m = j, \mathbf{n}_{1:H} \Big] \tag{5.32}$$

By definition, the total empirical return R_t is given by the summation of individual value function for all the agents m:

$$R_t = \sum_m Q_t^{\mathbf{n},\boldsymbol{d}}(s_t^m, a_t^m) = \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) \tag{5.33}$$

For the last equation, we have used the fact that agents which are in the same state i and and take the same action j, they have the same value function (as all the agents are identical). □

Individual Value Function Based Loss: Given lemma 5.1, we next derive an upper bound on the on the true loss (5.31) which effectively utilizes individual value functions:

$$\begin{aligned}
&\sum_t \Big(\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) - R_t \Big)^2 \\
&= \sum_t \Big(\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w(i,j,\mathrm{n}_t^{\mathrm{s}}, d_t) - \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) \Big)^2 \\
&= \sum_t \Big(\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \Big(f_w(i,j,\mathrm{n}_t^{\mathrm{s}}, d_t) - Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) \Big) \Big)^2 \qquad (5.34) \\
&\leq M \sum_{t,i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \Big(f_w(i,j,\mathrm{n}_t^{\mathrm{s}}, d_t) - Q_t^{\mathbf{n},\boldsymbol{d}}(i,j) \Big)^2 \qquad (5.35)
\end{aligned}$$

where the last relation is derived by Cauchy-Schwarz inequality. We train the critic using the modified loss function in (5.35). Empirically, we observed that for larger problems, this new loss function resulted in much faster convergence than the original loss function.

5.4 Evolutionary Game Theory

In this section, we show the relationship of our algorithm to evolutionary game theory. We consider the class of population game [99], in which policy of homogeneous (or symmetric) agents in a population evolve over time until they reach an

equilibrium or stable states. The evolution of the policy can be defined by a dynamic equation to update policy in favor of some individual value functions. We consider the two well-known dynamics, which are Gradient Ascent (GA) [111] and Replicator dynamics [122]. These evolutionary dynamics are popular methods to find equilibriums in game theory. By studying GA and Replicator dynamics for $\mathbb{C}$Dec-POMDP, later we can justify the converged solutions found by our methods in the perspective of the equilibrium notion in game theory.

5.4.1 Dynamics in Agent Population

Let us start by considering a stateless population game [99]. A population game is defined for M individuals with a distribution $\langle \pi(j) \rangle_{j \in A}$ over a set of discrete action A. Each agent in the population chooses an action j with the probability $\pi(j)$. As agents in the population are symmetric and share a homogeneous policy, we can define a pay-off function $Q^{\pi}(j)$ for an *anonymous* agent choosing action j when the population policy is π. Under the policy π and individual pay-off function Q^{π}, each agent perceives its total expected pay-off as:

$$V(\pi, Q^{\pi}) = \sum_{j} \pi(j) Q^{\pi}(j) \tag{5.36}$$

At each iteration, each agent follows an fictitious play [18] assumption that other agents follow the stationary policy π, therefore it presumes that its individual Q^{π} would be unchanged. Then, agents replace policy π with a policy π' to improve the individual expected pay-off $V(\pi', Q^{\pi})$ as

$$\sum_{j} \pi'(j) Q^{\pi}(j) > \sum_{j} \pi(j) Q^{\pi}(j) \tag{5.37}$$

In general, we can write the policy update in such apopulation game as:

$$\pi' = project\big(\pi + \alpha\Delta(\pi, Q)\big), \tag{5.38}$$

in which α is the learning rate, $\Delta(\pi, Q)$ is the change of policy depending on π, Q and $project$ is the projection of updated parameters to probability space ($\sum_j \pi'(j) = 1$).

We denote $\Delta\pi(j)$ to be component of Δ corresponding to element $\pi(j)$. The Gradient Ascent [111] update can be defined as:

$$\Delta\pi(j) = \frac{\partial V(\pi, Q)}{\partial \pi(j)} = Q(j) - \sum_{j' \in A} Q(j')\pi(j'). \tag{5.39}$$

Notice that $\sum_{j' \in A} Q(j')\pi(j')$ plays the role as a baseline (as in RL), which does not introduce any bias in the policy gradient.

On the other hand, the Replicator dynamic, as introduced by [122], is:

$$\Delta\pi(j) = \pi(j)\frac{\partial V(\pi, Q)}{\partial \pi(j)} = \pi(j)\big[Q(j) - \sum_{j' \in A} Q(j')\pi(j')\big]. \tag{5.40}$$

As noted by Kaisers et al. [50], Replicator dynamic update can be considered as a weighted (by the action probability) gradient update.

5.4.2 Stateful dynamics in population

Hennes et al. proposed an extension of Replicator dynamics to stateful environment. Without loss of generality, we consider a population game in an MDP with joint state $\boldsymbol{s}_t$ over a planning horizon H. The total expected pay-off of an agent is replaced by the value function:

$$V(\pi, Q) = \sum_t \sum_{\boldsymbol{s}_t} P^\pi(\boldsymbol{s}_t)\pi_t(j|\boldsymbol{s}_t)Q_t(j, \boldsymbol{s}_t), \tag{5.41}$$

in which $P^{\pi}(\boldsymbol{s}_t)$ is the probability of joint state $\boldsymbol{s}_t$ appearing at time t under the population policy π. The state-action value function $Q_t(j, \boldsymbol{s}_t)$ specifies the cumulative expected rewards of an anonymous agent when it takes action j under the joint state $\boldsymbol{s}_t$:

$$Q_t(j, \boldsymbol{s}_t) = r(j, \boldsymbol{s}_t, \pi) + \sum_{\boldsymbol{s}_{t+1} \in \boldsymbol{S}, j' \in A} P(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, j) \pi_{t+1}(j' | \boldsymbol{s}_{t+1}), \tag{5.42}$$

in which $P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, j)$ is the transition probability when this agent takes action j and others follow the policy π_t.

The stateful Gradient Ascent dynamics for an MDP over planning horizon H are defined as [45]:

$$\begin{aligned} &\Delta \pi_t(j|\boldsymbol{s}_t) = \frac{\partial V(\pi, Q)}{\partial \pi_t(j|\boldsymbol{s}_t)} \\ &= P^{\pi}(\boldsymbol{s}_t)[Q_t(j, \boldsymbol{s}_t) - \sum_{j'} Q_t(j', \boldsymbol{s}_t)\pi_t(j'|\boldsymbol{s}_t)], \forall j \in A, \boldsymbol{s}_t \in \boldsymbol{S}, t \in [1:H], \end{aligned} \tag{5.43}$$

in which the state-action policy $\pi_t(j|\boldsymbol{s}_t)$ denotes the probability an individual taking action j at time period t (in a planning horizon) when the system joint state to be $\boldsymbol{s}_t$. The whole policy can be represented as $\pi = (\pi_1, \ldots, \pi_H)$.

Correspondingly, the stateful Replicator dynamics for an MDP over planning horizon H are defined as [45]:

$$\begin{aligned} &\Delta \pi_t(j|\boldsymbol{s}_t) = \pi_t(j|\boldsymbol{s}_t) \frac{\partial V(\pi, Q)}{\partial \pi_t(j|\boldsymbol{s}_t)} \\ &= \pi_t(j|\boldsymbol{s}_t) P^{\pi}(\boldsymbol{s}_t)[Q_t(j, \boldsymbol{s}_t) - \sum_{j'} Q_t(j', \boldsymbol{s}_t)\pi_t(j'|\boldsymbol{s}_t)], \forall j \in A, \boldsymbol{s}_t \in \boldsymbol{S}, t \in [1:H], \end{aligned} \tag{5.44}$$

Extension to $\mathbb{C}$Dec-POMDP

Similar to the state-coupled extension in [45], we can consider the extension of Replicator dynamics in (5.44) into the partial observation setting of $\mathbb{C}$Dec-POMDPs by:

$$\begin{aligned}
&\Delta\pi_t(j|i,o_t) = \pi_t(j|i,o_t)\frac{\partial V(\pi,Q)}{\partial \pi_t(j|i,o_t)} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} \mathbb{I}\big(o_t = o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\big) P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j|i,o_t)\frac{\partial \sum_{j'}\pi_t(j'|i,o_t)Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)}{\partial \pi_t(j|i,o_t)} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} \mathbb{I}\big(o_t = o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\big) P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j|i,o_t)[Q_t(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) \\
&\qquad - \sum_{j'} Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o_t)], \forall i \in S, j \in A, o_t \in O, t \in [1:H], \quad (5.45)
\end{aligned}$$

with $Q_t(i,j,\mathrm{n}_t^{\mathrm{s}},d_t)$ to be the cumulative reward of an anonymous agent when it is in local state i, observes o_t and takes a local action j. $P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)$ is the probability that an agent is in local state i and the global state is $(\mathrm{n}_t^{\mathrm{s}},d_t)$. The baseline $\sum_{j'} Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o_t)$ is used correspondingly to the stateless baseline in (5.39).

Analogously, we can define the Gradient Ascent dynamics in $\mathbb{C}$Dec-POMDPs as:

$$\begin{aligned}
&\Delta\pi_t(j|i,o_t) = \frac{\partial V(\pi,Q)}{\partial \pi_t(j|i,o_t)} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} \mathbb{I}\big(o_t = o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\big) P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)\frac{\partial \sum_{j'}\pi_t(j'|i,o_t)Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)}{\partial \pi_t(j|i,o_t)} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} \mathbb{I}\big(o_t = o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\big) P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)[Q_t(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) \\
&\qquad - \sum_{j'} Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o_t)] \quad (5.46)
\end{aligned}$$

Parameterized policy

As in $\mathbb{C}$Dec-POMDPs, we consider policy function π to be parameterized by θ, the policy parameter gradients follow the chain rule:

$$\frac{\partial V(\pi, Q)}{\partial \theta} = \frac{\partial V(\pi, Q)}{\partial \pi} \frac{\partial \pi}{\partial \theta} \tag{5.47}$$

Based on equation (5.47), we propose the corresponding parameter dynamics for population as follows:

$$\Delta\theta = \Delta\pi \times \frac{\partial \pi}{\partial \theta} \tag{5.48}$$

Replace the Gradient Ascent dynamics in (5.46) and Replicator dynamics in (5.45) into (5.48) we have the following parameter updates at each time t:

Replicator Dynamics:

$$\begin{aligned}
\Delta\theta &= \sum_{i,j} \pi_t(j|i, o_t) \frac{\partial V(\pi, Q)}{\partial \pi_t(j|i, o_t)} \frac{\partial \pi}{\partial \theta} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}}, d_t} \sum_{i} P^{\pi}(i, \mathrm{n}_t^{\mathrm{s}}, d_t) \sum_{j \in A} \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) [Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \\
&\qquad - \sum_{j'} Q_t(i, j', \mathrm{n}_t^{\mathrm{s}}, d_t) \pi_t(j'|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))] \frac{\partial \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))}{\partial \theta}
\end{aligned} \tag{5.49}$$

Gradient Ascent Dynamics:

$$\begin{aligned}
\Delta\theta &= \sum_{i,j} \frac{\partial V(\pi, Q)}{\partial \pi_t(j|i, o_t)} \frac{\partial \pi}{\partial \theta} \\
&= \sum_{\mathrm{n}_t^{\mathrm{s}}, d_t} P^{\pi}(i, \mathrm{n}_t^{\mathrm{s}}, d_t) \sum_{j \in A} [Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \\
&\qquad - \sum_{j'} Q_t(i, j', \mathrm{n}_t^{\mathrm{s}}, d_t) \pi_t(j'|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))] \frac{\partial \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))}{\partial \theta}
\end{aligned} \tag{5.50}$$

We notice that the joint individual probability $P^{\pi}(i, \mathrm{n}_t^{\mathrm{s}}, d_t)$ can be computed us-

ing corollary 5.2 as

$$\begin{aligned} P^{\pi}(i, \mathrm{n}_t^{\mathrm{s}}, d_t) &= \sum_{j} \sum_{\mathbf{n}'_{1:t}, d'_{1:t}} P\big(i, j, \mathbf{n}'_{1:t}, d'_{1:t}\big) \mathbb{I}(\mathrm{n}_t'^{\mathrm{s}} = \mathrm{n}_t^{\mathrm{s}}, d'_t = d_t) \\ &= \sum_{j} \sum_{\mathbf{n}'_{1:t}, d'_{1:t}} P(\mathbf{n}'_{1:t}, d'_{1:t}) \frac{\mathrm{n}_t'^{\mathrm{sa}}(i, j)}{M} \mathbb{I}(\mathrm{n}_t'^{\mathrm{s}} = \mathrm{n}_t^{\mathrm{s}}, d'_t = d_t) \\ &= \sum_{\mathbf{n}'_{1:t}, d'_{1:t}} P(\mathbf{n}'_{1:t}, d'_{1:t}) \frac{\mathrm{n}_t'^{\mathrm{s}}(i)}{M} \mathbb{I}(\mathrm{n}_t'^{\mathrm{s}} = \mathrm{n}_t^{\mathrm{s}}, d'_t = d_t) \end{aligned} \tag{5.51}$$

Analogously, we have

$$P^{\pi}(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) = \sum_{\mathbf{n}'_{1:t}, d'_{1:t}} P(\mathbf{n}'_{1:t}, d'_{1:t}) \frac{\mathrm{n}_t'^{\mathrm{sa}}(i, j)}{M} \mathbb{I}(\mathrm{n}_t'^{\mathrm{s}} = \mathrm{n}_t^{\mathrm{s}}, d'_t = d_t) \tag{5.52}$$

Equations (5.51) and (5.52) give us a way to estimate $P^{\pi}(i, \mathrm{n}_t^{\mathrm{s}}, d_t)$ and $P^{\pi}(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ by sampling the counts instead of the individual trajectory.

To compute $Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$, we apply proposition 5.3 to have

$$\begin{aligned} Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) &= \sum_{\mathrm{n}_t^{\mathrm{sa}}} P(\mathrm{n}_t^{\mathrm{sa}} \,|\, \mathrm{n}_t^{\mathrm{s}}, d_t) Q_t(i, j, \mathrm{n}_t^{\mathrm{sa}}, d_t) \\ &= \sum_{\mathrm{n}_t^{\mathrm{sa}}} P(\mathrm{n}_t^{\mathrm{sa}} \,|\, \mathrm{n}_t^{\mathrm{s}}, d_t) \mathbb{E}_{\mathbf{n}_{t:H}, d_{t:H}} \left[Q_t^{\mathbf{n}, \boldsymbol{d}}(i, j) \middle| \, \mathrm{n}_t^{\mathrm{sa}}, d_t \right] \\ &= \mathbb{E}_{\mathbf{n}_{t:H}, d_{t:H}} \left[Q_t^{\mathbf{n}, \boldsymbol{d}}(i, j) \middle| \, \mathrm{n}_t^{\mathrm{s}}, d_t \right] \end{aligned} \tag{5.53}$$

Equation (5.53) gives us a way to estimate $Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ by sampling $Q_t^{\mathbf{n}, \boldsymbol{d}}(i, j)$ from the conditional collective probability $P(\mathbf{n}_{t:H}, d_{t:H} | \, \mathrm{n}_t^{\mathrm{s}}, d_t)$. We can approximate $Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ by the individual critic critic $f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ trained by the loss function (5.35).

Applying (5.51) and $Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \approx f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ into (5.49) and (5.50), we can estimate the parameter updates of Replicator Dynamics and Gradient Ascent Dynamics as follows:

Gradient Ascent Dynamics $\Delta\theta$:

$$= \mathbb{E}_{\mathbf{n}_t \in \Omega_t, d_t}\Big[\sum_{i,j} \frac{\mathrm{n}_t^{\mathrm{s}}(i)}{M}\Big[f_w(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) - \sum_{j'} f_w(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))\Big]\frac{\partial \pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))}{\partial\theta}\Big] \tag{5.54}$$

Replicator Dynamics $\Delta\theta$:

$$= \mathbb{E}_{\mathbf{n}_t \in \Omega_t, d_t}\Big[\sum_{i,j} \frac{\mathrm{n}_t^{\mathrm{s}}(i)}{M}\pi\big(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\big)\Big[f_w(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) - \sum_{j'} f_w(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))\Big] \times \frac{\partial \pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))}{\partial\theta}\Big] \tag{5.55}$$

Equivalence between Gradient Ascent Dynamics and Factored Actor Update

With notice of log-trick

$$\frac{\partial \sum_{j'} \pi_t(j'|i,o_t)Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)}{\partial\theta} = \pi_t(j|i,o_t)Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\frac{\partial \log \pi_t(j'|i,o_t)}{\partial\theta},$$

we can also re-write equation (5.50) of the Gradient Ascent dynamics as:

$$\Delta\theta = \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))\sum_{j\in A}[Q_t(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) - \sum_{j'} Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j'|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))]\frac{\partial \log \pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))}{\partial\theta}$$

using the fact $P^{\pi}(i,\mathrm{n}_t^{\mathrm{s}},d_t)\pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t)) = P^{\pi}(i,j,\mathrm{n}_t^{\mathrm{s}},d_t)$, we have

$$= \sum_{\mathrm{n}_t^{\mathrm{s}},d_t} P^{\pi}(i,j,\mathrm{n}_t^{\mathrm{s}},d_t)\sum_{j\in A}[Q_t(i,j,\mathrm{n}_t^{\mathrm{s}},d_t) - \sum_{j'} Q_t(i,j',\mathrm{n}_t^{\mathrm{s}},d_t)P^{\pi}(i,j,\mathrm{n}_t^{\mathrm{s}},d_t)]\frac{\partial \log \pi_t(j|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t))}{\partial\theta} \tag{5.56}$$

Applying (5.52) by (5.52) and $Q_t(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \approx f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)$ into (5.56), we can estimate the parameter updates of Gradient Ascent Dynamics as follows:

$$= \mathbb{E}_{\mathbf{n}_t \in \Omega_t, d_t}\Big[\sum_{i,j} \frac{\mathrm{n}_t^{\mathrm{sa}}(i,j)}{M}\Big[f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) - \sum_{j'} f_w(i, j', \mathrm{n}_t^{\mathrm{s}}, d_t)\pi_t(j'|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))\Big]\frac{\partial \log \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))}{\partial \theta}\Big] \tag{5.57}$$

This Gradient Ascent Dynamics formula is equivalent to factored actor update in equation (4.47).

5.5 Algorithms

Algorithm 3: Actor-Critic RL for $\mathbb{C}$Dec-POMDPs

1 Initialize network parameter θ for actor π and and w for critic f_w
2 $\alpha \leftarrow$ actor learning rate
3 $\beta \leftarrow$ critic learning rate
4 **repeat**
5 Sample count vectors $\mathbf{n}_{1:H} \sim P(\mathbf{n}_{1:H}; \pi)$
6 Update critic as:
7 C $: w = w - \beta \nabla_w \Big[\sum_t \Big(\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) - \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)\Big)^2\Big]$
8 fC $: w = w - \beta \nabla_w \Big[\sum_{t,i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j)\Big(f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) - Q_t^{\mathbf{n},\boldsymbol{d}}(i,j)\Big)^2\Big]$
9 Update actor as:
10 A $: \theta = \theta + \alpha \nabla_\theta \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \log \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big]\Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)\Big]$
11 fA $: \theta = \theta + \alpha \nabla_\theta \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \log \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)\Big]$
12 gA $: \theta = \theta + \alpha \nabla_\theta \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{s}}(i) \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)\Big]$
13 rA $: \theta = \theta + \alpha \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{s}}(i) \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \nabla_\theta \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)\Big]$
14 **until** ***convergence***
15 **return** θ, w

Algorithm 3 shows the outline of our AC approach for $\mathbb{C}$Dec-POMDPs. Lines 7

and 8 show two different options to train the critic. Line 7 represents critic update based on local value functions, also referred to as factored critic update (fC). Line 8 shows update based on global reward or global critic update (C). Line 10 shows the policy gradient computed using theorem 4.1 (fA). Line 11 shows how the gradient is computed by directly using f_w from eq. (4.30) in eq. 4.43.

In addition, we consider the gradient descent dynamics update (gA in line 12) and Replicator dynamics update (rA in line 13). Both gA and rA policy updates use the critic estimation fC.

5.6 Experiments

This section compares the performance of different actor critic algorithms with individual rewards in a taxi fleet management domain and grid navigation domain. We consider *closed loop* policy as a policy where action selection depends on the current local state of the agent as well as the counts and global component values (the demand counts in taxi domain) at its local state and its adjacent states. In our domains, the local state is the current location of agent, i.e. in one of a zone in an urban area or one of the node in a grid network. The adjacency relation between locations is defined by the domain map.

To make comparisons between average-flow estimation and collective sampling estimation, we consider *open loop policy* where action selection only depends the agent's local state. We use the Soft-Max based flow update (SMFU) algorithm [133] and an average-flow based policy gradient ("fA+avg-flow") to optimize open loop policy. In "fA+avg-flow" algorithm, the individual value is estimated by average flows and the policy gradient is computed by using this individual value estimation. To benchmark against average-flow methods, we design an factor-actor-factor-critic algorithm to optimize the open loop policy, which is called $\mathrm{fAfC}-0$ in our experiment.

5.6.1 Taxi Supply-Demand Matching

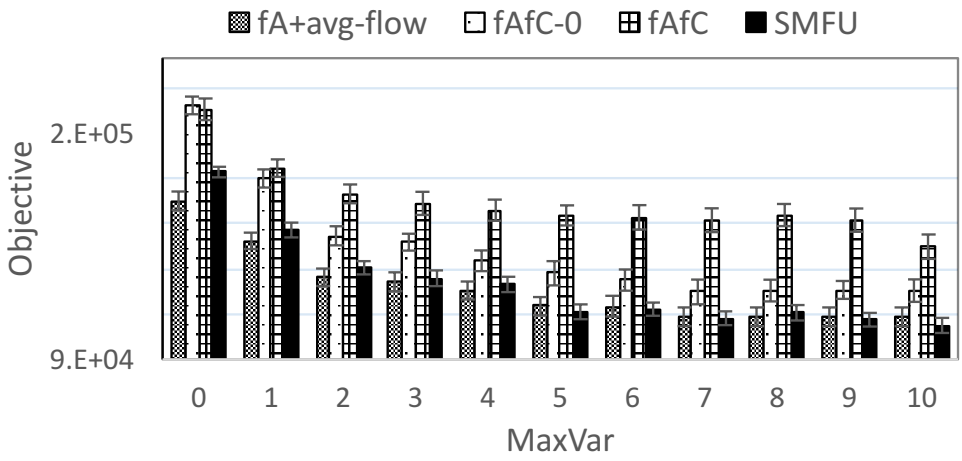
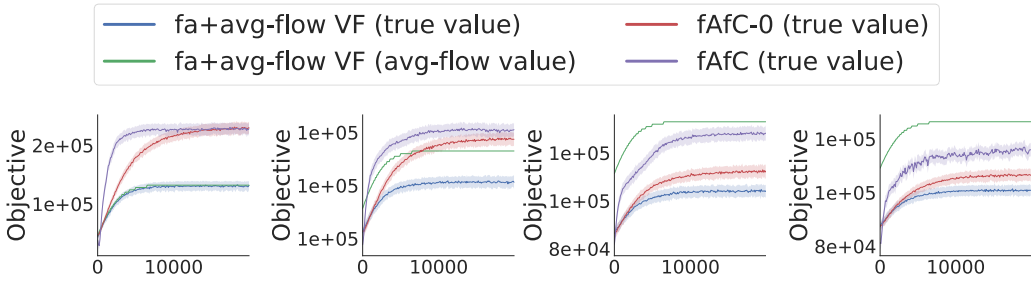
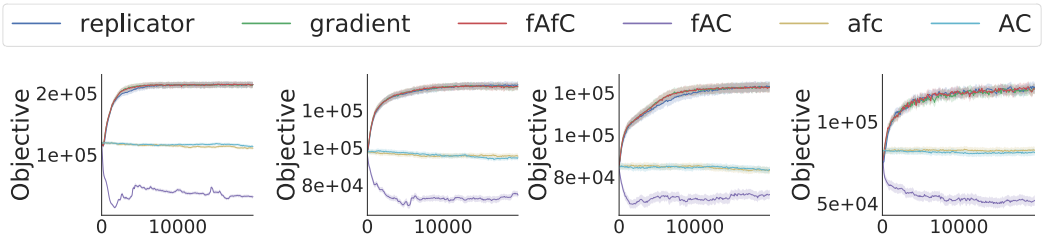


Figure 5.2: Solution quality with varying MaxVar in taxi domain



(a) $MaxVar = 0$ (b) $MaxVar = 1$ (c) $MaxVar = 5$ (d) $MaxVar = 10$

Figure 5.3: Convergence of average-flow based policy gradient and fAfC optimizing static policy on taxi domain.



(a) $MaxVar = 0$ (b) $MaxVar = 1$ (c) $MaxVar = 5$ (d) $MaxVar = 10$

Figure 5.4: Convergence of different actor-critic variants on the taxi problem.

We test our approach on this real-world domain described in section (2.1.1), and introduced in [133]. In this problem, the goal is to compute taxi policies for optimizing the total revenue of the fleet. The data contains GPS traces of taxi movement in a large Asian city over 1 year. We use the observed demand information extracted from this dataset. On an

average, there are around 8000 taxis per day (data is not exhaustive over all taxi operators). The city is divided into 81 zones and the plan horizon is 48 half hour intervals over 24 hours. For details about the environment dynamics, we refer to the appendix A.1.

Previous work only considers a fixed expected taxi demand in each city zone. To make the problem more realistic, we address stochastic taxi demand. While sampling demand, we multiply the given expected demand in a zone z with $v_z \sim \hat{\mathcal{N}}(1, \sigma_z)$, where $\hat{\mathcal{N}}$ is a truncated normal distribution between $[0, 2]$. We generate several problem settings by sampling the variance σ_z uniformly from $[0, \mathrm{MaxVar}]$ and varying MaxVar from 1 to 10. Intuitively, with higher value of σ_z, multiplier v_z tends to follow a uniform distribution over $[0, 2]$; with lower value of σ_z, v_z is close to constant (≈ 1). Figure 5.2 shows the solution quality of different approaches for varying MaxVar. We observe that collective sampling approaches fAfC and $\mathrm{fAfC}-0$ outperform average-flow approaches (SMFU and fA+avg-flow). Notably, when the MaxVar parameter increases, it increases the stochasticity in the problem. With increasing stochasticity, closed loop policy optimized by fAfC is significantly better than open loop policy optimized by $\mathrm{fAfC}-0$ and average-flow methods. This highlights the benefit of using collective sampling to optimize closed loop policy which average-flow methods can not optimize.

Furthermore, we investigate the convergence of average-flow based method $\mathrm{fA}+$avg-flow and collective flow methods in figures 5.3. The "true value" of the objective function is estimated by collective sampling of the counts and the rewards. Meanwhile, the "avg-flow value" is the estimation of objective value using the average-flow method. We observe that when $MaxVar = 0$, the transition and reward functions are in linear form, therefore, the 'true value" and "avg-flow value" of $\mathrm{fA}+$avg-flow are almost identical. When the $MaxVar$ increases, the increased stochasticity widens the gap between these 2 values. This implies the average-flow estimation is highly inaccurate and not suitable with stochastic environment.

Figures 5.4 show the quality Vs. iterations for different variations of our actor critic approach—fAfC, AC, AfC, fAC, rA, gA to optimize closed loop policy. These figures clearly show that using factored actor and the factored critic update in fAfC, rA and gA is the most reliable strategy over all the other variations and for all the observation models.

Variations such as AC and fAC were not able to converge at all despite having exactly the same parameters as fAfC. These results validate different strategies that we have developed in our work to make vanilla AC converge faster for large problems. The convergences of fAfC, rA and gA are almost identical, which shows the relation of our fAfC solution to the equilibrium in population game.

5.6.2 Robot Grid Navigation

We also tested on a synthetic benchmark. The goal is for a population of robots ($= 20$) to move from a set of initial locations to a goal state in a 5x5 grid. If there is congestion on an edge, then each agent attempting to cross the edge has higher chance of action failure. Similarly, agents also receive a negative reward if there is edge congestion. On successfully reaching the goal state, agents receive a positive reward and transition back to one of the initial state. We set the horizon to 100 steps. More details of the domain are provided in the appendix A.2.

In the robot grid navigation domain, we compare performance of different algorithms with varying the number of agents from 1 to 20. Figure 5.5 shows the solution quality comparisons among different approaches. Overall, when the congestion increases with the number of agents, the closed loop policy optimized by fAfC becomes significantly better than the open loop policy. Among open loop policy optimizers, we observe collective sampling method (fAfC -0) is better than average-flow methods. Figures 5.6 compare convergence of average-flow method and collective sampling methods. We observe that when the population size is small $N = 1$, average-flow estimation is close to the true value, which explains the high quality of average-flow methods. When the number of agents increase, the stochasticity becomes higher, as a result, average-flow estimation is no longer accurate. As the average-flow objective estimation becomes far from the "true value", fA $+$avg-flow method converges to poor solutions.

We show the convergence of different variations of our actor critic approach—fAfC, AC, AfC, fAC, rA, gA to optimize closed loop policy in figures 5.7. Similar to the observation in taxi domain,fAfC, rA and gA behave similarly and produce the best solution quality.

Among other variants, fAC has comparable solution with fAfC, rA and gA, however it becomes worse when the number of agents increases.

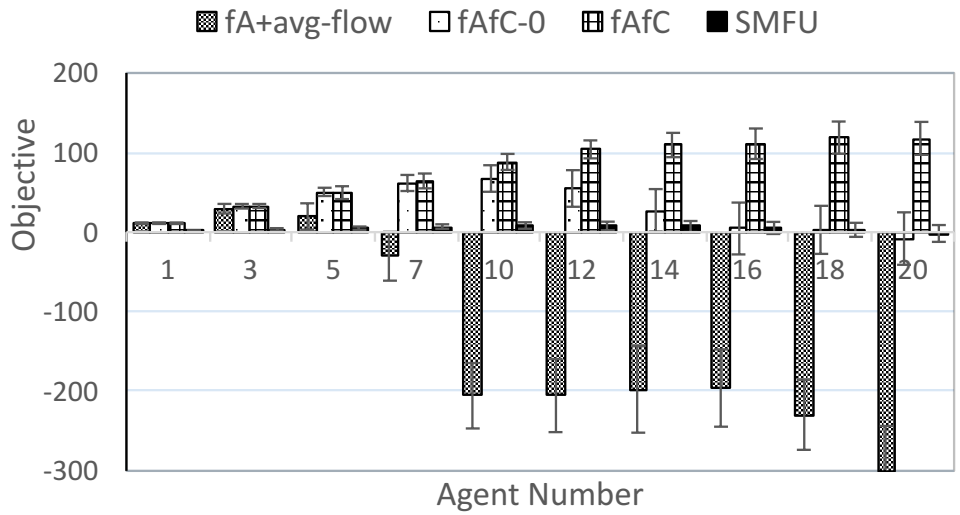
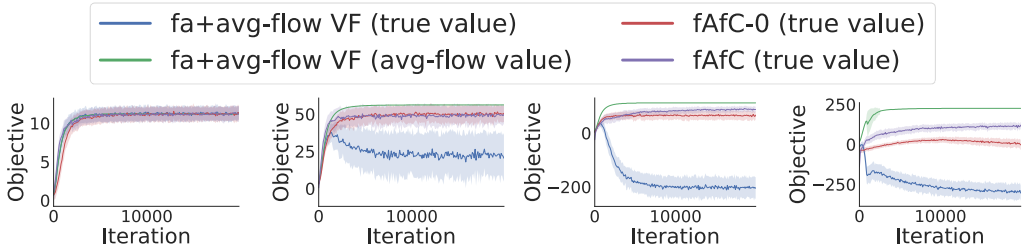
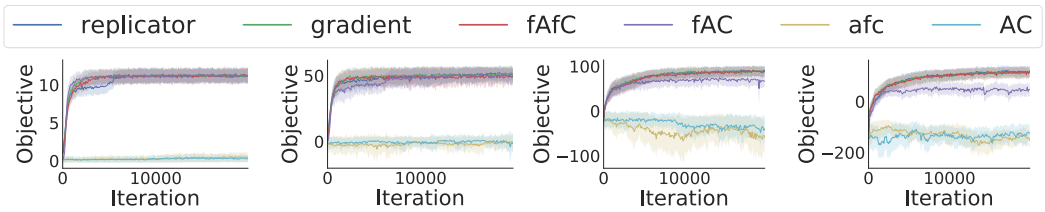


Figure 5.5: Solution quality with varying population size in grid domain



(a) $N = 1$ (b) $N = 5$ (c) $N = 10$ (d) $N = 20$

Figure 5.6: Convergence of average-flow based policy gradient and fAfC on the grid navigation problem.



(a) $N = 1$ (b) $N = 5$ (c) $N = 10$ (d) $N = 20$

Figure 5.7: Convergence of different actor-critic variants on the grid navigation problem.

5.7 Related Works

It has been shown in both model-based planning and model-free learning that value decomposition facilitates multi-agent algorithms. One of the earliest works in decomposing value function in MDPs planning is by Schneider et al. [100] who showed that the decomposing value function can reduce the complexity of computing joint policy. Later, Guestrin et al. [38] showed that the optimal joint centralized policy in MDPs with factored transition and reward functions can be efficiently computed by using the factored value function in approximate dynamic programs. Using a similar idea for model-based factored value function in [38], Kok and Vlassis [53] proposed a model-free counterpart where value function components are learnt by local rewards. For Dec-(PO)MDPs with independent-transition, Kumar et al. [58] proposed individual value functions to be computed by a factored dynamic program based on a sparse interaction structure. The main problem with model-based value function decomposition is that they assume the sparse interaction between agents, meanwhile in our $\mathbb{C}$Dec-POMDP domains an agent can interact with all other agents along its trajectory. Recently, in parallel to our work, Sunehag et al. [115] proposed to approximate the global value function by a sum of local value functions one for each agent. Agent decomposition in [115] requires maintaining value function for every agent and the training of value function uses global rewards, which is not effective in $\mathbb{C}$Dec-POMDPs with large number of agents.

In large multi-agent systems, computing equilibrium is sometimes easier than global optimum, hence there is a large body of literature in multi-agent planning focusing on finding equilibrium solutions. When studying network distributed POMDPs modeling interaction of sensor agents in the network, Nair et al. [74] proposed the best response algorithm to optimize an individual policy while fixing other agent policies. The best response algorithm in [74] is equivalent to local search and able to converge equilibrium with a high-quality. However, best response method by Nair et al. [74] is a synchronous algorithm and produce heterogeneous policies, hence it is difficult to scale-up to large population system in $\mathbb{C}$Dec-POMDP. Fictitious play [60] is an asynchronous version of best response. In fictitious play, instead of each agent taking turn to update its values and policy, *every* agent would

compute value at the same time. Fictitious play in reinforcement learning context is used in independent learner algorithms [121, 23] in which each agent maintains an independent value function based on its observed local rewards and updates its policy based on its value function. Independent learner is shown to be effective in many domains, e.g. traffic light controls [142, 59], and multi-agent pursuit-evader [40]. Analogously, the main issue with traditional methods is that maintaining value function for every agent is not tractable in large populations. Our proposed algorithms address this by estimating a single individual value function shared by all agents and the shared individual value function is centrally updated using the counts.

The scalability of our individual value function approach is attributed to marginalizing agent state-action trajectories into the sufficient-statistics counts. This idea is also explored in different research problems in the literature. Robbel et al. [97] extended Guestrin et al. [38]'s idea to large graphical model planning by considering the count-based value functions for nodes in the graph. The complexity of the algorithm in [97], however, is exponential to the tree-width of the problem. Computing individual value function as an expectation over the counts is also studied in action graph game [48]. In action graph game studied in [48], all agents have the same mixed strategy to choose their actions and the congestion based individual reward of an agent only depends on the number of other agents choosing the same action with it. To exploit the homogeneity, the individual value function can be computed by the counts. Upon individual value function computation, fictitious play method or Replicator dynamic can be used to compute an equilibrium [141]. Although the action graph game shares some similarity with our methods, it can only solve a special class of single time period problem with open loop policy.

Our work is related to congestion games [136, 98] where the equilibrium of path choosing agents is computed with the flow (or number) of agents in each link (or road). A conventional congestion game is defined as a one-shot deterministic planning problem where agents deterministically choose their paths as a sequence of links (or roads) to travel from origin to destination and there is a flow-based delay penalty in each link imposed on agents crossing that link. The routing agents in a congestion game is usually assumed to be homogeneous [98]. Equilibrium is obtained when the delay penalty of any path B is not smaller

than a path A's if we change a flow unit (corresponding to a change in the path of one agent) from path A to path B . In other words, any agent in path A is not better off by switching to a path B. As the equilibrium is determined by the flow balance, the identities of agents can be ignored. Varakantham et al. [133] extended the congestion game into sequential setting with multi time steps by estimating an average-flow of agents in each time step. For sequential multi-agent domains, Varakantham et al. [133], Ahmed et al. [3] proposed to compute the individual value function by the approximate average-flow of agents in the system. As shown in the experiment with SMFU, the average-flow approaches can only find open-loop policy which is insufficient for agents to well behave in dynamic environment.

5.8 Summary

In this chapter, we have shown that the individual value function in $\mathbb{C}$Dec-POMDPs can be estimated from samples of the counts. An efficient count-based computation of the individual value function was proposed by exploiting exchangeability properties of agents in $\mathbb{C}$Dec-POMDP. Using the individual value function and applying fictitious play principle into $\mathbb{C}$Dec-POMDPs, we derived a fictitious play based policy gradient method to optimize individual policy. In addition, we justified our fictitious play based algorithms by showing that optimizing the individual value function is equivalent to optimizing a lower bound of global value function, which theoretically explains our high quality solutions in experiment.

Chapter 6

Reinforcement Learning with Global Reward Signals

In Chapter 4, we have shown that distinguishing the values of different agents through credit-assignment is vital to learn and optimize individual policy. The credit-assignment can be considered as the process of decomposing the joint feedback into individual feedbacks. In Chapter 5, we show that if the global reward is the sum of local rewards, we can consider the decomposable value function as a sum of individual value functions trained by local reward signals. Training individual policy with a decomposable value function can converge to an equilibrium with high global quality. However, the decomposability of values is not applicable to many real-world domains where the reward function is non-decomposable. An example is in a patrolling game, when an incident is attended late, a penalty is given to the whole team instead of single agent. Even when the reward is decomposable, the transition of agents' states are interdependent, consequently the value functions of agents are not separable. In such cases, a decomposable value function is insufficient to capture the interdependency between agents. In this chapter, we would address this problem by studying multi-agent reinforcement learning algorithms using non-decomposable critics.

To develop collective actor-critic RL approaches using *non-decomposable* critics, we show in this chapter how to i) address the problem of multiagent credit assignment, and ii) compute low variance policy gradients that result in faster convergence and better solutions

than previous methods. In particular, we propose a method to estimate policy gradients by an expectation of critic value. In addition, we extend the previous technique of counterfactual based credit assignment to the collective setting. Empirically, when comparing with previous methods, our new approaches provide significantly better solutions in the presence of global rewards on a real world police re-allocation problem, a taxi fleet optimization problem and a grid patrolling synthetic benchmark. We also show that our approaches are competitive even with a centralized online planning approach.

6.1 Collective Decentralized POMDP Model

We recall the $\mathbb{C}$Dec-POMDP model with transition as defined in Chapter 2

- Finite planning horizon H.
- M denotes the total number of agents.
- An agent m can be in one of the states $i \in S$. The joint state space is $\times_{m=1}^{M} S$.
- An agent m can take an action $j \in A$. The joint action space is $\times_{m=1}^{M} A$.
- Let $(s_{1:H}, a_{1:H})^m = (s_1^m, a_1^m, s_2^m \ldots, s_H^m, a_H^m)$ denote the complete state-action trajectory of an agent m. The state and action of agent m at time t are denoted using random variables s_t^m, a_t^m.
- The environment is partially observable wherein agents can have different observations based on the collective influence of other agents. An agent observes its local state s_t^m. In addition, it also observes o_t^m at time t based on its local state s_t^m and the count table $\mathrm{n}_t^{\mathrm{s}}$. E.g., an agent m in state i at time t can observe the count of other agents also in state i (=$\mathrm{n}_t^{\mathrm{s}}(i)$) or other agents in some neighborhood of the state i (=$\{\mathrm{n}_t^{\mathrm{s}}(j)\ \forall j \in \mathrm{Nb}(i)\}$).
- The transition function is $\phi_t\big(s_{t+1}^m = i' | s_t^m = i, a_t^m = j, \mathrm{n}_t^{\mathrm{s}}\big)$. The transition function is the same for all the agents. Note that it is affected by $\mathrm{n}_t^{\mathrm{s}}$, which depends on the collective behavior of the agent population.
- Each agent m has a non-stationary policy $\pi_t^m(j|i, o_t^m(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ denoting the probability of agent m to take action j given its observation $(i, o_t^m(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ at time t. We denote the policy an agent m to be $\pi^m = (\pi_1^m, \ldots, \pi_H^m)$.

- The system receives a **global reward** $r_t = r(\mathrm{n}_t^{\mathrm{sa}}, d_t)$ dependent on the joint count actions $\mathrm{n}_t^{\mathrm{sa}}$.
- Initial state distribution, $b_o = (P(i) \forall i \in S)$, is same for all agents.

Global rewards: The key difference from previous works in Chapter 5 is that in our model we have a global reward signal r_g that is not decomposable among individual agents, which is crucial to model real world applications. Consider a real world multiagent patrolling problem in Figure 6.1. A set of homogeneous police patrol cars (or agents) are stationed in predefined geographic regions to respond to incidents that may arise over a shift (say 7AM to 7PM). When an incident comes, the central command unit dispatches the closet patrol car to the incident location. The dispatched car becomes unavailable for some amount of time (including travel and incident service time). To cover for the engaged car, other available patrol cars from nearby zones may need to reallocate themselves so that no zones are left vulnerable. The reward in this system depends on the response time to incidents (e.g., threshold to attend to urgent incidents is 10 min, non-urgent in 20 min). The goal is to compute a reallocation policy for agents to minimize the number of unsatisfied incidents where the response time was more than the specified threshold. To model this objective, we award penalty -10 whenever the response time requirement of an incident is not met and 0 otherwise. In this domain, the delay penalty is non-decomposable among patrol cars. It is not reasonable to attribute a penalty in an incident to its assigned agent because the delay is due to the intrinsic system-wide supply-demand mismatch. Furthermore, individual agent penalties may even discourage agents to go to nearby critical sectors, which is undesirable (we observed it empirically). Indeed, in this domain, all rewards are global, therefore, previous approaches that require local rewards for agents are not applicable. This is precisely the gap our work targets, and it significantly increases the applicability of multiagent decision making to real world applications. Previous approaches in Chapter 5 specifically exploit the additive decomposition of the reward signal among agents to perform multiagent credit assignment (that is, to com-

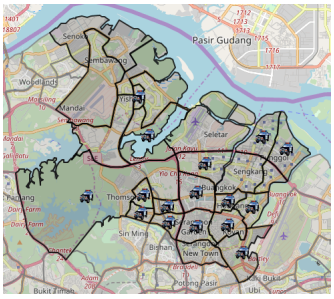


Figure 6.1: Solid black lines define 24 patrolling zones of a city district

pute the relative contributions of different agents to the total reward). In the presence of global rewards, techniques like fAfC for doing credit assignment in Chapter 5 are no longer theoretically justified and as shown empirically, perform poorly. This is precisely the gap our work targets as we develop two new techniques to perform effective multiagent RL.

6.2 Mean Collective Actor Critc

We follow an actor-critic approach for optimizing the $\mathbb{C}$Dec-POMDP policy π [55]. Recall from Chapter 2, we have the value function for a $\mathbb{C}$Dec-POMDP model as follows:

$$Q_t^{\pi}(\mathrm{n}_t^{\mathrm{sa}}, d_t) = r_t(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \sum_{\mathrm{n}_{t+1}^{\mathrm{s}}, \mathrm{n}_{t+1}^{\mathrm{sa}} \in \Omega_{t+1}, d_{t+1}} P(\mathrm{n}_{t+1}^{\mathrm{s}}, d_{t+1} | \, \mathrm{n}_t^{\mathrm{sa}}, d_t) P(\mathrm{n}_{t+1}^{\mathrm{sa}} \mid \mathrm{n}_{t+1}^{\mathrm{s}}, d_{t+1}; \pi) Q_{t+1}^{\pi}(\mathrm{n}_{t+1}^{\mathrm{sa}}, d_{t+1})$$

where Ω_{t+1} is the subset of consistency constraints (2.7) linking counts for time t and $t+1$. Notice that the action-value function is defined over count tables directly and does not require sampling individual agent trajectories. We define $P(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, d_t; \pi)$ as the collective distribution of the action counts given the action probabilities π and state counts:

$$= \prod_{i \in S} \Big[\frac{\mathrm{n}_t^{\mathrm{s}}(i)!}{\prod_{j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)!} \prod_{j \in A} \pi\big(j | i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \tag{6.1}$$

Notice that the above formula is essentially a multinomial distribution—for each state i, we perform $\mathrm{n}_t^{\mathrm{s}}(i)$ trials independently. Each trial's outcome is one of the action $j \in A$ with the probability of falling in category j given by $\pi(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$. We can similarly define another multinomial distribution for $P^{\pi}(\mathrm{n}_{t+1}^{\mathrm{s}} \mid \mathrm{n}_t^{\mathrm{sa}}, d_t)$.

We can estimate $Q_t^{\pi}(\mathrm{n}_t^{\mathrm{sa}}, d_t)$ using empirical returns, but it has high variance. To address this, in actor-critic approaches, we use a function approximation Q_w for Q^{π} and learn its parameters w during the training process. Next we show how to design the critic Q_w for our setting.

6.2.1 Critic Design For Collective Policy Gradient With Global Rewards

The key requirements for designing the critic structure are the following. First, the critic should be *trainable with global rewards* which is non-decomposable among agents. Second, the critic design should address the problem of *multiagent credit assignment* [30, 115, 26]. One of the key challenges for effective multi-agent reinforcement learning is to assign "credit" value to each individual agent's actions so that we can know which action of which agent is more preferable in the policy update. The credit value should appropriately reflect the contribution of an agent's action towards the global reward. Finally, we also require the policy gradient computed using the critic to have *low variance* for effective training. We next detail the design of such a critic for our collective setting that addresses such requirements. We start by showing the policy gradient expression for $\mathbb{C}$Dec-POMDPs.

Proposition 6.1. *For $\mathbb{C}$Dec-POMDPs, the policy gradient is given as:*

$$\begin{aligned}\nabla_\theta V(\pi) &= \nabla_\theta \sum_{t=1}^{H} \mathbb{E}_{\mathbf{n}_{1:H}\in\Omega_{1:H},d_{1:H}}\Big[Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t)\Big] \\ &= \sum_{t=1}^{H} \mathbb{E}_{\mathrm{n}_t^{\mathrm{s}}}\Big[\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}\sim P^\pi(\mathrm{n}_t^{\mathrm{sa}}\,|\,\mathrm{n}_t^{\mathrm{s}})}\Big[Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t)\Big]\Big] \end{aligned} \tag{6.2}$$

Proof. Recall from theorem 4.2 that the policy gradient in $\mathbb{C}$Dec-POMDPs can be computed as:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{H} \mathbb{E}_{d_t,\mathrm{n}_t^{\mathrm{sa}}\,|b_o,b_o^g,\pi}\left[Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t)\left(\sum_{i\in S, j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)\nabla_\theta \log \pi_t(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}}))\right)\right] \tag{6.3}$$

We notice that inner summation in (6.3) can rewritten as:

$$\begin{aligned}
&\sum_{i\in S, j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)\nabla_\theta \log \pi_t(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})) \\
&= \nabla_\theta \log \left(\prod_{i\in S} \left[\frac{\mathrm{n}_t^{\mathrm{s}}(i)!}{\prod_{j\in A} \mathrm{n}_t^{\mathrm{sa}}(i,j)!} \prod_{j\in A} \pi\big(j|i, o(i, \mathrm{n}_t^{\mathrm{s}})\big)^{\mathrm{n}_t^{\mathrm{sa}}(i,j)} \right] \right) \\
&= \nabla_\theta \log P^\pi(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}) \qquad (6.4)
\end{aligned}$$

where we used the fact that the gradient of the factorial terms is zero. Using this above result:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{H} \mathbb{E}_{d_t, \mathrm{n}_t^{\mathrm{s}} | b_o, b_o^g, \pi} \left[\sum_{\mathrm{n}^{\mathrm{sa}}} Q_t^\pi(d_t, \mathrm{n}^{\mathrm{sa}}) P^\pi(\mathrm{n}^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}) \nabla_\theta \log P^\pi(\mathrm{n}^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}) \right] \qquad (6.5)$$

$$= \mathbb{E}_{d_t, \mathrm{n}_t^{\mathrm{s}} | b_o, b_o^g, \pi} \left[\sum_{\mathrm{n}^{\mathrm{sa}}} Q_t^\pi(d_t, \mathrm{n}^{\mathrm{sa}}) \nabla_\theta P^\pi(\mathrm{n}^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}) \right] \qquad (6.6)$$

Taking gradient ∇_θ outside of the summation proves the proposition. □

In the next section, we address two concerns with the critic design—low variance and multiagent credit assignment. Recent work has shown that computing the inner expectation $\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}} \left[Q_t^\pi(\mathrm{n}_t^{\mathrm{sa}}, d_t) \right]$ in closed form (rather than using sampling) results in lower variance of the gradient [22, 104, 6]. Furthermore, as shown by Asadi et al., a baseline is not needed in the expected case to reduce the variance of the gradient. Therefore, we would also show computation of expected (or mean) collective policy gradient.

Factorization form

A major challenge in multi-agent RL is to assign *"credit"* values to each individual agent's actions so that we know which action is more preferable in the policy update. One solution studied in [115, 77] for this problem is to consider a factored critic function among all the agents:

$$\tilde{Q}_w(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) = \sum_{m=1}^{M} f_w^m\big(s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{s}}, d_t\big), \qquad (6.7)$$

Such a factored critic structure is particularly suited for the credit assignment problem

as we are explicitly assigning (and learning) f_w^m as an agent m's contribution to the global critic value. Crucially, we show later, the policy gradient computed using such a critic also gets factored among agents, which is essentially credit assignment at the level of gradients among agents. In the collective setting, counts are the sufficient statistic for planning, and we assume a homogenous stochastic policy. Therefore, the above equation simplifies as

$$\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w\big(i, j, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big) \tag{6.8}$$

Intuitively, there are $\mathrm{n}_t^{\mathrm{sa}}(i,j)$ agents which are in state i and taking action j, and the contribution of each of such agents is $f_w\big(i, j, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)$. Another property of such a factored action-value function is that it is consistent with the form of compatible value functions for $\mathbb{C}$Dec-POMDPs [118, 77].

Mean collective gradient: Once we have set the factored design of the critic in (6.8) which helps in the multiagent credit assignment, the next step is to compute the expectation $\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}}\Big[\tilde{Q}_w\Big]$. Previous work computed this expectation using Monte-Carlo sampling [77]. However, as mentioned earlier, computing this expectation in the closed form helps reduce the variance [22, 6]. Therefore, we next develop techniques to compute this expectation in the closed form. We start by the following result from [63]:

Proposition 6.2. *The collective distribution $P^\pi(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}})$ has mean $\mathrm{n}_t^{\star\,\mathrm{sa}}$ which is a vector of length $|S| \times |A|$ with $\mathrm{n}_t^{\star\,\mathrm{sa}}(i,j) = \mathrm{n}_t^{\mathrm{s}}(i) \times \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ and the co-variance matrix is a square matrix with each element given as $\tilde{\Sigma}_t(i,j,i',j') = 0 \;\; \forall i \neq i'; \tilde{\Sigma}_t(i,j,i,j') = \mathrm{n}_t^{\mathrm{s}}(i)\pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))\pi_t(j'|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ otherwise.*

Using the above result, we prove the following theorem. We also use a more general definition of f_w which can depend on the whole count table $\mathrm{n}_t^{\mathrm{s}}$:

Theorem 6.1. *Linear collective critic function $\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) + b(\mathrm{n}_t^{\mathrm{s}})$ has the expected policy gradient under the collective distribution $P^\pi(\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}})$ and the*

policy π^θ as:

$$\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}} \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \nabla_\theta \sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{s}}(i) \pi_t^\theta(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) \times f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) \tag{6.9}$$

Proof. Let $\boldsymbol{f}_w(\mathbf{n}_{s_t})$ be the vector $(f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t))_{i,j}$. Notice that $\boldsymbol{f}$ and $b(\mathrm{n}_t^{\mathrm{s}})$ are both independent of action counts $\mathrm{n}_t^{\mathrm{sa}}$ and θ under the given state counts $\mathbf{n}_{s_t}$. The expected policy gradient has the form:

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}} \Big[\mathbf{n}_{s_t a_t} \cdot \boldsymbol{f}_w(\mathbf{n}_{s_t}) + b(\mathrm{n}_t^{\mathrm{s}}) \Big] \\
&= \nabla_\theta \big(\boldsymbol{f}_w(\mathbf{n}_{s_t}) \cdot \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}}[\mathbf{n}_{s_t a_t}] \big) + \nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}}[b(\mathrm{n}_t^{\mathrm{s}})] \\
&\text{using proposition 6.2 and noticing that the last term is zero:} \\
&= \nabla_\theta \big(\boldsymbol{f}_w(\mathbf{n}_{s_t}) \cdot \mathrm{n}^{\star\,\mathrm{sa}} \big) \\
&= \nabla_\theta \sum_{i,j} \mathrm{n}_t^{\mathrm{s}}(i) \pi_t(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t)
\end{aligned}$$

□

The above result shows that computing the inner expectation in the collective policy gradient (6.2) using an approximate action-value function f_w can be computed in the closed form, and hence the variance of the gradient is reduced. Furthermore, the gradient is also decomposable among agents, which makes its computation fast and efficient using only count samples.

6.2.2 Mean Collective Policy Update from the Global Critic

To highlight, we have two contrasting objectives to achieve: (i) We would like to have a decomposable critic (that results in a decomposable policy gradient) among agents that helps in the multiagent credit assignment, and (ii) learn such a critic with global rewards which are not decomposable among agents. We next outline how to achieve these objectives. Our key insight is that instead of learning a decomposable critic, we *learn a global critic* which is not factorized among agents. This addresses the problem of learning from global rewards;

as the critic is defined over the input from all the agents (count tables $\mathbf{n}$ in our case), we can learn from global rewards. However, instead of computing the policy gradient directly from the global critic, we compute policy gradients from a *linear approximation* to the global critic using first-order Taylor approximation. Actor update using linear approximation of the critic was studied previously in [22, 110]. As noted by researchers in their continuous action single agent domains, given the small step size, the linear critic approximation is sufficient to estimate the direction of the policy gradient to move towards a higher value. The key usefulness of the linear critic in our case is its relationship with that of multiagent credit assignment which we show next.

Consider the global $\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t)$, we consider its first order Taylor expansion at the mean value of action counts $\mathrm{n}_t^{\star\,\mathrm{sa}} = \mathbb{E}[\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, d_t] = \langle \mathrm{n}_t^{\mathrm{s}}(i)\pi(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}}))\forall i, j\rangle$ (Proposition 6.2) with π^* as the current policy:

$$\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) \approx \tilde{Q}_w(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) + (\mathrm{n}_t^{\mathrm{sa}} - \mathrm{n}_t^{\star\,\mathrm{sa}})^{\intercal}(\nabla_{\mathrm{n}^{\mathrm{sa}}}\tilde{Q}_w|_{=\mathrm{n}_t^{\star\,\mathrm{sa}}}) \tag{6.10}$$

We next show that the above linear approximation fits into the factored critic in theorem 6.1 by re-writing this Taylor expression as:

$$= \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j)\frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) + \left[\tilde{Q}_w(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) - (\mathrm{n}_t^{\star\,\mathrm{sa}})^{\intercal}(\nabla_{\mathrm{n}^{\mathrm{sa}}}\tilde{Q}_w|_{=\mathrm{n}_t^{\star\,\mathrm{sa}}})\right]$$

Using the above expression, we can see the relation:

$$f_w(i, j, \mathrm{n}_t^{\mathrm{s}}, d_t) = \frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t)$$
$$b(\mathrm{n}_t^{\mathrm{s}}) = \tilde{Q}_w(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) - (\mathrm{n}_t^{\star\,\mathrm{sa}})^{\intercal}(\nabla_{\mathrm{n}^{\mathrm{sa}}}\tilde{Q}_w|_{=\mathrm{n}_t^{\star\,\mathrm{sa}}})$$

Applying theorem 6.1, we have:

Corollary 6.1. *Using the first-order Taylor approximation of the critic at the expected state-action counts* $\mathrm{n}_t^{\star\,\mathrm{sa}} = \mathbb{E}[\mathrm{n}_t^{\mathrm{sa}} \mid \mathrm{n}_t^{\mathrm{s}}, d_t; \pi]$*, the collective policy gradient is:*

$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{H} \mathbb{E}_{\mathrm{n}_t^{\mathrm{s}}, d_t | b_o, b_o^d} \Big[\sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{s}}(i) \nabla_\theta \pi_t(j | i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})) \frac{\partial Q_w}{\partial\, \mathrm{n}^{\mathrm{sa}}(i,j)} (\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) \Big] \quad (6.11)$$

Intuitively, credit assignment is done using the term $\frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}_t^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t)$. When the expression $\frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}_t^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t)$ has a high value, it implies that a higher count of agents in state i and taking action j would increase the overall critic value $\tilde{Q}$. This will encourage more agents to take action j in state i. Thus, this term assigns appropriately the gradient for different actions of agents in different states.

6.3 Difference Rewards Based Credit Assignment

Difference rewards provide a powerful way to perform credit assignment when there are several agents, and have been explored extensively in the MRL literature [128, 2, 126, 127, 26]. Difference rewards (DR) are *shaped rewards* that help individual agents filter out the *noise* from the global reward signal (which includes effects from other agents' actions), allowing them to assess their individual contribution to the global reward. We will discuss two popular types of DRs—wonderful life utility (WLU) and aristocratic utility (AU) [128].

Wonderful Life Utility (WLU): Let $\boldsymbol{s}, \boldsymbol{a}$ denote the joint state-action; and $r(\boldsymbol{s}, \boldsymbol{a}, d)$ be the system reward. The WLU based DR for an agent m is $r^m = r(\boldsymbol{s}, \boldsymbol{a}, d) - r(\boldsymbol{s}, \boldsymbol{a}^{-m}, d)$ where $\boldsymbol{a}^{-m}$ is the joint-action without the agent m. The WLU DR compares the global reward to the reward received when agent m is not in the system. Agent m can use this shaped reward r^m for its individual learning. However extracting such shaped rewards from the simulator is very challenging and may not be feasible for a large number of agents. Therefore, we apply this reasoning to the critic (or action-value function approximator) $Q_w(\mathrm{n}^{sa}, d)$. We assume that Q_w is differentiable in all input parameters. Similar to WLU, we define WLQ (*wonderful life Q-function*) for an agent m as $Q^m = Q_w(\mathrm{n}^{\mathrm{sa}}, d) - Q_w(\mathrm{n}^{\mathrm{sa}-m}, d)$ where $\mathrm{n}^{\mathrm{sa}-m}$ is the state-action count table without the agent m.

For a given $(\mathrm{n}^{\mathrm{sa}}, d)$, we show how to estimate Q^m. Assume that the agent m is in some state $i \in S$ and performing action $j \in A$. As agents do not have identities, we use Q^{ij}

to denote the WLQ for any agent in state-action (i, j). Let e^{ij} be a vector with the same dimension as n^{sa}; all entries in e^{ij} are zero except value 1 at the index corresponding to state-action (i, j). We have $Q^{ij} = Q_w(\mathrm{n}^{\mathrm{sa}}, d) - Q_w(\mathrm{n}^{\mathrm{sa}} - e^{ij}, d)$. Typically, critic Q_w is represented using a neural network; we normalize all count inputs to the network (denoted as $\tilde{\mathrm{n}}^{\mathrm{sa}} = \mathrm{n}^{\mathrm{sa}}/M$) using the total agent population M. We now estimate WLQ assuming that M is large:

$$
\begin{aligned}
Q^{ij} &\approx \lim_{M\to\infty} \left[Q_w\left(\mathrm{n}^{\mathrm{sa}}/M, d\right) - Q_w\left((\mathrm{n}^{\mathrm{sa}} - e^{ij})/M, d\right)\right] \\
&= \lim_{\Delta=1/M\to 0} \left[Q_w\left(\tilde{\mathrm{n}}^{\mathrm{sa}}, d\right) - Q_w\left(\tilde{\mathrm{n}}^{\mathrm{sa}} - \Delta \cdot e^{ij}, d\right)\right] \\
&= -1 \cdot \lim_{\Delta=1/M\to 0} \left[Q_w\left(\tilde{\mathrm{n}}^{\mathrm{sa}} - \Delta \cdot e^{ij}, d\right) - Q_w\left(\tilde{\mathrm{n}}^{\mathrm{sa}}, d\right)\right] \qquad (6.12) \\
&= -1 * (-\Delta) \frac{\partial Q_w}{\partial \tilde{\mathrm{n}}^{\mathrm{sa}}(i,j)}(\tilde{\mathrm{n}}^{\mathrm{sa}}, d) \qquad \text{(by definition of total differential)} \\
Q^{ij} &\approx \frac{1}{M} \frac{\partial Q_w}{\partial \tilde{\mathrm{n}}^{\mathrm{sa}}(i,j)}(\tilde{\mathrm{n}}^{\mathrm{sa}}, d) \qquad (6.13)
\end{aligned}
$$

Thus, upon experiencing the tuple $(\mathrm{n}_t^{\mathrm{s}}, d_t, \mathrm{n}_t^{\mathrm{sa}}, \mathrm{n}_t^{\mathrm{sas}}, d_{t+1}, r_t)$, global reward r_t is used to train the global critic Q_w. An agent m in state-action (i, j) accumulates the gradient term $Q^{ij} \nabla_\theta \log \pi_t(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}}))$ as per the standard policy gradient result [118](notice that policy π is the same for all the agents). Given that there are $\mathrm{n}_t^{\mathrm{sa}}(i, j)$ agents performing action j in state i, the total accumulated gradient based on WLQ updates (6.13) by all the agents for all time steps is given as:

$$
\nabla_\theta^{wlq} J(\theta) = \sum_{t=1}^{H} \mathbb{E}_{d_t, \mathrm{n}_t^{\mathrm{sa}} | b_o, b_o^g} \left[\sum_{i \in S, j \in A} \mathrm{n}_t^{\mathrm{sa}}(i, j) Q_t^{ij}(\mathrm{n}_t^{\mathrm{sa}}, d_t) \nabla_\theta \log \pi_t(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})) \right] \qquad (6.14)
$$

We can estimate $\nabla_\theta^{wlq} J(\theta)$ by sampling counts and the state d_t for all the time steps.

Aristocratic Utility (AU): For a given joint state-action $(\boldsymbol{s}, \boldsymbol{a}, d)$, the AU based DR for an agent m is defined as $r^m = r(\boldsymbol{s}, \boldsymbol{a}, d) - \sum_{a^m} \pi^m(a^m|o^m(\boldsymbol{s}, d)) r(\boldsymbol{s}, \boldsymbol{a}^{-m} \cup a^m, d)$ where $\boldsymbol{a}^{-m} \cup a^m$ is the joint-action where agent m's action in $\boldsymbol{a}$ is replaced with a^m; o^m is the observation of the agent; π^m is the probability of action a^m. The AU marginalizes over all

the actions of agent m keeping other agents' actions fixed.

In the context of deep multi-agent policy gradient with value function approximation, Foerster et al. [30] show that computing the *counterfactual* value for each agent i can be computed by subtracting an agent-dependent baseline from the critic. In particular, given a centralized critic function $\tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d)$, where $(\boldsymbol{s}, \boldsymbol{a}, d)$ denote the joint state, action of agents), the advantage value of sampled action a^m of an agent m is computed as:

$$A^m(\boldsymbol{s}, \boldsymbol{a}, d) = \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}, d) - \sum_{a'^m} \pi^m(a^m | o^m(\boldsymbol{s}, d)) \tilde{Q}_w(\boldsymbol{s}, \boldsymbol{a}^{-m} \cup a'^m, d) \tag{6.15}$$

in which $\boldsymbol{a}^{-m} \cup a'^m$ is the joint action obtained by replacing sampled action a^m of the agent m in the sample $\boldsymbol{a}$ by a'^m. Then for each sampled joint trajectory $(\boldsymbol{s}_t, \boldsymbol{a}_t)_{t=1:H}$, the policy gradient is computed as:

$$\nabla_\theta \sum_t \sum_m A^m(\boldsymbol{s}_t, \boldsymbol{a}_t, d_t) \log \pi^m(a_t^m | o_t^m(\boldsymbol{s}_t, d_t)) \tag{6.16}$$

The agent-dependent baseline in the right hand side of (6.15) helps to reduce the variance of policy gradient. We next extend this counterfactual computation to our collective setting. We first notice that in the collective setting, by aggregating agents taking the same action in the same state into sampled counts $\mathbf{n}_{\boldsymbol{s}_t \boldsymbol{a}_t}$, we have:

Proposition 6.3. *The counterfactual (COMA) policy gradient in $\mathbb{C}$Dec-POMDPs can be computed as:*

$$\nabla_\theta \sum_t \sum_{i \in S_t, j \in A_t} A_t^{i,j}(\mathrm{n}_t^{\mathrm{sa}}, d_t)\, \mathrm{n}_t^{\mathrm{sa}}(i,j) \log \pi\big(j | i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)\big)$$

where we have:

$$A^{ij}(\mathrm{n}_t^{\mathrm{sa}}, d_t) = Q_w(\mathrm{n}^{\mathrm{sa}}, d) - \sum_{j'} \pi\big(j' | i, o(i, \mathrm{n}^{\mathrm{s}}, d)\big) Q_w(\mathrm{n}^{\mathrm{sa}} - e^{ij} + e^{ij'}, d) \tag{6.17}$$

where vectors e^{ik} are defined as for WLQ.

Notice that computing collective COMA policy gradient is expensive in $\mathbb{C}$Dec-POMDPs because we have to evaluate all possible changes in action counts. Therefore, we use

a similar technique as for WLQ by normalizing counts, and computing differentials $\lim_{\Delta=1/M\to 0}\left[Q_w(\tilde{\mathrm{n}}^{\mathrm{sa}},d)-Q_w(\tilde{\mathrm{n}}^{sa}+\Delta\cdot(e^{ij'}-e^{ij}),d)\right]$ to have:

Lemma 6.1. *We can approximate COMA advantage $A_t^{i,j}(\mathrm{n}_t^{\mathrm{sa}},d_t)$ as:*

$$A_t^{ij}(\mathrm{n}_t^{\mathrm{sa}},d_t)\approx\frac{1}{M}\Big[\frac{\partial\tilde{Q}_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j)}(\tilde{\mathrm{n}}_t^{\mathrm{sa}},d_t)-\sum_{j'}\pi\big(j'|i,o(i,d_t,\mathrm{n}_t^{\mathrm{s}})\big)\frac{\partial\tilde{Q}_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j')}(\tilde{\mathrm{n}}_t^{\mathrm{sa}},d_t)\Big] \tag{6.18}$$

Proof. We show how to estimate A^{ij} in equation (6.17) in assuming agent population M is large. Similar to WLQ, we have count inputs normalized for Q_w.

$$\begin{aligned}
A^{ij}&\approx\lim_{M\to\infty}\left[Q_w(\,^{\mathrm{n}^{\mathrm{sa}}}\!/M,d)-\sum_{j'}\pi\big(j'|i,o(i,d,\mathrm{n}^{\mathrm{s}})\big)Q_w(\,^{(\mathrm{n}^{\mathrm{sa}}-e^{ij}+e^{ij'})}\!/M,d)\right]\\
&=\sum_{j'}\pi(j'|i,o(i,d,\mathrm{n}^{\mathrm{s}}))\lim_{\Delta=1/M\to 0}\left[Q_w(\tilde{\mathrm{n}}^{\mathrm{sa}},d)-Q_w(\tilde{\mathrm{n}}^{sa}+\Delta\cdot(e^{ij'}-e^{ij}),d)\right]\\
&=\sum_{j'}\pi(j'|i,o(i,d,\mathrm{n}^{\mathrm{s}}))\big[\Delta\cdot\frac{\partial Q_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j)}(\tilde{\mathrm{n}}^{\mathrm{sa}},d)-\Delta\cdot\frac{\partial Q_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j')}(\tilde{\mathrm{n}}^{\mathrm{sa}},d)\big]\\
A_t^{ij}(\mathrm{n}_t^{\mathrm{sa}},d_t)&\approx\frac{1}{M}\Big[\frac{\partial\tilde{Q}_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j)}(\tilde{\mathrm{n}}_t^{\mathrm{sa}},d_t)-\sum_{j'}\pi\big(j'|i,o(i,d_t,\mathrm{n}_t^{\mathrm{s}})\big)\frac{\partial\tilde{Q}_w}{\partial\tilde{\mathrm{n}}^{\mathrm{sa}}(i,j')}(\tilde{\mathrm{n}}_t^{\mathrm{sa}},d_t)\Big]
\end{aligned} \tag{6.19}$$

where we used total differential similar to WLQ to derive (6.18) from (6.19). □

Crucially, the above computation is independent of agent population M, and is thus highly scalable. Using the same reasoning as WLQ, the gradient ∇_θ^{au} is exactly the same as (6.14) with Q_t^{ij} replaced by advantages A_t^{ij} in (6.18). Empirically, we observed that using advantages A^{ij} resulted in better quality because the additional term $\sum_{j'}$ in A^{ij} acts as a baseline and reduces variance.

Another way to derive the approximate COMA advantages is by using the first order Taylor expansion of the critic $\tilde{Q}_w$:

Lemma 6.2. *By approximating the critic $\tilde{Q}_w$ with its first-order Taylor approximation at the sampled count $\mathrm{n}_t^{\mathrm{sa}}$, we can approximate COMA advantage $A_t^{i,j}(\mathrm{n}_t^{\mathrm{sa}},d_t)$ as:*

$$\frac{\partial\tilde{Q}_w}{\partial\,\mathrm{n}^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\mathrm{sa}},d_t)-\sum_{j'}\pi\big(j'|i,o(i,\mathrm{n}_t^{\mathrm{s}},d_t)\frac{\partial\tilde{Q}_w}{\partial\,\mathrm{n}^{\mathrm{sa}}(i,j')}(\mathrm{n}_t^{\mathrm{sa}},d_t)$$

Algorithm 4: Policy Gradient with Global Rewards

Initialize network parameter θ for actor π and and w for critic $\tilde{Q}_w$
$\gamma \leftarrow$ actor learning rate
$\beta \leftarrow$ critic learning rate
repeat
  Sample count vectors $\mathbf{n}_{1:H} \sim P(\boldsymbol{n}_{1:H}; \pi)$ and empirical reward $r_{1:H}$
  Update critic by $\mathbf{n}_{1:H} \sim P(\boldsymbol{n}_{1:H}; \pi)$ and $r_{1:H}$ with learning rate β
  Computing the policy gradient δ_θ by:
  MCAC: $\nabla_\theta \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{s}}(i) \pi\big(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})\big) \frac{\partial \tilde{Q}_w}{\partial n_t(i,j)}(\mathrm{n}_t^{\star\,\mathrm{sa}}, d_t) \Big]$
  CCAC: $\nabla_\theta \sum_t \Big[\sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) \log \pi\big(j|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})\big)$
  $\big(\frac{\partial \tilde{Q}_w}{\partial n_t(i,j)}(\mathrm{n}_t^{\mathrm{sa}}, d_t) - \sum_{j'} \pi\big(j'|i, o(i, d_t, \mathrm{n}_t^{\mathrm{s}})\big) \frac{\partial \tilde{Q}_w}{\partial n_t(i,j')}(\mathrm{n}_t^{\mathrm{sa}}, d_t) \Big]$
  Update actor $\theta \rightarrow \theta + \gamma \delta_\theta$.
until ***convergence***
return θ, w

Proof. The first-order Taylor expansion of the critic function $\tilde{Q}(\mathrm{n}_t'^{\mathrm{sa}}, d_t)$ at the sampled action counts $\mathrm{n}_t^{\mathrm{sa}}$ is:

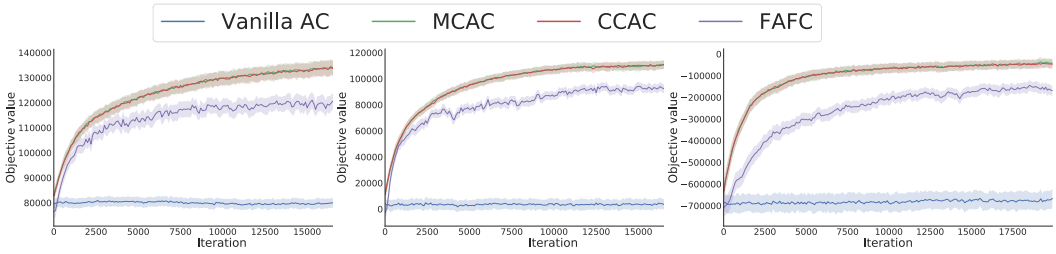
$$\approx \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) + (\mathrm{n}_t'^{\mathrm{sa}} - \mathrm{n}_t^{\mathrm{sa}}) \nabla_{\mathrm{n}^{\mathrm{sa}}} \tilde{Q}_w|_{=\mathrm{n}_t^{\mathrm{sa}}}$$

Using this approximation in the second term of right hand side of (6.17), we have:

$$\begin{aligned} & \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}} - e^{ij} + e^{ij'}) \\ \approx\ & \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) + (-e^{ij} + e^{ij'}) \nabla_{\mathrm{n}^{\mathrm{sa}}} \tilde{Q}_w|_{=\mathrm{n}_t^{\mathrm{sa}}} \\ =\ & \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \Big(- \frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}^{\mathrm{sa}}(i,j)}(\mathrm{n}_t^{\mathrm{sa}}, d_t) + \frac{\partial \tilde{Q}_w}{\partial\, \mathrm{n}^{\mathrm{sa}}(i,j')}(\mathrm{n}_t^{\mathrm{sa}}, d_t) \Big) \end{aligned} \tag{6.20}$$

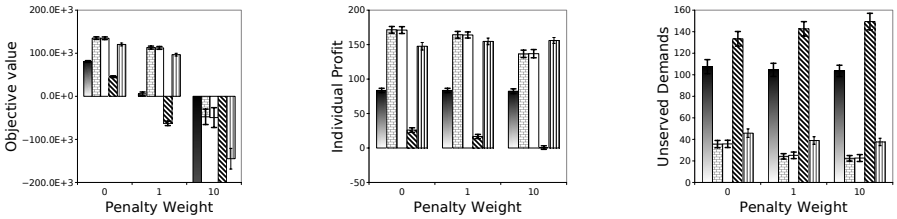
Replace (6.20) into the (6.17), we prove the lemma. □

Empirically, we noticed that in problems with a small number of agents, mean collective actor critic (MCAC) worked better. Intuitively, the reason is because by using the mean of action counts, $\mathrm{n}_t^{\star\,\mathrm{sa}}$, and computing the expectation $\nabla_\theta \mathbb{E}_{\mathrm{n}_t^{\mathrm{sa}}} \tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t)$ in closed form, the policy gradient variance is lower than the variance in the collective COMA case where we use sampled action counts $\mathrm{n}_t^{\mathrm{sa}}$. For domains with a large number of agents, sampled action-counts are much closer to the mean action-counts. Therefore in such domains, collective COMA performed similar to the MCAC approach. The MCAC and collective counterfactual actor critic (CCAC) are summarized in algorithm 4.



(a) $w = 0$ (b) $w = 1$ (c) $w = 10$

Figure 6.2: Convergence of different actor-critic variants on the taxi problem. The curves for MCAC and CCAC almost overlap.



(a) Objective value (b) Average profit per taxi (c) Average unserved

Figure 6.3: Different metrics on the taxi problem with different penalty weights w.

6.4 Experiments

In this section, we experimentally show the effectiveness of our two proposed approaches, mean collective actor critic (MCAC) and collective counterfactual actor critic (CCAC), for a real world inspired taxi supply-demand matching problem [77], a real-world police patrolling benchmark and a synthetic robot patrolling in a grid domain. We represent policies using neural networks. Their design and other experiments settings are described in the appendix.

We compare the performance of MCAC and CCAC against the following approaches:

- Standard actor critic ('vanilla AC') which fits the critic using global rewards and computes policy gradients from the global critic (without the linear Taylor approximation of critic).
- The factored actor critic ('fAfC') approach of [77] which is the previous best RL approach for $\mathbb{C}$Dec-POMDPs with *decomposable* rewards. In our domains (specifically the taxi problem), we have both local and global rewards. The local rewards are incorporated in 'fAfC' as before; for global rewards, we change the training procedure of the

critic in 'fAfC'.

The exact actor and critic updates for both 'vanilla AC' and 'fAfC' are provided in the appendix.

6.4.1 Taxi Supply-Demand Matching

We test our approach on this real-world motivated problem (described in Section 6.1). In particular, we consider a fleet of 8000 taxis (or agents) in the city divided in 81 zones. In addition to individual revenues, we also have *global rewards* to maintain a *quality-of-service* (QoS) above some threshold for selected zones which cover the central business district and residential areas. We analyzed the demand data and selected the topmost 15 busiest zones for such global rewards. To enforce QoS level $\alpha = 95\%$ for each zone i and time t, we add penalty terms $w \times \left(\hat{d}_t(i) - \alpha d_t(i)\right)$ where w is the penalty weight, $\hat{d}_t(i)$ is the total *served* demand at time t, and $d_t(i)$ is the total demand at time t. We test the effect of QoS penalty by using different penalty weights $w \in \{0, 1.0, 10.0\}$. Intuitively, $w = 1$ implies the (average) penalty for missing a customer is the negative of (average) reward for serving a customer in zone i; $w = 10$ implies missing a customer is 10 times more expensive than the average reward from serving 1 customer. Notice that even though the total profit is decomposable among agents, the QoS penalty is not.

In addition to comparisons against 'Vanilla AC' and 'fAfC', we also compare against a strong online centralized planner (which fully observes all taxi locations, current demand) based on the online anticipatory approach ('OAA') [67]. Details of this online planner are in the appendix.

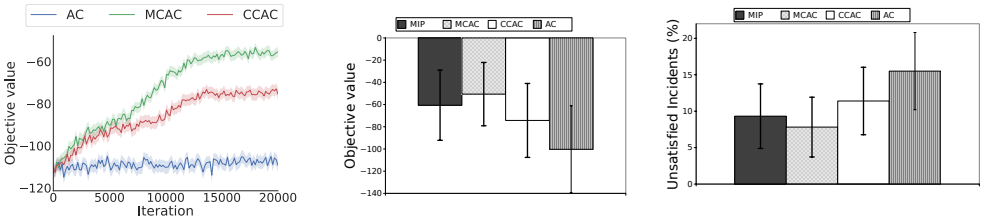
Convergence: Figure 6.2 shows the convergence of different actor-critic approaches on the taxi problem with different weights w, and QoS (α) being 95%. We use QoS penalty for all the 48 time steps for busiest 15 zones. Figures 6.2(a-c) clearly show that both MCAC and CCAC have much better convergence rate than the previous best 'fAfC' approach which is slow to converge due to the presence of global rewards. Furthermore, 'Vanilla AC' performs poorly on these instances as it does not explicitly takes into account the issue of multiagent credit assignment and low variance gradient estimates. Given that the agent population is

high (8000), this results in worst performance by 'Vanilla AC'. These results illustrate that the linear approximation of the critic is effective in doing multiagent credit assignment and moves the policy in the right direction in both MCAC and CCAC.

Quality: Figure 6.3(a) shows final quality (that includes both the total fleet profit and the penalty term) comparisons among different actor-critic approaches and the centralized 'OAA' planner (we exclude 'vanilla AC' as it does not learn any useful policy). As expected, 'OAA' provides best quality. However, MCAC and CCAC are competitive with 'OAA' in the final objective value. For $w=0$ (i.e., the objective is just profit maximization), the final quality by CCAC and MCAC is about 89% of the 'OAA' quality and for $w=1$, it is about 81%. The 'fAfC' achieves about 85% of the quality by MCAC for both $w=1$ and $w=0$, confirming the superior handling of global rewards by our approaches. For higher penalty $w=10$, we see that all the RL approaches are worst off mainly because the penalty term $w \times \left(\hat{d}_t(i) - \alpha d_t(i)\right)$ overshadows the fleet profit term due to high w. We next show that a higher value of w does not necessarily benefit RL approaches and a lower value of w is preferable to keep a balance between QoS and overall profit.

Effectiveness with global rewards: Figures 6.3(b,c) together show the tradeoff between the overall fleet profit (figure 6.3(b) shows the average profit per taxi per day) and the QoS (figure 6.3(c) shows unserved demand below the QoS threshold or $(\alpha \cdot d_t(i) - \hat{d}_t(i))$ averaged over all 15 zones and all the time steps). From these two figures, we can see that when the penalty increased from $w=0$ to 1 in Figure 6.3(b), the average profit remains almost the same for both MCAC and CCAC. However, Figure 6.3(c) shows that the unserved demand decreased significantly (by 32%) for both MCAC and CCAC from $w=0$ to $w=1$. Increasing the weight w to 10 did not significantly decrease the unserved demand (in Figure 6.3(c)) but led to the drop in profit (as shown in 6.3(b)) for both MCAC and CCAC. Interestingly, 'fAfC' fails to achieve such a balance between profit and the unserved demand. The unserved demand by 'fAfC' does not decrease much from $w=0$ to $w=1$. Intuitively this is because the QoS penalty constitutes global rewards whereas 'fAfC' is optimized for decomposable rewards and cannot learn effectively from global reward signals.

6.4.2 Police Patrolling



(a) Algorithm convergence (b) Objective value (c) Unsatisfied Percentage

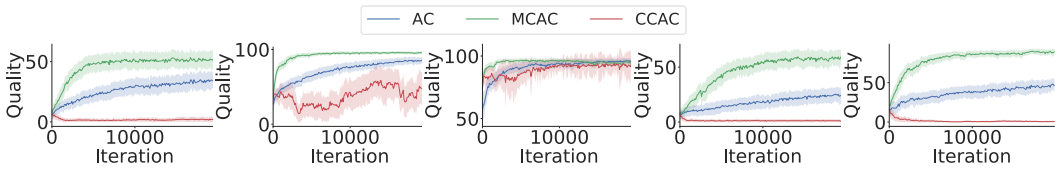
Figure 6.4: Police patrolling problem.

The problem is introduced in Section 6.1. There are 24 city zones, and 16 patrol cars (or agents). We have access to real world data about all incidents for 31 days in 24 zones. Roughly 50-60 incidents happen per day (7AM-7PM shift). The goal is to compute reallocation policy for agents such that the number of incidents with response time more than the threshold is minimized (further details in appendix). This domain has only global rewards. Therefore, we compare MCAC, CCAC and AC (Note: fAfC, AverageFlow are unable to model this domain). As a baseline, we compare against a static allocation of patrol cars that is optimized using a stochastic math program [21], denoted as 'MIP'. Figure 6.4(a) shows the convergence results. MCAC performs much better than CCAC. This is because this problem is sparse with sparse tables n^{sa}, resulting in higher gradient variance for CCAC; MCAC marginalizes out n^{sa}, thus has lower variance. Figure 6.4(b) shows overall objective comparisons (higher is better) among all three approaches. It confirms that MCAC is the best approach. MCAC has 7.8% incidents where response time was more than the threshold versus 9.32% for MIP (figure 6.4(c)). Notice that even this improvement is significant as it allows ≈25 more incidents to be served within the threshold over 31 days (assuming 55 avg. incidents/day). In emergency scenarios, improving response time even by a few minutes is potentially life saving.

6.4.3 Synthetic Robot Patrolling Game

Next, we consider a synthetic grid world problem to test the sensitivity of our proposed algorithms (MCAC and CCAC) with respect to different population sizes. The goal is for a population of robots to move from a depot to find a victim in an MxM grid. At each time

step, there would be a victim at a uniformly distributed location in the grid. Robots have to cooperate with each other to cover the whole grid to maximize the total number of rescued victims. A global reward to the whole team is given when the victim is rescued by any agent. The reward is not decomposable among agents because regardless of the number of agents reaching the victim's location, only a fixed team reward is given. In this domain, as only global reward signal given, 'fAfC' is not applicable; we only compare against the 'vanilla AC'. Figure 6.5 shows the convergence results among different approaches for varying population size and grid dimensions. In this setting, MCAC is significantly better than CCAC as the domain is sparse with a small number of agents. Intuitively, MCAC computes policy gradient directly from mean action counts; where CCAC computes the policy gradient using sampled actions counts (which are sparse as number of agents is small). Therefore, the variance of the gradient estimate is high for CCAC, and as also highlighted in section 4, CCAC does not work well in such sparse domains.



(a) $N=5$ in (5x5) (b) $N=20$ in (5x5) (c) $N=50$ in (5x5) (d) $N=20$ in (9x9) (e) $N=50$ in (9x9)

Figure 6.5: Convergence of different actor-critic variants on the grid patrolling with varying population size N and grid size.

6.5 Related Works

6.5.1 Difference of Reward

To derive our CCAC algorithm for $\mathbb{C}$Dec-POMDP domains, we base our model on the notion of difference reward proposed by Wolpert and Tumer [149] with Wonderful Life Utility (WLU) and Aristocrat Utility (AU) algorithms. The difference of reward addresses the credit assignment problem by estimating the contribution of an agent by the difference of system utility between with and without that agent cases. Difference rewards (DR) are *shaped rewards* that help individual agents filter out the *noise* from the global reward signal (which includes effects from other agents' actions), and assess their individual contribution

to the global reward [149]. [128] showed WLU andAU can efficiently solve multi-agent sequential planning problems by considering the difference of value functions. Later on, Foerster et al. [30] extended AU algorithm into policy gradient algorithm by considering a counterfactual difference of critic value into individual policy update. In addition, Foerster et al. [30] proved that their counterfactual difference method introduced no bias into the policy gradient computation. These methods, however, are not scalable to large populations. Our main contribution in designing CCAC is to show how to apply difference-of-reward into $\mathbb{C}$Dec-POMDP settings.

6.5.2 Expected Policy Update

One remedy for the high variance of stochastic policy gradient is to compute an analytic expression of the policy gradient. The idea was first suggested to apply to tabular Q-learning in the book of Sutton and Barto [117] as expected Sarsa. Instead of a Q-value update based on a deterministic action a_{t+1}, Sutton and Barto [117] proposed that value could be computed by an expectation over all possible actions. Seijen et al. [104] later proved the benefit of expected Sarsa in reducing the variance of Sarsa algorithm. Recently, Asadi et al. [6] and Ciosek and Whiteson [22] proposed expected policy gradient methods which have lower variance than stochastic policy gradient. Instead of updating the policy by sampled action, expected policy gradient approaches directly compute policy gradient by an expectation over all possible action. Asadi et al. [6] showed this expectation can be easily done in discrete and finite actions problem. In continuous action domains, the expectation can be computed under a closed-form formula if the policy and value functions have Gaussian form [22]. In case of non-Gaussian value function, Ciosek and Whiteson [22] suggested that we can use a Taylor approximation of value function instead. In our multi-agent decision making in $\mathbb{C}$Dec-POMDPs, the action space is discrete (hence method in [22] is not applicable) and has exponential size due to joint value (it is impossible to use method in [6] to compute the expectation over all possible joint actions).

Our MCAC algorithm uses the mean of the action count to estimate the policy gradient. The action mean is also used in some RL algorithm in the literature. Gu et al. [36] and

Ciosek and Whiteson [22] used the action mean to compute Taylor approximation for value function of single agent. Tumer and Agogino [126] and Wu et al. [150] used mean action to estimate the difference of reward. Our MCAC approach is different from these work in its use of mean joint action to estimate policy gradient of multi-agent policy in $\mathbb{C}$Dec-POMDP domains. Our policy is decentralized and the value function depends on the joint-count of agents (in different states and actions) rather than the single-agent setting considered in [22, 36].

6.6 Summary

In this work, we addressed the problem of collective multiagent RL with global rewards. Our main contributions include developing techniques for multiagent credit assignment and computing low variance gradient estimates in the presence of global rewards. In such settings, we showed that an effective critic which is trainable using global rewards is not decomposable among agents. To use non-decomposable critic in multi-agent settings, we addressed the credit assignment problem by proposing MCAC and CCAC algorithms.

To derive MCAC, we highlighted a general structure of the critic in the multiagent RL setting that is suited for the credit assignment problem, but unfortunately is difficult to train using global rewards. Therefore, we developed techniques based on approximation of the critic that can resolve such contrasting requirements. For lower variance of the gradients, we showed how to compute *expected* or *mean* collective policy gradients by exploiting the special feature of $\mathbb{C}$Dec-POMDPs.

To derive CCAC algorithm, we used the notion of difference-of-reward/utility [126, 30] in multi-agent RL. We showed how difference-of-reward can be used in $\mathbb{C}$Dec-POMDP planning without agent identity. As the number of agents and joint action space are large, we derived an approximation of difference-of-reward using total differential. In large population, the contribution of one agent to the whole population becomes small, which makes the differential adequately approximate the difference-of-reward function.

We tested our approaches on a synthetic multirobot grid navigation domain, and a real

world supply-demand taxi matching problem in a large Asian city with 8000 taxis and a police re-allocation problem. Thanks to our techniques for multiagent credit assignment and low variance policy gradients, our multiagent RL algorithms converge to high quality solutions faster than the standard policy gradient method and the best factored actor-critic approach from Chapter 5. Our approaches are also competitive even with a strong centralized online planner based on anticipatory algorithms [67] despite decentralized and partially observable environment in our case.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

This thesis contributed to the literature of multi-agent systems by a "*lifted* " multi-agent planning framework using the count variables. Our framework allows us to develop multi-agent reinforcement learning algorithms to optimize decentralized policy of a large population (up to 8000 agents). In particular, we addressed the high complexity of joint trajectory by proposing a novel representation with agent counts. The counts are more compact than the joint trajectory as their dimensions depend only on the size of the local state spaces rather than the population size. Based on this count-based representation, we proposed collective reinforcement learning algorithms to solve large scale multi-agent planning problems with sampled values of the count variables. In local reward optimization problems, we proposed collective algorithms combined with fictitious play rule to be able to optimize individual policy. As inherited from fictitious play, our algorithms were also applicable to non-cooperative settings. Our fictitious play based algorithms could converge to a symmetric equilibrium in population game. However, similar to other fictitious play algorithms, convergence to equilibrium cannot be guaranteed in general. In global reward optimization problems, we addressed the *credit-assignment* problem in multi-agent system by proposing collective algorithms based on the notion of "difference of reward" and "approximation by differentials". We showed that our algorithms can efficiently optimize decentralized policy

in multiple cooperative multi-agent domains.

Our planning framework is based on two key ideas: the collective distribution of the counts in planning and the count-based value functions. These ideas were inspired by the counting formulas for lifted inference in Markov Logic Network [17], [69] and collective inference in Collective Graphical Model (CGM) [108]. The lifted inference technique was first proposed to compute marginal probability of individual state rather to learn individual behavior as in our case. Recently, there were research works extending the counting formulas in lifted inference to compute value functions in MDP planning [97], [113]. However, these works focused on finding policies of heterogeneous agents in domains with sparse interaction graph. On the contrary, our count-based planning framework considered domains where agents fully interact with each other. Second, although the collective distribution in CGM shares some similarities with ours, it is only applicable to domains where there are typically no interactions between agents. Our work is the first one that considers agent interactions and applying collective distribution of counts in multi-agent planning domains.

7.2 Future works

We believe that the contribution of this thesis is an important step to apply AI planning into real world domains such as traffic network controls with thousands or millions of autonomous vehicles. Nevertheless, there are important questions to be addressed in our future works, which include: 1) how to extend the $\mathbb{C}$Dec-POMDP algorithms to learn heterogeneous behaviors of agents; 2) how to handle the large state-action spaces; 3) how to use our $\mathbb{C}$Dec-POMDP solutions to aid centralized decision making. We provide a discussion on these questions in the next section.

7.2.1 Heterogeneous behaviours

In this thesis, we focus on systems of homogeneous agents. Under the homogeneity assumption, agents in a same local state have the same behaviour, i.e. an identical distribution over local actions. Because the optimal solution in multi-agent planning is not always ho-

mogeneous policy, optimality is traded for scalability.

One of common method to resolve this problem of shared policy in multi-agent reinforcement learning is to embed identities to agents to break the behavioural symmetry. When an agent enquires the shared policy to decide which action to take, the shared policy takes input of not only local state and local observation (which can be similar amongst some agents) but also the identity of the enquirer (which is *unique* to that agent). The identity of an agent can be in the form of a randomly generated feature vector [153, 152] or an integer index [83] of that agent. Although the identity can break the behavioural symmetry in shared policy, it makes the population completely heterogeneous. In other words, the number of agent types is equal to the number of agents. Consequently, the count tables have sparse values of $\{0; 1\}$ and the collective planning loses the compactness advantage.

One of our future works is to find a trade-off between optimality and scalability in collective planning. We plan to explore methods to break the behavioural symmetry but at the same time preserve the compactness of the count representation. We can pre-define a number of agent types. An agent is assigned to a type at the beginning so that it can behave differently from agents in other types. The pre-defined number of agent types is smaller than the number of agents, hence it is more tractable than the agent identity. Main research questions include how to assign type to agent an how to determine the optimal number of type. We leave these non-trivial research problems to the future work.

7.2.2 Large state space

Our collective planning framework with count representation assumes the number of local states is smaller than the number of agents, so the count tables can compactly represent the joint state of multi-agent systems. However, when the number of local states is large, the count tables become sparse. In the case when agents are in completely different local states, all values in the count table are in $\{0; 1\}$. This can happen, for example, in video game domains where the local state is the location of agent in a spatial map with high resolution or continuous space. It can also happen when the recurrent neural network is used and the memory state is represented by a vector. It is important to handle such scenarios.

Deep learning is known for the ability to reduce the dimension of the data [46], which can be used to regain the effectiveness of the counts. In particular, we can use a neural network to take high-dimensional features of local state as an input and output the abstract representation of that state. The abstract representation can be designed to be in finite space which is much smaller than the original space. We can define the count table over the abstract state instead of the original state. One example of a neural network abstraction is the convolutional neural network (CNN) [61] which constructs the abstract state representation by scanning through the 2D spatial feature by filters [34]. A filter in CNN is used to aggregate information of sub-regions in a big spatial map. In a special case when the filter size is similar to the stride (the jumping step in scanning the feature), applying CNN to the collective domains can be considered to be similar to divide the big map into a finite number of regions and count the number of agents in each region. Developing a general framework to combine neural networks with the count-based representation for collective planning is one of the future research avenues.

7.2.3 Online Decision Making

Decentralized execution is shown to be more tractable than centralized execution [19] because at a decision epoch, each decentralized controller considers a small number of local actions $a \in A$ instead of joint action space A^M which is exponential to the number of agents M. With the advantage of scalability, decentralized control can quickly make decisions on the actions for agents, which is critical to domains such as vehicle driving which requires requiring real-time decisions. On the other hand, although centralized solvers using linear programs (LPs), mixed integer programs (MIPs) or centralized MDP approaches take time (in minutes or sometimes hours) to determine the joint action, the solution quality is provably optimal or near to the optimum. To preserve optimality, centralized solvers are preferable in domains allowing planning time. For example, in logistics domains, the daily delivery routes of trucks can be planned one day in advance. Stochasticity can be considered in planning by using methods such as Sample Average Approximation [52, 134]. When the environment state changes from the expected state, one can re-solve the planning problem with the realized state to get dynamic action. However, the significant run-time of

centralized planning methods hinders the real-time re-solving. It is interesting to see a hybrid method combining the benefits of a centralized solver and decentralized reinforcement learning.

A centralized solver can be combined with decentralized reinforcement learning by value function approximation. Value function approximation (VFA) is a common technique to estimate the value of a joint state or joint decision. A centralized solver can resort to a compact VFA instead of generating and considering all possible future scenarios, which can reduce the complexity and consequently the run-time. The use of VFA in MIPs was studied before in 2-stage scheduling for inventory management [93], fleet management [33], etc. The method is also known as Approximate Dynamic Program (ADP) in Operations Research (OR) literature [94]. However, learning VFA for ADP using classical methods in OR can take a great deal of run-time. On the contrary, learning the critic as VFA using our $\mathbb{C}$Dec-POMDP RL algorithms is shown to be fast and accurate. For future works, we would like to try to use the critic function learnt by $\mathbb{C}$Dec-POMDP RL algorithms as a VFA for ADP to solve centralized online decision making problems.

Bibliography

[1] Aberdeen, D. (2006). Policy-gradient methods for planning. In *Advances in Neural Information Processing Systems*, pages 9–16.

[2] Agogino, A. K. and Tumer, K. (2004). Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 980–987. IEEE Computer Society.

[3] Ahmed, A., Varakantham, P., and Cheng, S.-F. (2012). Uncertain congestion games with assorted human agent populations. *arXiv preprint arXiv:1210.4848*.

[4] Amato, C., Oliehoek, F. A., et al. (2015). Scalable planning and learning for multiagent pomdps. In *AAAI*, pages 1995–2002.

[5] Arai, T., Pagello, E., and Parker, L. E. (2002). Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, 18(5):655–661.

[6] Asadi, K., Allen, C., Roderick, M., Mohamed, A.-R., Konidaris, G., and Littman, M. (2017). Mean Actor Critic. In *eprint arXiv:1709.00503*.

[7] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[8] Bagnell, J. A. and Ng, A. Y. (2005). On local rewards and scaling distributed reinforcement learning. In *International Conference on Neural Information Processing Systems*, pages 91–98.

[9] Balch, T., Stone, P., and Kraetzschmar, G. (2001). *RoboCup 2000: Robot Soccer World Cup IV*. Springer.

[10] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1988). Neuronlike adaptive elements that can solve difficult learning control problems. *Neurocomputing: foundations of research*, pages 535–549.

[11] Bayındır, L. (2016). A review of swarm robotics tasks. *Neurocomputing*, 172:292–321.

[12] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840.

[13] Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific, Cambridge, MA, USA.

[14] Bhatnagar, S., Ghavamzadeh, M., Lee, M., and Sutton, R. S. (2008). Incremental natural actor-critic algorithms. In *Advances in neural information processing systems*, pages 105–112.

[15] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Oxford university press.

[16] Bowling, M. and Veloso, M. (2001). Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, volume 17, pages 1021–1026. LAWRENCE ERLBAUM ASSOCIATES LTD.

[17] Braz, R. D. S., Amir, E., and Roth, D. (2005). Lifted first-order probabilistic inference. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 1319–1325. Citeseer.

[18] Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376.

[19] Buffet, O. and Aberdeen, D. (2009). The factored policy-gradient planner. *Artificial Intelligence*, 173(5):722–747.

[20] Chang, Y.-H., Ho, T., and Kaelbling, L. P. (2004). All learning is local: Multi-agent learning in global reward games. In *Advances in neural information processing systems*, pages 807–814.

[21] Chase, J., Du, J., Fu, N., Le, T. V., and Lau, H. C. (2017). Law enforcement resource optimization with response time guarantees. In *IEEE Symposium Series on Computational Intelligence*, pages 1–7.

[22] Ciosek, K. and Whiteson, S. (2018). Expected policy gradients. In *(AAAI) Conference on Artificial Intelligence*. AAAI.

[23] Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752.

[24] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977a). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical society, Series B*, 39(1):1–38.

[25] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977b). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

[26] Devlin, S., Yliniemi, L., Kudenko, D., and Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems.

[27] Diaconis, P. and Freedman, D. (1980a). De Finetti's generalizations of exchangeability. *Studies in Inductive Logic and Probability*, 2:233–249.

[28] Diaconis, P. and Freedman, D. (1980b). Finite exchangeable sequences. *The Annals of Probability*, 8(4):745–764.

[29] Foerster, J., Assael, Y. M., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145.

[30] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *(AAAI) Conference on Artificial Intelligence*. AAAI.

[31] Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., Whiteson, S., et al. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*.

[32] Gartner, N. (1983). Opac: A demand-responsive strategy for traffic signal control. *Transportation Research Record, Journal of the Transportation Research Board*, No. 906:75–81.

[33] Godfrey, G. A. and Powell, W. B. (2002). An adaptive dynamic programming algorithm for dynamic fleet management, ii: Multiperiod travel times. *Transportation Science*, 36(1):40–54.

[34] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[35] Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530.

[36] Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2017). Q-prop: Sample-efficient policy gradient with an off-policy critic. In *5th International Conference on Learning Representations (ICLR 2017)*.

[37] Guestrin, C., Koller, D., and Parr, R. (2002a). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, pages 1523–1530.

[38] Guestrin, C., Koller, D., and Parr, R. (2002b). Multiagent Planning with Factored {MDPs}. In *Advances in Neural Information Processing Systems 14*, pages 1523–1530.

[39] Guestrin, C., Lagoudakis, M., and Parr, R. (2002c). Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234.

[40] Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer.

[41] Gupta, T., Kumar, A., and Paruchuri, P. (2018). Planning and learning for decentralized mdps with event driven rewards. In *(AAAI) Conference on Artificial Intelligence*. AAAI.

[42] Hajek, B. and van Loon, T. (1982). Decentralized dynamic control of a multiaccess broadcast channel. *IEEE transactions on automatic control*, 27(3):559–569.

[43] Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *International Conference on Uncertainty in Artificial Intelligence*, pages 211–219.

[44] Haynes, T. and Sen, S. (1995). Evolving behavioral strategies in predators and prey. In *International Joint Conference on Artificial Intelligence*, pages 113–126. Springer.

[45] Hennes, D., Tuyls, K., and Rauterberg, M. (2009). State-coupled replicator dynamics. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 789–796. International Foundation for Autonomous Agents and Multiagent Systems.

[46] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[47] Jiang, A. X. and Leyton-Brown, K. (2010). Bayesian action-graph games. In *Advances in Neural Information Processing Systems*, pages 991–999.

[48] Jiang, A. X., Leyton-Brown, K., and Bhat, N. A. (2011). Action-graph games. *Games and Economic Behavior*, 71(1):141–173.

[49] Jordan, M. I. and Weiss, Y. (2002). Probabilistic inference in graphical models. *Handbook of neural networks and brain theory*.

[50] Kaisers, M., Bloembergen, D., and Tuyls, K. (2012). A common gradient in multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on*

Autonomous Agents and Multiagent Systems-Volume 3, pages 1393–1394. International Foundation for Autonomous Agents and Multiagent Systems.

[51] Karkus, P., Hsu, D., and Lee, W. S. (2017). Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4697–4707.

[52] Kleywegt, A. J., Shapiro, A., and Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502.

[53] Kok, J. R. and Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828.

[54] Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

[55] Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166.

[56] Kraemer, L. and Banerjee, B. (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94.

[57] Kumar, A., Sheldon, D., and Srivastava, B. (2013). Collective diffusion over networks: Models and inference. In *International Conference on Uncertainty in Artificial Intelligence*, pages 351–359.

[58] Kumar, A., Zilberstein, S., and Toussaint, M. (2015). Probabilistic inference techniques for scalable multiagent decision making. *Journal of Artificial Intelligence Research*, 53(1):223–270.

[59] Kuyer, L., Whiteson, S., Bakker, B., and Vlassis, N. (2008). Multiagent reinforcement learning for urban traffic control using coordination graphs. *Machine learning and knowledge discovery in databases*, pages 656–671.

[60] Lambert Iii, T. J., Epelman, M. A., and Smith, R. L. (2005). A fictitious play approach to large-scale optimization. *Operations Research*, 53(3):477–489.

[61] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

[62] Lesser, V., Ortiz Jr, C. L., and Tambe, M. (2012). *Distributed sensor networks: A multiagent perspective*, volume 9. Springer Science & Business Media.

[63] Liu, L., Sheldon, D., and Dietterich, T. (2014). Gaussian approximation of collective graphical models. In *International Conference on Machine Learning*, pages 1602–1610.

[64] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393.

[65] Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2012). Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31.

[66] Melo, F. S. and Veloso, M. (2011). Decentralized mdps with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789.

[67] Mercier, L. and Van Hentenryck, P. (2007). Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *IJCAI*, pages 1979–1984.

[68] Meyers, C. A. and Schulz, A. S. (2012). The complexity of congestion games. *Networks*, 59:252–260.

[69] Milch, B., Zettlemoyer, L. S., Kersting, K., Haimes, M., and Kaelbling, L. P. (2008). Lifted probabilistic inference with counting formulas. In *Aaai*, volume 8, pages 1062–1068.

[70] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[71] Morimura, T., Osogami, T., and Idé, T. (2013). Solving inverse problem of markov chain with partial observations. In *Advances in Neural Information processing Systems*, pages 1655–1663.

[72] Murphy, R. R. (2000). Marsupial and shape-shifting robots for urban search and rescue. *IEEE Intelligent Systems and their applications*, 15(2):14–19.

[73] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Marsella, S. (2003). Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI*, volume 3, pages 705–711.

[74] Nair, R., Varakantham, P., Tambe, M., and Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI Conference on Artificial Intelligence*, pages 133–139.

[75] Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

[76] Nguyen, D. T., Kumar, A., and Lau, H. C. (2017a). Collective multiagent sequential decision making under uncertainty. In *AAAI Conference on Artificial Intelligence*, pages 3036–3043.

[77] Nguyen, D. T., Kumar, A., and Lau, H. C. (2017b). Policy gradient with value function approximation for collective multiagent planning. In *The Thirty-first Annual Conference on Neural Information Processing Systems*. NIPS.

[78] Nguyen, D. T., Kumar, A., Lau, H. C., and Sheldon, D. (2016). Approximate inference using DC programming for collective graphical models. In *International Conference on Artificial Intelligence and Statistics*, pages 685–693.

[79] Niepert, M. and Van den Broeck, G. (2014). Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *AAAI Conference on Artificial Intelligence*, pages 2467–2475.

[80] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge.

[81] Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6114–6124.

[82] Oliehoek, F. A., Vlassis, N. A., et al. (2007). Q-value heuristics for approximate solutions of dec-pomdps. In *AAAI Spring Symposium: Game Theoretic and Decision Theoretic Agents*, pages 31–37.

[83] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., and Vian, J. (2017). Deep decentralized multi-task multi-agent rl under partial observability. In *ICML*.

[84] Ooi, J. M. and Wornell, G. W. (1996). Decentralized control of a multiple access broadcast channel: Performance bounds. In *Decision and Control, 1996., Proceedings of the 35th IEEE Conference on*, volume 1, pages 293–298. IEEE.

[85] Pajarinen, J., Hottinen, A., and Peltonen, J. (2014). Optimizing spatial and temporal reuse in wireless networks by decentralized partially observable Markov decision processes. *IEEE Trans. on Mobile Computing*, 13(4):866–879.

[86] Palmer, G., Tuyls, K., Bloembergen, D., and Savani, R. (2017). Lenient multi-agent deep reinforcement learning. *arXiv preprint arXiv:1707.04402*.

[87] Panait, L., Sullivan, K., and Luke, S. (2006). Lenient learners in cooperative multiagent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 801–803. ACM.

[88] Papadimitriou, C. H. and Tsitsiklis, J. (1982). On the complexity of designing distributed protocols. *Information and Control*, 53(3):211–218.

[89] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers Inc.

[90] Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 489–496. Morgan Kaufmann Publishers Inc.

[91] Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

[92] Poole, D. (2003). First-order probabilistic inference. In *IJCAI*, volume 3, pages 985–991.

[93] Powell, W. and Godfrey, G. (2001). An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. *Management Science*, 47(8):1101–1112.

[94] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons.

[95] Qiu, L., Yang, Y. R., Zhang, Y., and Shenker, S. (2003). On selfish routing in internet-like environments. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 151–162. ACM.

[96] Radner, R. (1962). Team decision problems. *The Annals of Mathematical Statistics*, 33(3):857–881.

[97] Robbel, P., Oliehoek, F. A., and Kochenderfer, M. J. (2016). Exploiting anonymity in approximate linear programming: Scaling to large multiagent MDPs. In *AAAI Conference on Artificial Intelligence*, pages 2537–2543.

[98] Roughgarden, T. (2005). *Selfish routing and the price of anarchy*, volume 174. MIT press Cambridge.

[99] Sandholm, W. H. (2015). Population games and deterministic evolutionary dynamics. In *Handbook of game theory with economic applications*, volume 4, pages 703–778. Elsevier.

[100] Schneider, J., Wong, W.-K., Moore, A., and Riedmiller, M. (1999). Distributed value functions. *Robotics Institute*, page 264.

[101] Schulman, J., Heess, N., Weber, T., and Abbeel, P. (2015a). Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pages 3528–3536.

[102] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015b). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

[103] Schweitzer, F. (2007). Collective decisions in multi-agent systems. In *Advancing Social Simulation: The First World Congress*, pages 7–12. Springer.

[104] Seijen, H. V., van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184.

[105] Seuken, S. and Zilberstein, S. (2012). Improved memory-bounded dynamic programming for decentralized pomdps. *arXiv preprint arXiv:1206.5295*.

[106] Sheldon, D., Elmohamed, M. A. S., and Kozen, D. (2007). Collective inference on markov models for modeling bird migration. In *Advances in Neural Information Processing Systems*, pages 1321–1328.

[107] Sheldon, D., Sun, T., Kumar, A., and Dietterich, T. G. (2013). Approximate inference in collective graphical models. In *International Conference on Machine Learning*, pages 1004–1012.

[108] Sheldon, D. R. and Dietterich, T. G. (2011). Collective graphical models. In *Advances in Neural Information Processing Systems*, pages 1161–1169.

[109] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[110] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395.

[111] Singh, S., Kearns, M., and Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 541–548. Morgan Kaufmann Publishers Inc.

[112] Smith, M. J. (1979). The existence, uniqueness and stability of traffic equilibria. *Transportation Research Part B: Methodological*, 13(4):295–304.

[113] Sonu, E., Chen, Y., and Doshi, P. (2015). Individual planning in agent populations: Exploiting anonymity and frame-action hypergraphs. In *International Conference on Automated Planning and Scheduling*, pages 202–210.

[114] Sun, T., Sheldon, D., and Kumar, A. (2015). Message passing for collective graphical models. In *International Conference on Machine Learning*, pages 853–861.

[115] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

[116] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

[117] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[118] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *International Conference on Neural Information Processing Systems*, pages 1057–1063.

[119] Szer, D. and Charpillet, F. (2006). Point-based dynamic programming for dec-pomdps. In *AAAI*, volume 6, pages 1233–1238.

[120] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395.

[121] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337.

[122] Taylor, P. D. and Jonker, L. B. (1978). Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1-2):145–156.

[123] Toussaint, M., Harmeling, S., and Storkey, A. (2006). Probabilistic inference for solving (PO)MDPs. Technical report, University of Edinburgh, Edinburgh, UK.

[124] Tsitsiklis, J. and Athans, M. (1985). On the complexity of decentralized decision making and detection problems. *IEEE Transactions on Automatic Control*, 30(5):440–446.

[125] Tsitsiklis, J. N. et al. (1993). Decentralized detection. *Advances in Statistical Signal Processing*, 2(2):297–344.

[126] Tumer, K. and Agogino, A. (2007). Distributed agent-based air traffic flow management. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 255:1–255:8.

[127] Tumer, K. and Agogino, A. K. (2009). Multiagent learning for black box system reward functions. *Advances in Complex Systems*, 12(4-5):475–492.

[128] Tumer, K., Agogino, A. K., and Wolpert, D. H. (2002). Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 378–385. ACM.

[129] Van der Pol, E. and Oliehoek, F. A. (2016). Coordinated deep reinforcement learners for traffic light control. In *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*.

[130] van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. *arXiv preprint arXiv:1602.07714*.

[131] Varakantham, P., Adulyasak, Y., and Jaillet, P. (2014). Decentralized stochastic planning with anonymity in interactions. In *AAAI Conference on Artificial Intelligence*, pages 2505–2511.

[132] Varakantham, P., Kwak, J.-y., Taylor, M. E., Marecki, J., Scerri, P., and Tambe, M. (2009). Exploiting coordination locales in distributed pomdps via social model shaping. In *ICAPS*.

[133] Varakantham, P. R., Cheng, S.-F., Gordon, G., and Ahmed, A. (2012). Decision support for agent populations in uncertain and congested environments. In *AAAI Conference on Artificial Intelligence*, pages 1471–1477.

[134] Verweij, B., Ahmed, S., Kleywegt, A. J., Nemhauser, G., and Shapiro, A. (2003). The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimization and Applications*, 24(2-3):289–333.

[135] Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305.

[136] Wardrop, J. (1900). Some theoretical aspects of road traffic research. In *Inst Civil Engineers Proc London/UK/*.

[137] Weaver, L. and Tao, N. (2001). The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc.

[138] Weber, T., Heess, N., Eslami, A., Schulman, J., Wingate, D., and Silver, D. (2015). Reinforced variational inference. In *Advances in Neural Information Processing Systems (NIPS) Workshops*.

[139] Wei, E. and Luke, S. (2016). Lenient learning in independent-learner stochastic cooperative games. *The Journal of Machine Learning Research*, 17(1):2914–2955.

[140] Weintraub, G. Y., Benkard, C. L., and Van Roy, B. (2005). Oblivious equilibrium: A mean field approximation for large-scale dynamic games. In *NIPS*, pages 1489–1496.

[141] Wiedenbeck, B., Yang, F., and Wellman, M. P. (2018). A regression approach for modeling games with many symmetric players. In *(AAAI) Conference on Artificial Intelligence*. AAAI.

[142] Wiering, M. (2000). Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158.

[143] Williams, R. J. (1992a). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[144] Williams, R. J. (1992b). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.

[145] Winstein, K. and Balakrishnan, H. (2013). Tcp ex machina: Computer-generated congestion control. In *Proceedings of the ACM SIGCOMM 2013 Conference*, SIGCOMM '13, pages 123–134.

[146] Wiszniewska-Matyszkiel, A. (2014). Open and closed loop nash equilibria in games with a continuum of players. *Journal of Optimization Theory and Applications*, 160(1):280–301.

[147] Wolpert, D. H. and Tumer, K. (1999). An introduction to collective intelligence. *arXiv preprint cs/9908014*.

[148] Wolpert, D. H. and Tumer, K. (2001). Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(02n03):265–279.

[149] Wolpert, D. H. and Tumer, K. (2002). Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific.

[150] Wu, C., Rajeswaran, A., Duan, Y., Kumar, V., Bayen, A. M., Kakade, S., Mordatch, I., and Abbeel, P. (2018). Variance reduction for policy gradient with action-dependent factorized baselines. In *International Conference on Learning Representations*.

[151] Wu, F., Zilberstein, S., and Chen, X. (2010). Point-based policy generation for decentralized pomdps. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1307–1314. International Foundation for Autonomous Agents and Multiagent Systems.

[152] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. (2018a). Mean field multi-agent reinforcement learning. In *ICML*.

[153] Yang, Y., Yu, L., Bai, Y., Wang, J., Zhang, W., Wen, Y., and Yu, Y. (2018b). An empirical study of ai population dynamics with million-agent reinforcement learning. In *Proceedings of the 2018 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems.

[154] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312.

[155] Yuille, A. L. (2002). CCCP algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14(7):1691–1722.

[156] Yuille, A. L. and Rangarajan, A. (2001). The concave-convex procedure (CCCP). In *Advances in Neural Information Processing Systems*, pages 1033–1040.

[157] Zawadzki, E., Lipson, A., and Leyton-Brown, K. (2014). Empirically evaluating multiagent learning algorithms. *arXiv preprint arXiv:1401.8074*.

Appendix A

Domain description

A.1 Taxi fleet management

Taxi dataset: Our dataset contains the taxi trips of 8000 taxis in Singapore for 1 year. Each taxi call in the dataset is defined by the start time (1 in 48 periods), the origin zone and the destination zone (1 in 81 zones). Because the spatial and temporal dimensions have discrete values, the taxi calls are summarized by the taxi demand counts $d_t(i, j)$ to be the number of taxi passengers at time period t to go from a zone i to another zone j. From the dataset, we have the complete taxi demand tables $\langle d_t(i, j), \forall t \in [1, 48], i, j \in [1, 81]\rangle$ for 12 months. In simulation, we randomly choose a month data and extract the corresponding statistic demand table $\langle d_t(i, j), \forall t \in [1, 48], i, j \in [1, 81]\rangle$.

Autonomous taxi movement: We consider the planning problem of 8000 taxis over 48 half-hour periods. The goal is to learn a moving policy for taxis so that they can re-allocate themselves in each time period to match to the dynamic demands. The decision making and action execution of taxis in each time period t are as follows:

- At the beginning of the time period t, each taxi driver m in a zone z has the observation $o(z, d_t, \mathrm{n}_t^{\mathrm{s}})$ to be the number of taxis and passengers in that zone and/or in the surrounding zones.
- Based on the local observation, a taxi in a zone z decides to either stay in z or *relocate*

to one of neighboring zones $z' \in N(z)$.

- After deciding the *relocating* movement, all taxis concurrently execute their actions. As we consider half-hour period, we assume the taxis reach their destinations z' in the same time period t. In its destination zone $a_t^m = z'$ at time t, a taxi m will :
  - With the probability $\min\{\frac{d_t(z')}{\hat{\mathrm{n}}^{\mathrm{s}}(z')}, 1\}$ to pick a passenger and receive a payment, which is proportional to the ratio between the number of passengers $d_t(z')$ and the total number $\hat{\mathrm{n}}_t^{\mathrm{s}}(z') = \sum_z \mathrm{n}_t^{\mathrm{sa}}(z, z')$ of taxis moving into zone z'. If a taxi has passenger in z' , it will move to passenger destination zone z''. The distribution of passenger's destination is $\frac{d_t(z',z'')}{d_t(z')}$ which is the flow ratio of passengers from z' to z''. A taxi receives a payment $payment(z', z'')$ from its passenger in time t.
  - With the probability $1 - \min\{\frac{d_t(z')}{\hat{\mathrm{n}}^{\mathrm{s}}(z')}, 1\}$ to stay in z' until the beginning of the next time period $t+1$. A taxi without passenger does not receive any payment.

A.1.1 $\mathbb{C}$Dec-POMDP for taxi navigation problem

- $H = 48, M = 8000$.
- A set of action A for taxi m to be moving to either one of neighbouring zones $z' \in N(z)$ from its current zone z or staying in the current zone z.
- The local state of a taxi m is defined by its current zone z at beginning of each time period t.
- The global state component d_t at time t is the passenger demands over the whole city. It is determined also by a count table whose each component $d_t(z, z')$ is the number of passengers moving from zone z to z' in time period t. However, taxi can only observe the demand counts at zones level $d_t(z) = \sum_{z'} d_t(z, z')$ rather than the exact flows.
- The counts in each time t include:
  - $\mathrm{n}_t^{\mathrm{s}}(z) = \sum_{m=1}^{M} \mathbb{I}(s_t^m = z), \forall z \in Z$ to be the number of taxis in each zone z.

- $\mathrm{n}_t^{\mathrm{sa}}(z, z') = \sum_{m=1}^{M} \mathbb{I}(s_t^m = z, a_t^m = z'), \forall z, z' \in Z$ to be the number of taxis from zone z deciding to move to zone z'.
- $\mathrm{n}_t^{\mathrm{sas}'}(z, z', z'') = \sum_{m=1}^{M} \mathbb{I}(s_t^m = z, a_t^m = z', s_{t+1}^m = z''), \forall z, z', z'' \in Z$ to be the number of taxis moving from z to z' and being in zone z'' at the beginning of next time period.

- At decision state $s_t^m = z$, we assume a taxi agent m will have the observation about the demand counts $d_t(z') = \sum_{z''} d_t(z', z'')$ and taxi zone counts $\mathrm{n}_t^{\mathrm{s}}(z')$, $\forall z' \in \mathrm{Nb}(z) \cup \{z\}$ of the current zone z and its neighboring zones $\mathrm{Nb}(z)$.

- After deciding the next zone $a_t^m = z'$, a taxi agent will deterministically transit to z'. The taxi agent m will stay in z' until the next time period $t+1$ or move to another zone depending on whether it has passenger or not. We use the indicator variable $b_t^m = \{0; 1\}$ to denote whether a taxi m has passenger or not in time period t. The local transition of a taxi m is defined as:

$$P_l\big(s_{t+1}^m | s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t\big) = \sum_{b_t^m \in \{0;1\}} P(b_t^m | a_t^m, \mathrm{n}_t^{\mathrm{sa}}, d_t) P(s_{t+1}^m | a_t^m, b_t^m, d_t) \quad \text{(A.1)}$$

where

  - The probability of a taxi to have passenger is

$$P(b_t^m = 1 | a_t^m = z', \mathrm{n}_t^{\mathrm{sa}}, d_t) = \min\{\frac{d_t(z')}{\sum_z \mathrm{n}_t^{\mathrm{sa}}(z, z')}; 1\}$$

  - And taxi has no passenger with the probability

$$P(b_t^m = 0 | a_t^m = z', \mathrm{n}_t^{\mathrm{sa}}, d_t) = 1 - \min\{\frac{d_t(z')}{\sum_z \mathrm{n}_t^{\mathrm{sa}}(z, z')}; 1\}$$

– The local state of a taxi in next time period $t+1$ depends on its status:

$$P(s_{t+1}^m = z'' | a_t^m = z', b_t^m = 1, d_t) = \frac{d_t(z', z'')}{d_t(z')} \tag{A.2}$$

$$P(s_{t+1}^m = z'' | a_t^m = z', b_t^m = 0, d_t) = \begin{cases} 1 & \text{if } z' = z'' \\ 0 & \text{otherwise.} \end{cases} \tag{A.3}$$

$$\tag{A.4}$$

- Each agent m has a non-stationary policy $\pi_t^m(j|i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ denoting the probability of agent m to take action j given its observation $(i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ at time t. We denote the policy over planning horizon of an agent m to be $\pi^m = (\pi_1^m, \ldots, \pi_H^m)$. When agents have the same policy, we denote the common policy with π.

- A taxi agent moving from z to z' will incur a roaming cost $-cost(z, z')$. When a taxi has a passenger to travel from z' to z'', it will receive a trip payment $\bar{p}(z', z'')$. Hence, the local reward $r_t^m(s_t^m = z, a_t^m = z', \mathrm{n}_t^{\mathrm{sa}}, d_t)$ of a taxi moving from z to z' is

$$= -cost(z, z') + P(b_t^m = 1 | a_t^m = z', \mathrm{n}_t^{\mathrm{sa}}, d_t) \sum_{z''} \frac{d_t(z', z'')}{d_t(z')} \bar{p}(z', z'')$$

A.1.2 Local Reward Structure

When taxi m pick-ups a passenger in zone z at time t and move to the destination z', it receives a reward $\bar{p}(z, z')$ to be the profit of that trip. A taxi with no passenger incurs a roaming cost c. However, we notice that the roaming cost is similar among all zones, therefore, we shape the reward by ignoring the cost and consider only the reward $\bar{p}(z, z')$ for taxi agent with passenger. The total local reward of all taxis getting passengers from a zone z at the time period t is

$$R_t^{\text{trip}}(z, \mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{z \in Z} \hat{\mathrm{n}}_t^{\mathrm{s}}(z) \min\{\frac{d_t(z)}{\hat{\mathrm{n}}_t^{\mathrm{s}}(z)}; 1\} \sum_{z' \in Z} \frac{d_t(z, z')}{d_t(z)} \bar{p}(z, z') \tag{A.5}$$

Recall that in the above formula (A.5), $\hat{\mathrm{n}}_t^{\mathrm{s}}(z) = \sum_{z'} \mathrm{n}_t^{\mathrm{sa}}(z', z)$ is the incoming taxi flow into zone z, $\min\{\frac{d_t(z)}{\hat{\mathrm{n}}_t^{\mathrm{s}}(z)}; 1\}$ is the probability a taxi picking-up passenger in zone z and

$\sum_{z' \in Z} \frac{d_t(z,z')}{d_t(z)} \bar{p}(z,z')$ is the average reward of a taxi with passenger in zone z.

To maximize the total local reward $\mathbb{E}\big[\sum_{t=1}^{H} \sum_z R_t^{\text{trip}}(z, \text{n}_t^{\text{sa}}, d_t)\big]$, the policy should balance the movement of taxis with the expected demand in each city zone at different time periods. If more taxis are present in a zone than the aggregate demand in that zone, then unhired taxis incur loss of revenue. Therefore, a good policy would direct taxis to different city zones to match demand with supply.

A.1.3 Global Reward Structure

We can extend the reward structure in taxi domain to consider the QoS component for a zone z as follows:

$$R_t^{\text{QoS}}(z, \text{n}_t^{\text{sa}}, d_t) = w_z \min\{0, \hat{\text{n}}_t^{\text{s}}(z) - \alpha \cdot d_t(z)\} \tag{A.6}$$

in which $\alpha \in [0; 1]$ specifies the percentage of demand $d_t(z)$ we target to serve. $\hat{\text{n}}_t^{\text{s}}(z) = \sum_{z'} \text{n}_t^{\text{sa}}(z', z)$ is the total number of taxis moving in zone z at time t. The $\min$ operator ensures that the penalty function is activated only when the number of available taxi $\text{n}_t^{\text{s}}(z)$ at z below the target. The function is ignorable when $w_z = 0$, and it is emphasized when w_z increases.

The objective function in taxi domain under this QoS components is

$$\begin{aligned}
&\mathbb{E}\Big[\sum_{t=1}^{H} \sum_z \big[R_t^{\text{trip}}(z, \text{n}_t^{\text{sa}}, d_t) + R_t^{\text{QoS}}(z, \text{n}_t^{\text{sa}}, d_t)\big]\Big] \\
&= \mathbb{E}\Big[\sum_{t=1}^{H} \sum_{z \in Z} \Big(w_z \min\{0, \hat{\text{n}}_t^{\text{s}}(z) - \alpha \cdot d_t(z)\} + \hat{\text{n}}_t^{\text{s}}(z) \min\{\frac{d_t(z)}{\hat{\text{n}}_t^{\text{s}}(z)}; 1\} \sum_{z' \in Z} \frac{d_t(z,z')}{d_t(z)} \bar{p}(z,z')\Big)\Big].
\end{aligned} \tag{A.7}$$

A.2 Robot Grid Navigation

For a grid of size $L \times L$ with M agent, we consider the probability of an agent to firstly appear in one of the edge locations $\langle (0, l), \forall l \in [0; L-1] \rangle$ to be 0.1. An agent receives a penalty -1 when the number of agents crossing the same corridor is greater than the corridor

capacity ($= 2$). When the total number of agents simultaneously crossing an edge is less than its capacity, then each agent has a higher probability of moving to the next location (=0.8); this probability decreases sharply if total agents crossing the edge are more than the capacity. When an agent reaches the goal $(L-1, L-1)$, it receives the reward 1 and reset back uniformly to one of edge locations. The $\mathbb{C}$Dec-POMDP for Robot Grid Navigation is defined as follows:

- The local state s_t^m of an agent m is defined by its current location i in a grid $L \times L$.
- An agent m can select its action from either staying in the same location i or taking one of four directional movements (left, right, up, down). Therefore, the local action space is $A = \{\text{stay, left, right, up, down}\}$.
- The sufficient statistics counts in each time t include:
  - $\mathrm{n}_t^{\mathrm{s}}(i) = \sum_{m=1}^{M} \mathbb{I}(s_t^m = i), \forall z \in Z$ to be the number of agents in a location i.
  - $\mathrm{n}_t^{\mathrm{sa}}(i,j) = \sum_{m=1}^{M} \mathbb{I}(s_t^m = i, a_t(m) = j), \forall i \in S, j \in A$ to be the number of agents from i deciding to taking action j.
  - $\mathrm{n}_t^{\mathrm{sas}}(i,j,i') = \sum_{m=1}^{M} \mathbb{I}(s_t^m = i, a_t(m) = j, s_{t+1}^m(m) = i'), \forall i, i' \in S, j \in A$ to be the number of agents take from i taking action j and arriving i' in the next time step.
- An agent m will have the observation about zone counts $\mathrm{n}_t^{\mathrm{s}}(i'), \forall i' \in \mathrm{Nb}(i) \cup \{i\}$ of the current zone i and its neighboring zones $\mathrm{Nb}(i)$.
- We denote $dest(i,j)$ to be the heading zone of an agent in location i taking movement j. The transition function $P(s_{t+1}^m | s_t^m, a_t^m, \mathrm{n}_t^{\mathrm{sa}})$ is defined as follows:

$$P(dest(i,j)|i,j,\mathrm{n}_t^{\mathrm{sa}}(i,j)) = \begin{cases} 0.8 & \text{if } \mathrm{n}_t^{\mathrm{sa}}(i,j) \leq 2 \wedge i \neq (L-1,L-1) \\ 0.8 \times \frac{2}{\mathrm{n}_t^{\mathrm{sa}}(i,j)} & \text{if } \mathrm{n}_t^{\mathrm{sa}}(i,j) > 2 \wedge i \neq (L-1,L-1) \end{cases} \tag{A.8}$$

And the probability that agent fails to arrive its intended location and stay in the current location is

$$P(i|i,j,\mathrm{n}_t^{\mathrm{sa}}(i,j)) = 1 - P(dest(i,j)|i,j,\mathrm{n}_t^{\mathrm{sa}}(i,j)) \text{ if } i \neq (L-1,L-1).$$

When an agent reach the goal location $i_{goal} = (L-1, L-1)$, it is reset to one of edge location by

$$P(l|i, j, \mathrm{n}_t^{\mathrm{sa}}(i,j)) = \frac{1}{L} \text{ if } i = (L-1, L-1) \wedge l \in \langle (0, l), \forall l \in [0; L-1] \rangle$$

- The reward function is defined as follows:

$$r_t(i, j, \mathrm{n}_t^{\mathrm{sa}}(i,j)) = \begin{cases} 0 & \text{if } \mathrm{n}_t^{\mathrm{sa}}(i,j) \leq 2 \wedge i \neq (L-1, L-1) \\ -1 & \text{if } \mathrm{n}_t^{\mathrm{sa}}(i,j) > 2 \wedge i \neq (L-1, L-1) \\ 1 & \text{otherwise.} \end{cases} \tag{A.9}$$

A.3 Synthetic Robot Patrolling Game

For a grid of size $L \times L$ with N agent, we consider the initial location of all agents to be at the center of the grid $(\lfloor L/2 \rfloor, \lfloor L/2 \rfloor)$. Each time, there is exactly one target available. The fixed location of each target is generated uniformly in the grid. When the target is reached by at least 1 robot, the whole team receives a reward value to be 1, the old target disappears and the new target is generated. Notice that the team would receive reward 1 even though there is more than 1 robot that reaches the target, hence the reward is nondecomposable among the agents.

Each agent can observe other agents and target in its current location and its adjacent locations. In each time step, the agent can choose to deterministically move to one of adjacent locations or to stay in its current location. The state, action space and observation function of synthetic patrolling game is similar to robot navigation domain. We define additional components in $\mathbb{C}$Dec-POMDP model for robot patrolling game as follows:

- The global state $d_t \in L \times L$ is the location of the current target.
- The transition is deterministic $P(dest(i,j)|i,j) = 1$.
- When target is reached by at least one agent $\sum_i \mathrm{n}_t^{\mathrm{sa}}(i, d_t) > 0$, the target's location d_{t+1} is uniformly reset to one of grid cell as $P(d_{t+1} = i' | \sum_i \mathrm{n}_t^{\mathrm{sa}}(i, d_t) > 0) = \frac{1}{L \times L}$. Otherwise, $P(d_{t+1} = d_t | \sum_i \mathrm{n}_t^{\mathrm{sa}}(i, d_t) = 0) = 1$.

- The global reward function $r_t(\mathrm{n}_t^{\mathrm{sa}}, d_t) = 1$ if $\sum_i \mathrm{n}_t^{\mathrm{sa}}(i, d_t) > 0$ and $r_t(\mathrm{n}_t^{\mathrm{sa}}) = 0$ otherwise.

A.4 Real World Police Patrolling

We consider a police division in an Asian city with 24 sectors and 16 police vehicles. When an incident happens in a sector, a command center would assign the nearest police to the incident. Incident features include urgency indicator, service time required, incident location and incident time. After a police is assigned to an incident, the remaining police would autonomously determine whether to stay in the current sector or re-allocate to a neighboring sector. We do not consider the reallocation at every time step but the reallocation only happens when an incident happens. This re-allocating decision of police is given by the neural network policy function π. An action of police, either attending to an incident or re-allocating to another sector, could take more than one time step. To model this, we extend the local state $s^m = i$, originally to be the current sector i of police agent, into $s^m = \langle i, c \rangle$ including the time c for the police agent m returns the base sector i if it is still executing an action. We consider the simplest case where the travel time between sectors is pre-computed by a map service and used in the simulation as deterministic travel time.

The state counts are also extended into spatial-temporal dimensions accordingly. The spatial-temporal state counts at time t are $\langle n_t(i, c) \rangle_{i,c}$, in which each $\langle n_t(i, c) \rangle_{i,c}$ is the number of police agents completing their current actions and station in sector i in c time periods ahead from t. We consider the local observation of an agent to be the spatial-temporal state counts of its current sector and neighboring sectors. An example of the extended state count for a sector i is given by Table A.1. Based on this state count table, there is no police current available to patrol in sector i, but there is a police arriving this sector at the next time period and there is a police arriving this sector at the next 3 period. Notice that this table only summarizes based on incidents happening by time t. It will be changed when there is a new incident happening in time $t' \geq t + 1$.

We discretize the planning horizon into 5-minute periods and consider the response time threshold for urgent incident to be 10 minutes (within 2 time periods) and for non-urgent to

c	0	1	2	3	4
$\mathrm{n}_t(i,c)$	0	1	0	1	0

Table A.1: Example of temporal state count for a sector i

be 20 minutes (within 4 time periods). When the real response time exceeds the threshold, the *whole* team receives a *global* reward -10, otherwise it is 0.

The $\mathbb{C}$Dec-POMDP model for police patrolling domain is defined as follows:

- The local state $s_t^m = \langle l_t^m, c_t^m \rangle$ of agent m is determined by its current base sector l_t^m and c_t^m is the time it takes for agent m to be available at l_t^m. When m is not assigned any incident, $c_t^m = 0$.
- The global component $d_t = \langle d_t^s, d_t^u, d_t^\Delta \rangle$ is the incident call appearing in time t, which is defined by incident location d_t^s, urgency $d_t^u \in \{0; 1\}$ and engagement duration $d_t^\Delta \in \mathbb{N}$. If there is no new incident in time time t, $d_t = \emptyset$.
- A set of action A for police m to be moving to either one of neighbouring sectors $i' \in \mathrm{Nb}(l_t^m)$ from its current sector l_t^m or staying in the current sector l_t^m. An agent only makes relocating action after it is back to its current base sector. When the agent is busy in an incident, it can not make relocating action, hence we also include the null action $a_t^m = \emptyset$.
- The counts in police patrolling domain include:
  - $\mathrm{n}_t^{\mathrm{s}}(i,c) = \sum_m \mathbb{I}_t^m(l_t^m = i, c_t^m = c)$ to be the number of police becoming free in sector i in c time periods.
  - $\mathrm{n}_t^{\mathrm{sa}}(i, c=0, j) = \sum_m \mathbb{I}_t^m(l_t^m = i, c_t^m = c = 0, a_t^m = j)$ to be the number of free police deciding to re-allocate from sector i to a new sector j.
- An agent m will have the observation about temporal-spatial counts $\mathrm{n}_t^{\mathrm{s}}(i', c)$, $\forall i' \in \mathrm{Nb}(l_t^m) \cup \{l_t^m\}$ of the current zone l_t^m and its neighboring zones $\mathrm{Nb}(l_t^m)$. In additional, all agents are informed about whether there is a new incident $d_t \neq \emptyset$ appearing at time t.

- We consider that an agent only makes decision on relocation when there is a new incident happening, therefore the individual policy function is

$$\pi_t(a_t^m|s_t^m = \langle i, c\rangle, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) = \begin{cases} \tilde{\pi}(a_t^m = i'|s_t^m = i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t)) & \text{if } d_t \neq \emptyset \cap c = 0 \\ \mathbb{I}(a_t^m = \emptyset) & \text{otherwise.} \end{cases} \tag{A.10}$$

in which $\tilde{\pi}(a_t^m = i'|s_t^m = i, o(i, \mathrm{n}_t^{\mathrm{s}}, d_t))$ is a decision function (soft-max output of an neural network) called only when there is new incident happening and the agent is already at the base sector.

- The transition of an agent from time t to $t+1$ depends on whether there is a new incident appear in time $t+1$ (then an agent would be assigned to attend that incident). Denote $\delta(i, i')$ to be the travel time between 2 sectors i and i'. To define the transition function $P\Big(s_{t+1}^m = \langle i', c'\rangle | s_t^m = \langle i, c\rangle, a_t^m\Big)$, we distinguish 2 cases (with new incident and without new incident):

  - When there is no new incident at time $t+1$:

    - For busy policy agent with $c_t^m > 0$, it can not re-allocate, therefore, its transition is determined by

$$P\big(s_{t+1}^m = \langle l_{t+1}^m = l_t^m, c_{t+1}^m = c_t^m - 1\rangle | s_t^m = \langle l_t^m, c_t^m\rangle, a_t^m = \emptyset, d_{t+1} = \emptyset\big) = 1 \tag{A.11}$$

    - For free police agent with $c_t^m = 0$, after deciding to re-allocate to a new sector j, its transition is determined by

$$P\big(s_{t+1}^m = \langle l_{t+1}^m = j, c_{t+1}^m = \delta(i, j) - 1\rangle | s_t^m = \langle i, 0\rangle, a_t^m = j, d_{t+1} = \emptyset\big) = 1 \tag{A.12}$$

  - When there is a new incident $d_{t+1} = \langle d_{t+1}^s, d_{t+1}^u, d_{t+1}^\Delta\rangle$ in time $t+1$, it will be assigned to the nearest sector $\varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}) = \arg\min_{i'} \big[\delta(i', d_{t+1}^s) + \min_{c'|(n_t(i',c')>0)}\big]$, in which $\min_{c'|(n_t(i',c')>0)}$ is the earliest time there is a free vehicle in sector i'. We denote $\bar{\delta}(i, d_{t+1})$ to be the total time for an agent in a

sector i to complete incident d_{t+1} and come back to its base. The transition of earliest available agents $s_t^m = \langle l_t^m = \varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}), c_t^m = \min_{c' | (n_t(l_t^m, c') > 0)} \rangle$ in the assigned sector $\varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}})$ is defined as:

* If this agent is an busy agent $c_t^m > 0$, it could be assigned to the incident d_{t+1} with probability $\frac{1}{\mathrm{n}_t^{\mathrm{s}}(i,c)}$. Therefore its transition is

$$P(s_{t+1}^m = \langle l_{t+1}^m = l_t^m, c_t^m - 1 + \bar{\delta}(l_t^m, d_{t+1}) \rangle | s_t^m = \langle l_t^m, c_t^m \rangle, a_t^m, d_{t+1})$$
$$= \frac{1}{\mathrm{n}_t^{\mathrm{s}}(l_t^m, c_t^m)} \text{ with } l_t^m = \varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}), c_t^m = \min_{c' | (n_t(l_t^m, c') > 0)} \tag{A.13}$$

and

$$P(s_{t+1}^m = \langle l_{t+1}^m = l_t^m, c_t^m - 1 \rangle | s_t^m = \langle l_t^m, c_t^m \rangle, a_t^m, d_{t+1})$$
$$= 1 - \frac{1}{\mathrm{n}_t^{\mathrm{s}}(l_t^m, c_t^m)} \text{ with } l_t^m = \varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}), c_t^m = \min_{c' | (n_t(l_t^m, c') > 0)} \tag{A.14}$$

in which (A.13) is the probability for this agent to be assigned to the incident and (A.14) is the probability for this agent to be not assigned to the incident.

* Similarly, the transition of a free agent $c_t^m = 0$ in assigned sector is defined as

$$P\big(\langle l_{t+1}^m = j, c_{t+1}^m = \delta(l_t^m, j) + \bar{\delta}(j, d_{t+1}) - 1 \rangle | \langle l_t^m, c_t^m \rangle, a_t^m = j, d_{t+1}\big)$$
$$= \frac{1}{\mathrm{n}_t^{\mathrm{s}}(l_t^m, c_t^m)} \text{ with } l_t^m = \varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}), c_t^m = 0 \tag{A.15}$$
$$P\big(s_{t+1}^m = \langle l_{t+1}^m = j, c_{t+1}^m = \delta(l_t^m, j) - 1 \rangle | \langle l_t^m, c_t^m \rangle, a_t^m = j\big)$$
$$= 1 - \frac{1}{\mathrm{n}_t^{\mathrm{s}}(l_t^m, c_t^m)} \text{ with } l_t^m = \varkappa(d_{t+1}, \mathrm{n}_t^{\mathrm{s}}), c_t^m = 0 \tag{A.16}$$

in which (A.15) is the probability for this agent to attend incident d_{t+1} after arriving the new sector j and (A.16) is the probability that this agent is not assigned to incident d_{t+1}.

For other agents who are not the earliest available ones in the assigned sector, their transition function is similar to the case of no new incident.

- The global reward function is determined by the waiting time $\delta(\varkappa(d_t, \mathrm{n}_t^{\mathrm{s}}), d_t^s)$ for

incident d_t to be attended as follows:

$$r_t(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \begin{cases} 0 & \text{if } \delta(\varkappa(d_t, \mathrm{n}_t^{\mathrm{s}}), d_t^s) \leq QoS(d_t^u) \\ -1 & \text{otherwise.} \end{cases} \tag{A.17}$$

in which $QoS(d_t^u) = 10$ if the incident is urgent ($d_t^u = 1$) and $QoS(d_t^u) = 20$ otherwise.

Appendix B

Neural network design

B.1 Hyper-parameters

To optimize the policy and value function network, we use Adam optimizer with the learning rate chosen from $\{10^{-5}, 10^{-4}, 10^{-3}\}$ for the best performance of algorithms. As observation of the count can have different magnitude in grid navigation and taxi domain, we use layer normalization [7] for all the networks. To address the different magnitude of rewards, i.e. the grid navigation having maximum reward 1 and taxi domain having maximum reward 100, we normalize the advantage value of $f(i, j, d_t, \mathrm{n}_t^{\mathrm{s}})$ by adaptively rescaling targets method as in [130] before feeding them into the policy gradient computation.

For actor-critic update, we consider the batch size to be 100 for synthetic robot navigation and 48 for taxi navigation. For the police patrolling, we consider the batch to be whole incidents in one day.

B.2 Network structure

In all of our neural networks, we use relu unit for all hidden layers and softmax unit for output of policy and linear output for value function.

Policy networks: are dense neural networks with the hidden size (18×18) for the taxi domain and grid navigation. In patrolling domains, we consider the network of size (32×32).

B.2.1 Factored value function for local rewards

In Chapter 5, we consider the factorization form of the critic in the form $\tilde{Q}(\mathrm{n}_t^{\mathrm{sa}}, d_t) = \sum_{i,j} \mathrm{n}_t^{\mathrm{sa}}(i,j) f(i,j,\mathrm{n}_t^{\mathrm{s}}, d_t)$ to optimize individual policy with a local reward. Because we expect the value of each state to be different, we construct a neural network $f(i, \bullet)$ for each state i. We consider each neural network $f(i, \bullet)$ to have a similar structure to the policy network, i.e. dense-net with hidden layers to be (18×18) or (32×32) depending on the studied domain. Theoretically $f(i, \bullet)$ can take the input of the complete state counts $\mathrm{n}_t^{\mathrm{s}}$, however the complete state counts contain redundant information which causes noise in the local value function estimation. Therefore, we consider the input of $f(i, \bullet)$ to be the partial observation (of relevant neighboring locations) similar as input of policy network. $f(i, \bullet)$ outputs the values of all actions $j \in A$ available to take in state i.

B.2.2 Value function for global rewards

The non-decomposable critic function is designed for MCAC, CCAC and AC algorithms in Chapter 6. Motivated by recent advances in combining model-based RL and model-free RL in value network [81, 51], we design global critic function to consist of different components to predict both immediate reward and accumulative value. The network is also designed based domain knowledge. In particular, in both grid patrolling and taxi domain, we use a deterministic transition function $f^{trans}(\mathrm{n}_t^{\mathrm{sa}})$ to estimate the next state count $\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}$ by the incoming flow to each location from its neighboring locations. Using $\hat{\mathrm{n}}_{t+1}^{\mathrm{s}} = f^{trans}(\mathrm{n}_t^{\mathrm{sa}})$, we design the the critic network having the form

$$\tilde{Q}_w(\mathrm{n}_t^{\mathrm{sa}}, d_t) = f_w^{reward}(\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}) + f_w^{value}(\hat{\mathrm{n}}_{t+1}^{\mathrm{s}})$$

In taxi domain: We refer to the taxi setting described in Section A.1. The neural

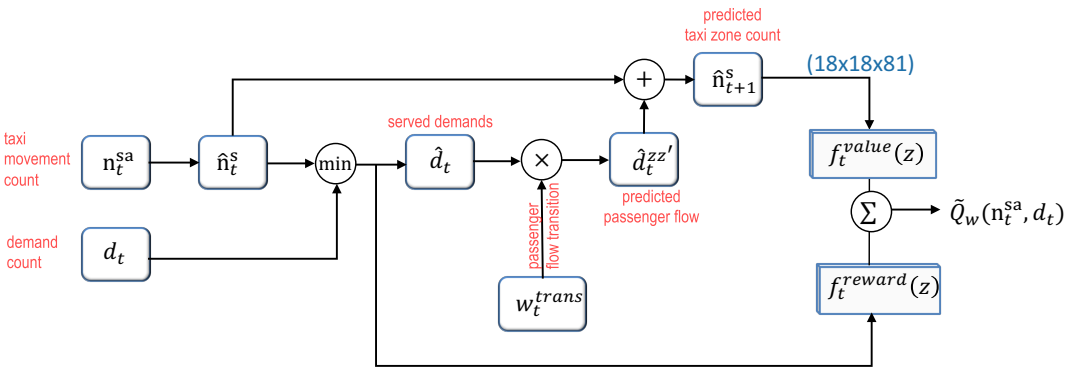


Figure B.1: Neural Network Architecture for Taxi Problem

network architecture for critic function in taxi domain is demonstrated by Figure B.1. Given the state, state-action count $\mathrm{n}_t^{\mathrm{sa}}$ and the demand counts at zone level d_t, we firstly compute the incoming state state $\hat{\mathrm{n}}_t^{\mathrm{s}}(z) = \sum_{z'} \mathrm{n}_t^{\mathrm{sa}}(z', z), \forall z$. Then we compute the counts of served demands $\hat{d}_t(z) = \min\{d_t(z), \hat{\mathrm{n}}_t^{\mathrm{s}}(z)\}, \forall z$. The *predicted* passenger flows $\hat{d}_t^{zz'}(z, z') = \hat{d}_t(z) \times w_t^{trans}(z, z')$ are obtained by the weight w_t^{trans} as output of a softmax function (to ensure $\sum_{z'} w_t^{trans}(z, z') = 1$). The *predicted* next state counts are computed as

$$\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}(z) = \hat{\mathrm{n}}_t^{\mathrm{s}}(z) - \sum_{z'} \hat{d}_t^{zz'}(z, z') + \sum_{z'} \hat{d}_t^{zz'}(z'), \forall z.$$

Notice that $\hat{d}^{zz'}$ provides estimated values for the transition count of taxis with passengers. $\hat{\mathrm{n}}_t^{\mathrm{s}}(z) - \sum_{z'} \hat{d}_t^{zz'}(z, z')$ provides estimated value for the transition count of taxis without passenger.

Given the demand count $\boldsymbol{d}_t$, the immediate reward component in the critic is defined as

$$f_w^{reward}(\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}) = \sum_z [\bar{p}_z \min(\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}(z), d_t(z)) + w_z \min\{0, \hat{\mathrm{n}}_{t+1}^{\mathrm{s}}(z) - \alpha d_t(z)\}]$$

in which $\bar{p}_z$ is learnable parameter corresponding to the average trip payment of zone z and $d_t(i)$ is corresponding to the total demands in zone i at time t.

The value predictor is $f_w^{value}(\hat{\mathrm{n}}_{t+1}^{\mathrm{s}}) = (H - t) \sum_z f_w^{value}(z, \hat{\mathrm{n}}_{t+1}^{\mathrm{s}})$ with $(H - t)$ to be remaining time and each $f_w^{value}(z, \hat{\mathrm{n}}_{t+1}^{\mathrm{s}})$ to be dense neural network with (18×18) hidden units to estimate the average (over remaining periods) reward collected in zone z given the

predicted state count $\hat{\mathrm{n}}^{\mathrm{s}}_{t+1}$. Each $f_w^{value}(z, \hat{\mathrm{n}}^{\mathrm{s}}_{t+1})$ is trained by rewards at zone z as follows:

$$\min \sum_t ||(H-t) f_w^{value}(z, \hat{\mathrm{n}}^{\mathrm{s}}_{t+1}) - \sum_{t'=t:H} \Big(\sum_{dest \in Z} \frac{d_t(z, dest)}{d_t(z)} [r_t(z, dest) + c] \min(\mathrm{n}^{\mathrm{s}}_{t+1}(z), d_t(z)) + w_z \min\{0, \mathrm{n}^{\mathrm{s}}_{t'}(z) - \alpha \cdot d_{t'}(z)\} \Big) ||^2 \quad \text{(B.1)}$$

In police patrolling domain: The predicted state counts are deterministically computed as

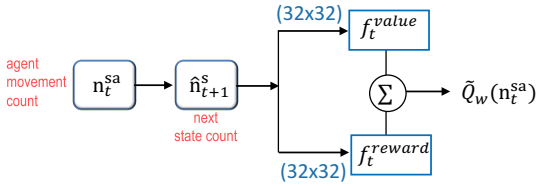


Figure B.2: Neural Network Architecture for Patrolling Problem

by the incoming flows $\hat{\mathrm{n}}^{\mathrm{s}}_{t+1}(z) = \sum_{z'} \mathrm{n}^{\mathrm{sa}}_t(z', z), \forall z$.

We design f_w^{reward} and f_w^{value} to be dense neural networks with hidden size (32×32). f_w^{value} is trained directly by the global empirical returns.

In synthetic robot patrolling domain: We encode location of target at time t by one-hot $v_t(i)$ and define the immediate value function component as $f_w^{reward}(\hat{\mathrm{n}}^{\mathrm{s}}_{t+1}) = \sum_i \min(\hat{\mathrm{n}}^{\mathrm{s}}_{t+1}(i), v_t(i))$. The value predictor f_w^{value} is a neural network with (32×32) hidden units. f_w^{value} is trained directly by the global empirical returns.