7-2023

Reinforcement learning for sequential decision making with constraints

Jiajing LING
Singapore Management University, jjling.2018@phdcs.smu.edu.sg

REINFORCEMENT LEARNING FOR SEQUENTIAL DECISION MAKING WITH CONSTRAINTS

LING JIAJING

SINGAPORE MANAGEMENT UNIVERSITY

2023

Reinforcement Learning for Sequential Decision Making with Constraints

LING Jiajing

Submitted to School of Computing and Information Systems
in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in Computer Science

Dissertation Committee:

SINGAPORE MANAGEMENT UNIVERSITY
2023

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.

Jiajing

Ling Jiajing
20 July 2023

Reinforcement Learning for Sequential Decision Making with Constraints

LING Jiajing

Abstract

Reinforcement learning is a widely used approach to tackle problems in sequential decision making where an agent learns from rewards or penalties. However, in decision-making problems that involve safety or limited resources, the agent's exploration is often limited by constraints. To model such problems, constrained Markov decision processes and constrained decentralized partially observable Markov decision processes have been proposed for single-agent and multi-agent settings, respectively. A significant challenge in solving constrained Dec-POMDP is determining the contribution of each agent to the primary objective and constraint violations. To address this issue, we propose a fictitious play-based method that uses Lagrangian Relaxation to perform credit assignment for both primary objectives and constraints in large-scale multi-agent systems. Another major challenge in solving both CMDP and constrained Dec-POMDP is the sample inefficiency issue, mainly resulting from finding valid actions that satisfy all constraints, which becomes even more difficult in large state and action spaces. Recent works in RL have attempted to incorporate domain knowledge from experts into the learning process through neuro-symbolic methods to address the sample inefficiency issue. We propose a knowledge compilation framework using decision diagrams by treating constraints as domain knowledge and introducing neuro-symbolic methods to support effective learning in constrained RL. Firstly, we propose a zone-based multi-agent pathfinding (ZBPF) framework that is motivated by drone delivery applications. We propose a neuro-symbolic method to efficiently solve the ZBPF problem with several domain constraints, such as simple path constraint and landmark constraint in ZBPF. Secondly, we propose another neuro-symbolic method to solve action constrained RL where the action space is discrete and combinatorial. Empirical results show that our proposed approaches achieve better performance than standard constrained RL algorithms in several real-world applications.

Contents

Acknowledgments

I still remember the moment when I received the offer from SMU to pursue a Ph.D. degree, and the excitement it brought. The five-year journey at SMU has been both challenging and rewarding. Throughout this process of learning and research, I have interacted with numerous individuals who have significantly impacted my academic and personal growth.

First and foremost, I express my heartfelt gratitude to my supervisor, Prof. Akshat Kumar, who has been an exceptional researcher and mentor. His guidance and suggestions have shaped my research direction and career aspirations. We have explored various research problems and generated novel ideas. I am deeply inspired by his approach to research.

I extend my sincere thanks to all my research collaborators and friends, including Tarun Gupta, Kushagra Chandak, Arambam James Singh, Nguyen Duc Thien, Moritz Lukas Schuler, Janaka Chathuranga Brahmanage, Chaithanya Basrur, and Chen Changyu. Working with each of them was a great experience, and their contributions have influenced my research journey.

I am also grateful to my committee members, Prof. Pradeep Varakantham, Prof. Mai Anh Tien, and Prof. Arunesh Sinha for offering their time and providing constructive feedback that has played an important role in refining my work.

Lastly, I would like to express my gratitude to myself for the effort, perseverance, and resilience that I have demonstrated throughout this Ph.D. journey. I reflect on the initial stress and the countless nights spent in the lab, as well as the challenging period during the COVID-19 pandemic when I faced isolation. I am grateful for the personal growth that has emerged from them.

Chapter 1

Introduction

In modern artificial intelligence, sequential decision making which refers to the process of making decisions in a sequential manner is an important and active research field, and is widely used in AI applications such as robotics [37, 62], Atari games [65, 97], and autonomous systems [89]. In sequential decision making, since the outcome of each decision affects the available options of the subsequent decisions, one common approach to solving it is reinforcement learning (RL) [103]. In RL, an agent interacts with an environment to receive positive feedback (rewards) or negative feedback (penalties), which reflect the quality of the agent's decisions. A policy is learned using the feedback, and agent is able to make better decisions over time. RL problems can be formalized using a mathematical framework known as a Markov decision process (MDP) [103]. An MDP consists of a set of states, a set of actions that can be taken in each state, a reward function that describes the goodness of each action taken in each state, and a transition function that describes the probability of transitioning to a new state when an action is taken in the current state [7]. Mathematically, the goal of the RL agent is to learn a policy that maximizes the cumulative reward over time. This goal motivates the agent to explore as many states and actions as possible in the environment.

However, in many safety-critical or resource-related decision making problems, the agent's exploration in the environment is typically restricted by constraints. For example, to ensure robots safety when they are conducting tasks, robots' motion must be restricted in a certain range [29, 84], and their energy level cannot drop below a specified level [62]. Another example is resource allocation in supply-demand matching [11, 99]. The allocation of resources must satisfy some constraints (e.g., total assigned resources should be within a limit). To tackle sequential decision making problems with constraints, researchers have proposed constrained Markov decision processes (CMDP) [2] as an essential model. CMDPs can potentially be solved using constrained reinforcement learning (RL). In this paradigm, the objective of the agent is to find a policy that maximizes the expected cumulative reward while adhering to constraints defined in the CMDP.

One of the primary challenges in solving a CMDP using RL is finding actions that satisfy all constraints through trial and error. Therefore, this method can suffer from severe sample inefficiency due to the constraints, especially in large state and action spaces [35]. Sample inefficiency is also a significant challenge in standard RL. To address this problem, several recent works have attempted to incorporate domain knowledge from experts into the learning process [57, 72, 118]. These approaches, called neuro-symbolic methods, combine the complementary strengths of neural and symbolic approaches. The neural side provides trainability from data and function approximation capabilities, while the symbolic side provides knowledge representation and reasoning abilities. By combining these strengths, neuro-symbolic methods show great potential for overcoming the sample inefficiency issue in RL. Notably, the neuro-symbolic methods offer a promising new direction for solving constrained RL, as the constraints defined in a CMDP can be treated as domain knowledge.

In this thesis, we study how to address sequential decision making problem with constraints in both single-agent and multi-agent settings using RL, and introduce novel neuro-symbolic methods to support effective learning in constrained RL. In fact, we show that neuro-symbolic methods for constrained RL can achieve better performance than standard constrained RL algorithms and is a promising research direction.

The introduction is structured as follows. Firstly, in Section 1.1, we provide several examples that motivate the problem we aim to tackle in this thesis. Following this, in Section 1.2, we introduce two formulations for the modelling of the sequential decision making problem with constraints in single-agent and multi-agent settings respectively. In Section 1.3, we present related works that have been published on constrained RL and neural-symbolic methods for RL. Our contributions to this field are outlined in

Section 1.4. Finally, we provide a roadmap for the rest of the thesis in Section 1.5, and a list of all publications that contribute to this thesis in Section 1.6.

1.1 Motivating examples

The sequential decision making problem with constraints arises in numerous real-world applications, ranging from single-agent to multi-agent settings. In the following section, we provide several examples of such applications.

Drone Delivery The logistics industry has witnessed a promising potential in air-delivery by drones, as demonstrated in Figure 1.1 [81]. In this example, the airspace is divided into multiple blocks, and there are hundreds of drones navigating different air blocks to deliver items from respective sources to destinations. This delivery system has three constraints to consider. The first constraint is the capacity of each air block, ensuring the safety of the vehicles and avoiding congestion within the air block. The second constraint is the landmark constraint, requiring drones to visit a set of specific delivery locations to accomplish the delivery task. The last constraint is the simple path constraint, prohibiting loops in the planned path for each drone during the delivery task. The decision making problem is to find a policy to coordinate the movements of the drones so that the delivery task can be finished as quickly as possible while satisfying these constraints.

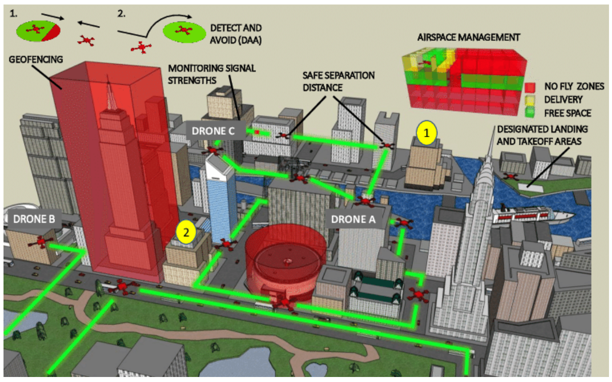
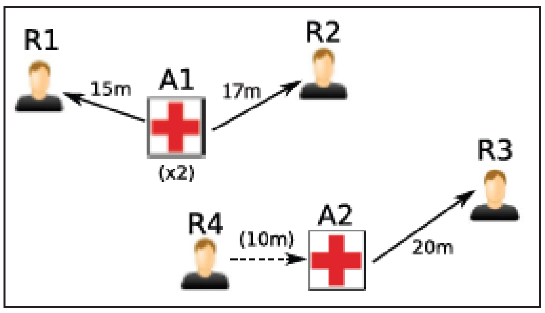
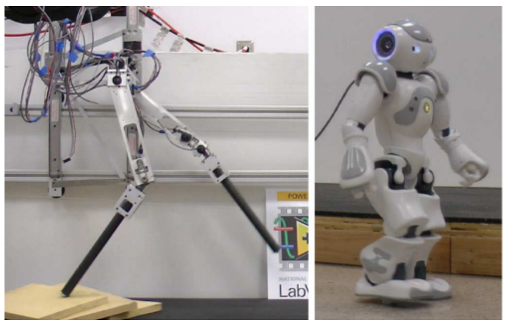


Figure 1.1: Drone Delivery

Emergency response system In Figure 1.2(a), we can see the emergency response system proposed in [122] that aims to solve the decision making problem of allocating ambulances to different stations in the city to provide better medical services for those with unplanned urgent and life-threatening health conditions. The allocation of ambulances depends on various factors, such as the daily demand for emergency services, the current allocation of ambulances to different stations, the time of the day, and even unexpected catastrophic accidents, such as a chemical plant explosion. However, due to the limited number of available ambulances, the allocation process must adhere to three types of resource constraints: global sum, local min and max, and group min and max. The global sum constraint dictates that the total number of ambulances assigned to all stations must be equal to the total number of available ambulances. The local min and max constraint, on the other hand, specifies the minimum and maximum number of ambulances that can be assigned to a single station. Lastly, in the group min and max constraint, stations are grouped together based on their locations to form larger bases, and the total number of ambulances assigned to stations belonging to the same group should be bounded. By taking these constraints into account, the emergency response system can make optimal decisions regarding ambulance allocation to ensure the efficient provision of medical services.



(a)



(b)

Figure 1.2: (a) Emergency response system; (b) 2D bipedal robot AMBER and 3D bipedal robot NAO

Bipedal walking robots Figure 1.2(b) displays the bipedal walking robots, AMBER and NAO, which were developed by the Mechanical Engineering Department at Texas A&M University [3]. The walking robots, which are inspired by the simplicity and elegance of human locomotion, are designed to aid in discovery and rescue missions, such as exploring polar ice caps or navigating through craters. The robot has three joints on each leg, which are hip joint, knee joint, and ankle joint. Each joint has a range of motion of restricted degrees. To ensure the stable and safe locomotion of walking robots, constraints on the robot's joint motion must be imposed. For instance, the range of motion for each joint must be limited to prevent the robot from taking steps that could lead to damage or falls. Additionally, the physical mechanism must also be followed to ensure that the movements of all joints are in sync. The decision making problem in this scenario is to control the motion of each joint so that the robot can traverse uneven or unknown terrains in a safe and stable manner.

1.2 Formulation

This section provides an introduction to two models that can be used to model sequential decision making problems with constraints in different settings. The first model is the constrained Markov decision process (CMDP), which is used to model sequential decision making problems with constraints in the single-agent setting. The second model is the constrained decentralized partially observable Markov decision process (Dec-POMDP), which is used for the multi-agent setting. For large-scale multi-agent systems, sequential decision making problems with constraints can be modelled using a constrained collective Dec-POMDP formulation, which will be presented in Chapter 2. We also introduce different types of constrains that used in both CMDPs and constrained Dec-POMDPs.

1.2.1 Constrained Markov decision process

A Markov decision process (MDP) model is defined using tuple $(S, A, P, R, \gamma, b_0)$. An agent can be in one of the states $s_t \in S$ at time t. It takes an action $a_t \in A$, receives a reward $R(s_t, a_t)$, and the world transitions stochastically to a new state s_{t+1} with probability $P(s_{t+1}|s_t, a_t)$. For the infinite-horizon setting, future rewards are discounted using a factor $0 < \gamma < 1$. The initial state distribution is denoted by $b_0(s)$. Let $\pi(\theta)$ denote a stochastic policy parameterized by θ. It is a mapping from state space (S) to action space (A). $\pi(a_t|s_t)$ is the probability of taking actions a in state s at time t. The goal of an MDP is to compute a stochastic policy π that maximizes the discounted cumulative reward $J_R^{\pi_\theta}$ as:

$$J_R^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \tag{1.1}$$

A constrained Markov decision process (CMDP) is defined as a tuple $(S, A, T, R, \{C_k\}_{k=1}^K, \gamma, b_0)$, where S, A, T, R, γ, and b_0 have the same definitions as those in a standard MDP. In addition, the tuple contains an extra component, namely $C_k(s_t, a_t)$, which represents the k^{th} cost function. This cost function specifies the immediate cost incurred after taking action a in state s at time t. Constraints in a CMDP can be formulated either explicitly or implicitly via cost functions. We will discuss different types of constraints in Section 1.2.3. Let Π_C denote the set of feasible policies that satisfy all the necessary constraints specified in the CMDP. In this context, the objective of a CMDP is to compute a stochastic policy π that maximizes the cumulative reward while satisfying all constraints. Formally, we have

$$\begin{aligned} \max_{\theta} \quad & J_R^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \\ \text{s.t.} \quad & \pi_\theta \in \Pi_C \end{aligned} \tag{1.2}$$

1.2.2 Constrained Dec-POMDP

A constrained Dec-POMDP, which extends the definition of traditional Dec-POMDP [8], is defined using the tuple $\langle S, \boldsymbol{A}, P, O, Z, R, \{C_k\}_{k=1}^K, N, \gamma, b_0\rangle$. There are N agents in the environment (indexed using $i = 1 : N$). The environment can be in one of the states $\boldsymbol{s} \in S$. At each time step, agent i chooses an action $a^i \in A$, resulting in the joint action $\boldsymbol{a} \in \boldsymbol{A} \equiv A^N$. As a result of the joint action, the environment transitions to a new state $\boldsymbol{s}'$ with probability $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$. The joint reward to the agent team is given as $R(\boldsymbol{s}, \boldsymbol{a})$. The joint k^{th} cost incurred for executing action $\boldsymbol{a}$ in state $\boldsymbol{s}$ at time t is given as $C_k(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The reward discount factor is $0 < \gamma < 1$, and b_0 is the initial state distribution.

We assume a partially observable setting in which agent i observes its own component $z^i \in Z$ with observation probability $O(z^i|\boldsymbol{a}, \boldsymbol{s}')$ where the last joint action taken was $\boldsymbol{a}$, and the resulting state was $\boldsymbol{s}'$. For simplicity, we have assumed the observation function is the same for all agents. As a result, different agents can receive different observations from the environment.

An agent's policy is a mapping from its action-observation history $\tau^i \in (Z \times A)^*$ to actions or $\pi_{\theta^i}(a^i|\tau^i)$, where θ^i parameterizes the policy. Let the discounted cumulative reward be denoted by $G_t = \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t)$. Let Π_C denote the set of feasible policies that satisfy all the necessary constraints specified in the constrained Dec-POMDP. The goal in the constrained Dec-POMDP is to find the best joint policy $\boldsymbol{\pi}$ to maximize the expectation of the discounted cumulative reward while satisfying all constraints. Formally, we have,

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & J_R^{\boldsymbol{\pi}_{\boldsymbol{\theta}}} = \mathbb{E}_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t)] \\ \text{s.t.} \quad & \boldsymbol{\pi}_{\boldsymbol{\theta}} \in \Pi_C \end{aligned} \tag{1.3}$$

1.2.3 Types of constraint

Both CMDPs and Dec-POMDPs incorporate two distinct types of constraints: cumulative constraints and instantaneous constraints. Furthermore, these two categories can be subdivided into two more categories: explicit constraints and implicit constraints. Figure 1.3 provides an overview of the common constraint categories utilized in constrained RL [70]. In this section, we focus on the different types of constraints present in CMDPs, which can also be defined similarly in Dec-POMDPs.

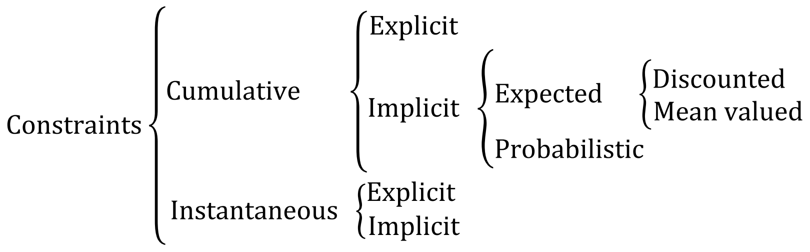


Figure 1.3: Types of constraint

Cumulative constraint

The cumulative constraint applies to the entire trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ of the CMDP. This constraint can be specified either explicitly or implicitly, with the latter being done using cost functions. Explicit cumulative constraints impose direct restrictions on all states jointly or all actions jointly in the trajectory, either through an analytical or Boolean function. In the air-delivery example, the landmark constraint and simple path constraint for an agent are examples of explicit cumulative constraints. we will provide a detailed discussion on how to model the landmark constraint and simple path constraint in Chapter 5. On the other hand, the implicit cumulative constraints use the cost function to define the restrictions on the trajectory. We can classify implicit cumulative constraints into three types, as follows.

- Discounted cumulative. This constraint requires that the sum of the discounted cost remains within a specified threshold, and it can be expressed in the form of

  $$J_{C_k}^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t C_k(s_t, a_t)] \leq c_k, \qquad k = 1, \ldots, K \tag{1.4}$$

  where $c_k, k = 1, \ldots, K$ is the constraint threshold for the k^{th} constraint.

- Mean valued. This constraint ensures that the mean of the total costs remains below a specified threshold within a finite time horizon, and it can be represented as

  $$J_{C_k}^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\frac{1}{T} \sum_{t=0}^{T-1} C_k(s_t, a_t)] \leq c_k, \qquad k = 1, \ldots, K \tag{1.5}$$

  where T is the total number of time steps in the horizon.

- Probabilistic. This constraint ensures that the probability of the cumulative cost violating a constraint remains within a specified bound, and it can be formulated as

  $$J_{C_k}^{\pi_\theta} = P(\sum_{t=0}^{T-1} C_k(s_t, a_t) \geq \epsilon_k) \leq c_k, \qquad k = 1, \ldots, K \tag{1.6}$$

  where $\epsilon_k, k = 1, \ldots, K$ is the threshold for the cumulative cost, and $c_k \in (0, 1)$ is the probability.

Instantaneous constraint

The instantaneous constraint, also known as an action constraint, applies to each action independently in a trajectory, restricting the agent's actions in every RL step. This constraint type can be specified explicitly or implicitly, similar to the cumulative constraint.

For certain instantaneous constraints, the restrictions on an action can be described explicitly using an analytical form. As an example, consider an emergency response system where 32 ambulances must be allocated to 25 stations at each decision making step. Here, an action with 25 dimensions can represent the allocation, with each action dimension $a_i, i = 1, \ldots, 25$ indicating the number of ambulances allocated to station i. In this case, the action constraint can be formulated as $\sum_{i=1}^{25} a_i = 32$. However, for some instantaneous constraints, an analytical form to describe the constraint does not exist. In such cases, using an immediate cost of taking action a in state s can provide a way to implicitly formulate the instantaneous constraint. Formally, we have,

$$C_k(s_t, a_t) \leq c_k, \quad k = 1, \ldots, K, \ \forall t \tag{1.7}$$

where $c_k, k = 1, \ldots, K$ is the constraint threshold for the k^{th} constraint.

1.3 Related works

In this section, we will begin by discussing relevant literature on solving single-agent constrained RL problems that involve both cumulative and instantaneous constraints. Then we introduce a few approaches for constrained multi-agent RL. Finally, we will discuss several recently proposed neuro-symbolic methods for RL. Table 1.1 provides an overview of representative works in single-agent constrained RL, multi-agent constrained RL, and neuro-symbolic.

Research Area	Setting	Method	Reference
Single-agent Constrained RL	Cumulative	Lagrangian	[2, 27, 106]
		IPO	[69]
		CPO & PCPO	[1, 119]
		Others	[28, 45, 114, 116]
	Instantaneous	Lagrangian	[14]
		Safety Layer	[4, 84]
		NFWPO	[66]
Multi-agent Constrained RL	Cumulative	Lagrangian	[31]
		CMIX	[68]
Neuro-symbolic	RL	PEORL & SDRL	[72, 118]
		SORL	[57]
		PROLONETS	[96]
	Non-RL	DL2	[39]
		MultiplexNet	[52]

Table 1.1: Overview of representative approaches in constrained RL and neuro-symbolic

1.3.1 Constrained single-agent reinforcement learning

Cumulative constraint

Implicit cumulative constraints are considered in most of the time. To address the CMDP with implicit cumulative constraints, various approaches have been proposed. Among these, Lagrangian Relaxation (LR) [2, 27, 106], interior-point policy optimization (IPO) [69], and trust region-based approaches, such as constrained policy optimization (CPO) [1] and projection-based constrained policy optimization (PCPO) [119], have gained significant attention.

The most popular approach is Lagrangian relaxation, which relaxes the original constrained problem to an unconstrained one by adding a Lagrange multiplier for each constraint. Policy and Lagrange multipliers are then iteratively updated using the gradient descent-ascent (GDA) approach on different time-scales. Theoretical proofs showing the convergence to the optimal solution under mild assumptions are provided in [82]. While LR is effective in penalizing constraint violations with Lagrange multipliers, it does have some limitations. Policy learning is sensitive to these multipliers. Learning the Lagrange multipliers is also sensitive to their initial values and learning rates. Furthermore, there is no guarantee that constraints will always be satisfied during training and execution. Other approaches, such as IPO and trust region-based approaches, have been proposed to address these limitations.

IPO, an approach inspired by the interior-point method, offers an alternative to Lagrangian Relaxation. It uses a differentiable logarithmic barrier function as the penalty function for constraint violations instead of Lagrange multipliers. The penalty function adds zero penalty to the objective if there is no constraint violation; otherwise, a penalty that approaches negative infinity is added to the objective. While IPO does require a hyperparameter related to the logarithmic barrier function to be tuned, this is generally easier than learning Lagrange multipliers. However, it is important to note that this approach does not provide theoretical guarantees for finding the optimal policy.

CPO and PCPO offer alternative approaches for solving CMDPs with cumulative constraints that do not rely on Lagrange multipliers or logarithmic barrier functions to penalize constraint violations. Instead, these methods extend trust region methods such as TRPO [88]. Practically, CPO updates policy parameters by solving an approximate quadratic constrained optimization problem, which can be efficiently solved using duality. PCPO, on the other hand, learns a policy in two steps. First, the policy is updated to maximize the cumulative reward using TRPO without considering constraints. Second, the intermediate policy is projected back to the constraint set. Both CPO and PCPO provide theoretical analysis of bounds for return improvement and constraint violation without requiring penalties for constraint violation. However, these methods are computationally expensive, as they rely on approximating the Fisher information matrix in solving the quadratic constrained program.

In addition to the above approaches, other approaches have also been proposed to solve CMDPs with implicit cumulative constraint. Constructing Lyapunov functions to guarantee constraint satisfaction via a set of local linear constraints is studied in [28]. Forming a max-min problem by constructing a lower bound for the objective is used in [45]. Choosing the objective and constraints randomly to optimize is proposed in [116]. Solving probabilistic constraints by converting them to cumulative constraints via state augmentation is proposed in [114]

Instantaneous constraint

LR has also been attempted to solve CMDPs with implicit instantaneous constraints, as seen in the work of Bohez et al. [14]. In this approach, the reward in each RL step is augmented with the sum of penalties for each instantaneous constraint violation. The penalty for each violation is determined by the Lagrange multiplier associated with the corresponding instantaneous constraint. However, optimizing these Lagrange multipliers in each RL step becomes challenging and intractable. Moreover, while Lagrange multipliers can penalize constraint violations, this approach cannot guarantee zero constraint violation during training and policy execution.

There are some recently proposed approaches aiming to solve explicit instantaneous constraint and aiming to achieve zero constraint violation. One natural approach is to add a differentiable projection layer at the end of the policy network to satisfy instantaneous constraints [4, 84]. The projection layer projects actions from the unconstrained policy onto the feasible action space by solving a Quadratic Program (QP). However, this approach has two primary limitations. First, its scalability is limited, especially for problems with large action spaces, due to the computational cost of solving a QP at each RL step. Second, there is a potential issue of zero-gradient, which could arise during the end-to-end training of the policy network. This issue could undermine the effectiveness of the projection layer approach, as has been noted in recent studies [66]. To tackle the zero-gradient issue, a learning algorithm called NFWPO that decouples policy gradients from the projection layer is proposed in [66], where policy parameters are updated by leveraging the Frank-Wolf method [42]. Unfortunately, this approach does not scale up well as it still requires solving a QP during RL training.

1.3.2 Constrained multi-agent reinforcement learning

Most of works in constrained reinforcement learning focus on single-agent setting. Although these approaches can solve the single agent constrained RL, extending them such as LR to multi-agent constrained RL directly is not trivial. There are few works aiming to solve multi-agent constrained RL.

In [31], the authors apply the LR method to solve the constrained multi-agent RL problem. To solve the relaxed problem, they employed a multi-agent actor-critic approach. Specifically, a centralized policy critic was trained using the modified reward, which includes the sum of costs weighted by the Lagrange multipliers. Additionally, they learned a centralized penalty critic for each constraint. The update of policy parameters and Lagrangian multipliers are guided by the policy critic and penalty critics respectively. However, centralized critics can be noisy since contributions from each individual agent are not clear. [71] also proposed LR but in a setting where agents are allowed to communicate over a pre-defined communication network.

As discussed in [82], the convergence to a neighborhood of the optimal solution in single-agent constrained RL relies on two crucial factors: (1) the error introduced by the parametrization of the policy and (2) the goodness of the solution obtained through maximizing the Lagrangian using RL algorithms like actor-critic. The theoretical analysis on convergence in single-agent constrained RL can be extended to the multi-agent setting, provided that these two factors are carefully addressed. However, in the multi-agent setting, encountering a policy parameterization with minimal error and designing an actor-critic RL algorithm that performs well are challenging tasks due to the large state and action spaces involved, as well as the difficulties of multi-agent credit assignment.

A recent work called CMIX [68] proposes a novel approach to combining the multi-objective programming and Q-mix framework [40] to address the constrained multi-agent problem. The approach involves converting the constrained problem into an equivalent max-min optimization problem by setting a lower bound on the objective. The resulting problem is then solved using the value function factorization technique, namely Q-mix. However, a major challenge of this approach is scalability, as it requires different

Q-function approximators for each constraint and each agent.

1.3.3 Neuro-symbolic for RL

Neuro-symbolic methods combine the strengths of both neural and symbolic approaches. There are six major ways to integrate them, each with its unique advantages and AI application fields, such as NLP and CV [60]. In this section, we will focus on two integration modes, which have been used in the field of RL.

The first integration forms a hybrid system where neural and symbolic parts work on different but complementary tasks to solve a problem collaboratively. The interaction between neural and symbolic method provided a way to solve the problem more efficiently. The PEORL framework combines symbolic planning utilizing the action language $\mathcal{BC}$ with hierarchical R-learning to tackle decision making problems with uncertainties [118]. This approach generates a symbolic plan, using the CLINGCON planner [5], based on an initial state, a goal, and a specific set of causal laws. The symbolic plan is then mapped into a sequence of stochastic options. Hierarchical R-learning tries to maximize the average of cumulative reward (termed as gain reward) based on the stochastic options. This gain reward can be used to generate a symbolic plan of improved quality via CLINGCON. This iterative process continues until the highest quality symbolic plan is obtained. Further improvements to the PEORL framework were made in the SDRL approach proposed by Lyu et al. (2019) [72], which introduced intrinsic goals that measure plan quality based on an internal utility function. A meta-controller was then introduced to connect between deep RL algorithms and the symbolic planner, allowing for even greater generality and flexibility in addressing decision making problems. In PEORL and SDRL, action models are typically defined manually by domain experts. However, this process can be challenging, particularly in complex domains. To address this issue, SORL has been proposed to learn high-level action and option models from collected trajectories obtained from the low level with minimal human knowledge [57]. A symbolic planner at the high level generates a plan using the learned action models, while the lower level learns a policy to achieve the specified options in the plan and sends the collected trajectories back to the high level.

The second integration compiles the symbolic knowledge into the weights of the neural network. The output of the neural network will satisfy constraints implied by the symbolic knowledge. PROLONETS is one representative recent work that utilizes this integration mode to address a reinforcement learning problem [96]. The approach begins by expressing a high-level hierarchical policy provided by human experts as a decision tree. Next, the decision nodes in the tree are translated into weights and biases, which are used to create linear layers in a neural network. This process enables an intelligent initialization of the neural network's weights and architecture based on the available domain knowledge. By incorporating symbolic knowledge into the neural network's design, PROLONETS can effectively leverage the strengths of both symbolic and neural approaches to solve a RL problem.

In addition to solving a RL problem, there are also other neuro-symbolic methods that tackle constraint-related problems. To ensure domain constraints are met, one widely used approach is to create a loss function that quantifies the degree of deviation between the output of a neural network and the closest feasible solution. DL2 [39] and the semantic loss [115] are two notable works in this area. DL2 converts logical constraints into a non-negative loss using fuzzy logic, which is differentiable almost everywhere and can be optimized using projected gradient descent. In contrast, the semantic loss involves a function that uses a sentence in propositional logic and probabilities over a set of Boolean variables. This function assesses the likelihood that a sample from the network output satisfies a Boolean constraint formulation.

MultiplexNet is a novel approach that differs from the conventional approach of using loss terms to enforce domain constraints [52]. Instead, it incorporates domain constraints that are represented using a logical formula in a disjunctive normal form (DNF) directly into a parameterized neural network. This method guarantees that the output of the neural network adheres to the constraints represented in DNF form. To learn the parameters of the model, the variational lower bound (ELBO) is maximized by introducing a categorical latent variable, which is used to select different terms in DNF.

1.4 Our contributions

In this thesis, we explored a wide range of settings in sequential decision making with constraints, including single-agent and multi-agent settings, cumulative and instantaneous constraints, as well as soft and hard constraints. We have developed various approaches tailored to different combinations of settings. Our thesis presents several significant contributions. Firstly, we address the problem of large-scale constrained multi-agent reinforcement learning by proposing a method that tackles a constrained collective Dec-POMDP model. To do this, we introduce a fictitious play based approach to perform credit assignment for both the primary objective and the constraints. Secondly, we propose a zone-based multi-agent pathfinding (ZBPF) framework that is motivated by the drone delivery application. We identify and consider several types of challenging constraints in this framework and propose a neuro-symbolic method to efficiently solve the sequential decision making problem with hard constraints in ZBPF. Lastly, we tackle the problem of constrained RL with discrete and combinatorial action spaces, and present another neuro-symbolic method that can solve the problem with hard constraints in a general sense. We summarize our contributions in details from four different aspects as follows.

1.4.1 Constrained collective multi-agent reinforcement learning

Solving constrained multi-agent reinforcement learning (MARL) is a challenging task, as credit assignment for both primary objectives and constraints must be performed jointly. The accurate deduction of each agent's role in optimizing the primary objective and minimizing constraint violations becomes increasingly difficult when there are hundreds of agents involved.

In our thesis, we focus on the constrained collective MARL problem, where agent interactions are governed by the aggregate count and types of agents using the collective Dec-POMDP framework [76] augmented with constraints. To address the credit assignment challenge, we propose a fictitious play based constrained MARL method that uses Lagrangian relaxation. We test our method on both real-world and synthetic datasets for the maritime traffic management problem [100] and the network routing problem [68], and we show that it significantly outperforms the standard LR method for MARL in satisfying constraints. Moreover, our approach achieves lower average and peak constraint violations than CMIX [68] while using significantly fewer samples to achieve a similar global objective.

1.4.2 Zone-based multi-agent pathfinding

We propose a novel framework named *zone-based pathfinding (ZBPF)* for solving the multi-agent sequential decision-making problem with constraints encountered in the drone delivery application. Our framework is inspired by the air blocks in the delivery application and based on the concept of zones. Agents move among zones from their respective resources to destinations, with zones having different capacities for accommodating multiple agents and variable navigation times. We also consider uncertain navigation times due to environmental factors such as weather conditions.

To model the problem, we use a variant of the constrained Dec-POMDP and a highly expressive class of exponential family distributions to model the uncertain duration of agents' movements. We use multi-agent reinforcement learning (MARL) to solve this problem, and instead of penalizing the violation of the zone capacity constraint using Lagrange multipliers, we design a cost function that assigns a congestion penalty to each agent in a zone if the total number of agents exceeds the zone's capacity. We then deduct the cost from the reward for each agent to reduce the negative impact of constraint violations, converting the constrained problem into an unconstrained one. To address the policy optimization problem, we present a novel difference-of-convex functions (DC) programming [67] based formulation, which allows us to solve a sequence of optimization problems using samples from the domain simulator. Additionally, we develop value function factorization techniques for faster convergence [22, 41].

To validate our approach, we also develop a highly expressive simulator for ZBPF in the *ml-agents* platform of the *Unity3D* game engine, which can represent both 2D and 3D maps, providing a clean interface from the simulation environment to learning algorithms [58].

1.4.3 Neuro-symbolic for zone-based multi-agent pathfinding

The zone-based multi-agent pathfinding problem is challenging due to the difficulty in finding a policy that satisfies zone capacity constraints and the time-consuming nature of finding a route for a single agent due to the lack of explicit exploitation of graph connectivity by model-free RL. Additionally, agents may move in cycles, leading to inefficient sampling, and the problem becomes more complex when agents must satisfy additional constraints such as visiting some landmark nodes before reaching the destination, or coverage constraints such as visiting some locations at least once every k time steps.

To overcome these challenges, we propose a neural-symbolic approach to ZBPF. We compile connectivity of the underlying graph where agents move on and other domain hard constraints such as landmark constraint, coverage constraint using propositional logic based *sentential decision diagrams* (sdd) [30]. An sdd is a succinct and tractable representation of a Boolean formula, and generalizes an ordered binary decision diagram (OBDD) [18]. The benefit of using SDDs is that any satisfiable instantiation of the SDD is a valid simple path between the source and destination that satisfies all domain constraints, which prunes the search space and generates high-quality training samples for the learning algorithm.

We integrate the compiled knowledge represented by an SDD with various MARL algorithms, including policy gradient and Q-learning, and we have developed an efficient inference method for SDDs with linear worst-case complexity in the size of the SDD representation. This allows for fast interaction between the symbolic method and MARL algorithms during training. Our neuro-symbolic approach outperforms the original ZBPF algorithm in terms of sample complexity and solution quality in multiple instances. We also compare the efficiency of our inference algorithm with specialized graph heuristics and find our approach to be significantly faster in several settings.

1.4.4 Neuro-symbolic for constrained combinatorial action space

In Section 1.3.1, we explored several recent methods for addressing CMDPs with instantaneous constraints on continuous actions in a single-agent setting. However, when the action space is discrete, and combinatorial, there are few effective algorithms available for this setting. While some methods exist for specific types of action constraints, such as resource allocation, they are not easily extendable to a general constrained combinatorial action space. To address this gap, we present a general neuro-symbolic approach for hard action constrains in discrete and combinatorial action spaces.

Our method represents an action using a set of propositional variables, and action constraints using Boolean formulas. We use *probabilistic sentential decision diagrams* (psdds) [61], a parameterized sdd that encodes a probability distribution over the set of all valid actions satisfying action constraints. One key benefit of using psdds is that they can represent a combinatorial, constrained action space compactly.

Our neuro-symbolic method reformulates the task of optimizing a policy over a constrained combinatorial action space as optimizing the parameters of the underlying psdd, which is significantly smaller than the action space, and thus computationally tractable. We show how to encode psdd parameters using deep neural networks, and optimize them using a deep Q-learning based algorithm for factored actions [108]. We develop new techniques to integrate the optimization of psdd parameters with deep RL algorithms. Our approach is designed to guarantee the output of a feasible action in each RL step without involving computationally expensive projection steps over the constraint space. As a result, our approach can achieve zero constraint violation during both training and execution.

We also demonstrate how to encode practical resource allocation constraints, such as those in emergency response systems, using a psdd, and prove that the compiled psdd has a polynomial size in the number of resources and entities. We evaluate our approach empirically with two previous ACRL methods [12, 66] and demonstrate that our method can achieve zero constraint violation, is scalable, and provides better solution quality in several instances.

1.5 Roadmap

The thesis is divided into six main chapters. In Chapter 2, we consider constrained collective multi-agent RL and develop a credit assignment scheme for both primary objective and constraints. In Chapter 3, we introduce the zone based pathfinding framework for multiple agents with capacity constraints. Chapter 4 provides an introduction to decision diagrams and their use in compiling various types of constraints in a symbolic manner. In Chapter 5, we propose a neuro-symbolic approach to effectively solve the zone-based pathfinding (ZBPF) problem while considering other domain constraints, such as landmark and coverage constraints, in addition to capacity constraints. In Chapter 6, we introduce another neuro-symbolic method to solve general action constraints problem with discrete and combinatorial action spaces. Chapter 7 concludes the thesis and introduces two promising future directions in the field of constrained RL.

1.6 Publications

We list the publications that contribute to this thesis as follows:

[1] **Ling, Jiajing**, Arambam James Singh, Nguyen Duc Thien, and Akshat Kumar. "Constrained Multiagent Reinforcement Learning for Large Agent Population." In *Machine Learning and Knowledge Discovery in Databases: European Conference*, pp. 183-199, 2023. (Chapter 2)

[2] **Ling, Jiajing**, Tarun Gupta, and Akshat Kumar. "Reinforcement Learning for Zone Based Multiagent Pathfinding under Uncertainty." In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 551-559, 2020. (Chapter 3)

[3] **Ling, Jiajing**, Kushagra Chandak, and Akshat Kumar. "Integrating knowledge compilation with reinforcement learning for routes." In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 542-550, 2021. (Chapter 5)

[4] **Ling, Jiajing**, Moritz Lukas Schuler, Akshat Kumar and Pradeep Varakantham. "Knowledge Compilation for Constrained Combinatorial Action Spaces in Reinforcement Learning." In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2023. (Chapter 6)

Chapter 2

Constrained Collective Multi-agent Reinforcement Learning

Learning control policies for a large number of agents in a decentralized setting poses several challenges due to partial observability, environmental uncertainty, and scalability issues. Despite the existence of scalable multi-agent reinforcement learning (MARL) methods, few approaches exist for addressing large-scale constrained MARL settings. This chapter aims to solve the problem of constrained large-scale MARL with cumulative constraints. We start by presenting a formulation for the constrained MARL problem in a collective multi-agent setting in Section 2.1. We then introduce a proposition regarding individual value representation in Section 2.2, which forms the basis of our proposed method. In Section 2.3, we develop a fictitious play based MARL method that uses Lagrangian relaxation to address the challenge of credit assignment for both primary objectives and constraints. Finally, we evaluate our approach on two real-world applications, namely, maritime traffic management and vehicular network routing, in Section 2.4.

2.1 Constrained collective Dec-POMDP

The collective Dec-POMDP ($\mathbb{C}$Dec-POMDP) is a widely used framework for modeling sequential decision-making problems in large-scale multi-agent systems. This framework has numerous real-world applications, such as traffic control, transport management, or resource allocation [76, 100, 112]. In the $\mathbb{C}$Dec-POMDP, the transition and reward of each agent depend on the count values of agents in various local states. In this work, we extend the $\mathbb{C}$Dec-POMDP model to incorporate constraints in multi-agent decision making scenarios, and introduce a constrained $\mathbb{C}$Dec-POMDP. A constrained $\mathbb{C}$Dec-POMDP is defined as a tuple $\langle S, A, \{\mathbf{n}_t\}_{t=1}^{T}, P, O, \{R_m\}_{m=1}^{M}, \{C_k\}_{k=1}^{K}, N, \gamma, b_0, T\rangle$, where

- N is the total number of agents in the system.
- T is the horizon of the problem.
- S is a finite set of states, which is the same for all agents. An agent $i \in \{1, \ldots, N\}$ can be in one of the states, i.e., $s^i \in S$. The joint state is $\boldsymbol{s} \in S^N$. We denote a single state as $s \in S$.
- A is a finite set of actions. An agent i chooses an action $a^i \in A$, resulting in the joint action $\boldsymbol{a} \in A^N$. We denote an individual action as $a \in A$.
- $\mathbf{n}_t = (\mathbf{n}_t^{\mathrm{s}}, \mathbf{n}_t^{\mathrm{sa}}, \mathbf{n}_t^{\mathrm{sas}})$ is a combined vector of count tables at time step t where $\mathbf{n}_t^{\mathrm{s}} = (\mathbf{n}_t(s)\ \forall s \in S)$ is the state count table, $\mathbf{n}_t^{\mathrm{sa}} = (\mathbf{n}_t(s,a)\ \forall s \in S, a \in A)$ is the state-action count table, and $\mathbf{n}_t^{\mathrm{sas}} = (\mathbf{n}_t(s,a,s')\ \forall s, s' \in S, a \in A)$ is the transition count table. Given a complete joint state-action trajectory $\tau = (\boldsymbol{s}_1, \boldsymbol{a}_1, \boldsymbol{s}_2 \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T)$, we define the counts $\mathbf{n}_t(s,a,s')$, $\mathbf{n}_t(s,a)$ and $\mathbf{n}_t(s)$ as follows.
  - $\mathbf{n}_t(s,a,s') = \sum_{i=1}^{N} \mathbb{I}(s_t^i = s, a_t^i = a, s_{t+1}^i = s') \qquad \forall s, s' \in S, a \in A$
  - $\mathbf{n}_t(s,a) \quad = \sum_{i=1}^{N} \mathbb{I}(s_t^i = s, a_t^i = a) \qquad \forall s \in S, a \in A$
  - $\mathbf{n}_t(s) \qquad = \sum_{i=1}^{N} \mathbb{I}(s_t^i = s) \qquad \forall s \in S$

  where s_t^i and a_t^i denote the state and action of agent i at time t respectively. The individual indicator function $\mathbb{I}(s_t^i = s, a_t^i = a) \in \{0, 1\}$ indicates whether the agent i is in local state s and taking action a at time step t. Other indicator functions are defined similarly. For a given subset $S' \subseteq S$, we define a count table for agents in S' as $\mathbf{n}_t[S'] = (\mathbf{n}_t(s,a,s')\ \forall s \in S', a \in A, s' \in S)$.
- P is a local transition function, which is the same for all agents. Given a local state s_t^i and action a_t^i taken by agent i at time step t, the probability of reaching the next local state s_{t+1}^i is $P(s_{t+1}^i | s_t^i, a_t^i, \mathbf{n}_t^{\mathrm{sa}})$. Note that it is also affected by $\mathbf{n}_t^{\mathrm{sa}}$,

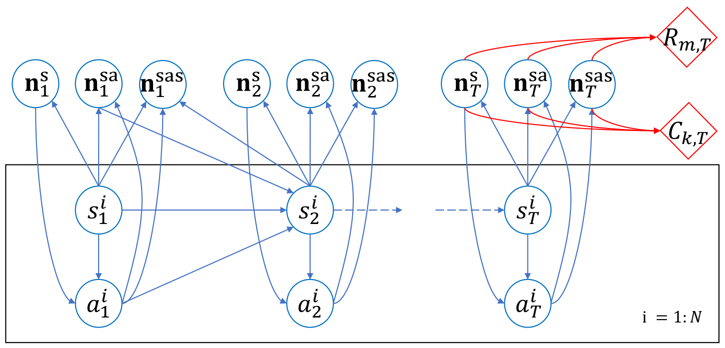

$\mathbf{n}_1^{\mathrm{s}}$
$\mathbf{n}_1^{\mathrm{sa}}$
$\mathbf{n}_1^{\mathrm{sas}}$
$\mathbf{n}_2^{\mathrm{s}}$
$\mathbf{n}_2^{\mathrm{sa}}$
$\mathbf{n}_2^{\mathrm{sas}}$
$\mathbf{n}_T^{\mathrm{s}}$
$\mathbf{n}_T^{\mathrm{sa}}$
$\mathbf{n}_T^{\mathrm{sas}}$
$R_{m,T}$
$C_{k,T}$
s_1^i
s_2^i
s_T^i
a_1^i
a_2^i
a_T^i
i = 1: N

Figure 2.1: Dynamic Bayes Net (DBN) for T-step reward and cost for constrained ℂDec-POMDP (assuming each agent has a full observation of the count table)

which depends on the collective behavior of the agent population.

- O is the observation function. For simplicity, we assume the observation function is the same for all agents. The observation of agent i is given by $O(i, \mathbf{n}_t^{\mathrm{s}})$.
- R_m is the m^{th} reward function, which depends on the combined count tables $\mathbf{n}_t$.
- C_k is the k^{th} cost function, which depends on the combined count tables $\mathbf{n}_t$ too.
- $\gamma < 1$ is the discount factor for reward.
- b_0 is the initial state distribution.

For simplicity, we assume each agent is able to fully observe the whole count table. Let policy $\pi_{\theta^i}(a|s, \mathbf{n}_t^{\mathrm{s}})$ denote the probability of agent i taking action a given its local state s and the count table $\mathbf{n}_t^{\mathrm{s}}$, where θ^i parameterizes the policy. When agents have the same policy, we can ignore the index and denote the common policy as π_θ. Figure 2.1 describes graphically how different agents interact with each other using a dynamic Bayes net (and plate notation) in the constrained ℂDec-POMDP model. The goal in a constrained ℂDec-POMDP is to find a policy (assuming the same for all agents) to solve the constrained program as follows.

$$\max_{\theta} \quad J_R^{\pi_\theta} = \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} \gamma^t \sum_{m=1}^{M} R_m(\mathbf{n}_t)|\pi_\theta] \tag{2.1a}$$

$$\text{s.t.} \quad J_{C_k}^{\pi_\theta} = \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} \gamma^t C_k(\mathbf{n}_t)|\pi_\theta] \le c_k, \qquad \forall k \tag{2.1b}$$

where $c_k, k = 1, \dots, K$ is the constraint threshold for the k^{th} constraint.

Agents with types We can also associate different types with different agents to distinguish them (e.g., 4-seater taxi, 6-seater taxi). This can be done using a type-augmented state space as $\mathcal{S} = S \times \mathcal{T}$, where $\mathcal{T}$ is the set of possible agent types. The main benefit of the collective modeling is that we can exploit the aggregate nature of interactions among agents when the number of types is much smaller than the number of agents

Simulator for MARL In the MARL setting, we do not have access to the transition and reward function. As shown in [77], a count based simulator provides the experience tuple for the centralized learner as $(\mathbf{n}_t^{\text{s}}, \mathbf{n}_t^{\text{sa}}, \mathbf{n}_t^{\text{sas}}, R_m(\mathbf{n}_t), C_k(\mathbf{n}_t))$. In other words, simulation and learning in the collective setting can be done at the abstraction of counts. This avoids the need to keep track of individual agents' state-action trajectories and increases computational scalability for a large number of agents.

2.2 Individual value representation

Solving Problem(2.1) is difficult because the constraints are globally coupled with the joint counts $\mathbf{n}_{1:T}$. In many domains in practice, the reward and cost functions only involve the count variables over a subset of states S. For example, in congestion domain, we have the penalty cost defined for a specific area/zone. We consider a general framework where we can define the subset $S_k \subseteq S$ that affects constraint k, and the subset $S_m \subseteq S$ that affects reward R_m. In extreme case where a function is non-decomposable, S_k can be set to S. Let $\mathbf{n}_t[S_m]$ denote the count table that summarizes the distribution of agents in S_m (as defined in Section 2.1). Let $|\mathbf{n}_t[S_m]|$ denote the number of agents in S_m. We can re-write (2.1) as follows:

$$\max_{\theta} \quad J_R^{\pi_\theta} = \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} \gamma^t \sum_{m=1}^{M} R_m(\mathbf{n}_t[S_m]) | \pi_\theta] \tag{2.2a}$$

$$\text{s.t.} \quad J_{C_k}^{\pi_\theta} = \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} \gamma^t C_k(\mathbf{n}_t[S_k]) | \pi_\theta] \leq c_k, \qquad \forall k \tag{2.2b}$$

Furthermore, we show that we can re-write the global constrained program in the form of an individual agent's constrained program. For a specific function f_g, which can be either cost $f_g = C_g(\mathbf{n}_t[S_g])$ or reward function $f_g = R_g(\mathbf{n}_t[S_g])$, we define an auxiliary individual function:

$$f_g^i(s_t^i, a_t^i, \mathbf{n}_t[S_g]) = \begin{cases} \frac{f_g(\mathbf{n}_t[S_g])}{|\mathbf{n}_t[S_g]|} & \text{if } s_t^i \in S_g \\ 0 & \text{otherwise} \end{cases}$$

We use $\boldsymbol{s}_{1:T}^i, \boldsymbol{a}_{1:T}^i$ to denote the state-action trajectory with length T of agent i, and use $\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}$ to denote the join state-action trajectory of all N agents in our system.

Proposition 1. *Consider any reward/cost component f_g. The global expected value of f_g is equal to a factor of individual value function:*

$$\mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} \gamma^t f_g(\mathbf{n}_t[S_g])|\pi_\theta] = N \times \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T} \gamma^t f_g^i(s_t^i, a_t^i, \mathbf{n}_t[S_g])] \tag{2.3}$$

where N is the total number of agents in the system.

Proof. By applying the exchangeability theorem from [76], we can derive the individual function for reward/cost component f_g as:

$$\mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T} \gamma^t f_g^i(s_t^i, a_t^i, \mathbf{n}_t[S_g])]$$

$$= \sum_{t=1}^{T} \gamma^t \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[f_g^i(s_t^i, a_t^i, \mathbf{n}_t[S_g])] \tag{2.4}$$

We replace the joint probability $P(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T})$ with $P(\boldsymbol{s}_{1:t}^i, \boldsymbol{a}_{1:t}^i, \mathbf{n}_{1:t})$.

$$= \sum_{t=1}^{T} \gamma^t \sum_{\boldsymbol{s}_{1:t}^i, \boldsymbol{a}_{1:t}^i, \mathbf{n}_{1:t}} P(\boldsymbol{s}_{1:t}^i, \boldsymbol{a}_{1:t}^i, \mathbf{n}_{1:t}) f_g^i(s_t^i, a_t^i, \mathbf{n}_t[S_g]) \tag{2.5}$$

$$= \sum_{t=1}^{T} \gamma^t \sum_{\boldsymbol{s}_{1:t}^i, \boldsymbol{a}_{1:t}^i, \mathbf{n}_{1:t}} P(\boldsymbol{s}_{1:t}^i, \boldsymbol{a}_{1:t}^i, \mathbf{n}_{1:t}) \sum_{s' \in S_g} \mathbb{I}(s_t^i = s') \frac{f_g(\mathbf{n}_t[S_g])}{|\mathbf{n}_t[S_g]|} \tag{2.6}$$

We now apply the exchangeability of agents with respect to the count variables $\mathbf{n}$ [76]:

$$= \sum_{t=1}^{T} \gamma^t \sum_{\mathbf{n}_{1:t}} P(\mathbf{n}_{1:t}) \sum_{s' \in S_g} \frac{\mathbf{n}_t(s')}{N} \frac{f_g(\mathbf{n}_t[S_g])}{|\mathbf{n}_t[S_g]|} \tag{2.7}$$

$$= \frac{1}{N} \mathbb{E}_{\mathbf{n}_{1:T}} [\sum_{t=1}^{T} \gamma^t f_g(\mathbf{n}_t[S_g]) | \pi_\theta] \tag{2.8}$$

□

2.3 Fictitious play based constrained optimization

By applying Proposition 1 to the global reward and cost functions in (2.2), we will get individual reward and cost functions. Therefore, we can define an individual constrained RL problem for each agent, and we have the following lemma.

Lemma 1. *Solving a collective constrained reinforcement learning problem defined in* (2.1) *is equivalent to solving the individual constrained reinforcement learning problems defined as follows:*

$$\max_{\theta} \quad \mathbb{E}_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}} [\sum_{t=1}^{T} \gamma^t \sum_{m=1}^{M} R^i_m(s^i_t, a^i_t, \mathbf{n}_t[S_m]) | \pi_\theta] \tag{2.9a}$$

$$\text{s.t.} \quad \mathbb{E}_{\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}} [\sum_{t=1}^{T} \gamma^t C^i_k(s^i_t, a^i_t, \mathbf{n}_t[S_k]) | \pi_\theta] \leq \frac{c_k}{N}, \qquad \forall k \tag{2.9b}$$

To solve Prolem (2.9), we apply fictitious-play [75] based constrained optimization (FICO) in which at each iteration, agent tries to optimize its own policy given the joint state-action samples and ignore the effect of its policy change on other agents. We also highlight that Problem (2.9) is the key to performing the credit assignment for primary objective and constraints. This problem clearly separates out the contribution of each agent i to the value function (or V^i) and each constraint k (or C^i_k). Therefore, the FICO method enables effective credit assignment for both primary objective and constraints. Amongst popular methods to solve constrained RL, we apply the Lagrangian relaxation method to solve FICO (2.9). The Lagrange dual problem is given as follows.

$$\min_{\boldsymbol{\lambda}\geq 0}\max_{\theta}\mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T}\gamma^t\big(\sum_{m=1}^{M}R^i_m(s^i_t,a^i_t,\mathbf{n}_t[S_m])-\sum_{k=1}^{K}\lambda_k C^i_k(s^i_t,a^i_t,\mathbf{n}_t[S_k])\big)+\sum_{k=1}^{K}\lambda_k\frac{c_k}{N}|\pi_\theta] \tag{2.10}$$

To solve this dual Problem (2.10), we apply stochastic gradient ascent-descent to alternatively update parameters θ of the policy and the Lagrange multiplier $\boldsymbol{\lambda}$ following the two-time scale approximation [106].

Individual policy update To optimize π_θ, we first compute the modified reward as follows.

$$R'(s^i_t,a^i_t,\mathbf{n}_t)=\sum_{m=1}^{M}R^i_m(s^i_t,a^i_t,\mathbf{n}_t[S_m])-\sum_{k=1}^{K}\lambda_k C^i_k(s^i_t,a^i_t,\mathbf{n}_t[S_k])$$

Given the fixed Lagrange multipliers, parameters θ are optimized by solving the following problem,

$$\max_{\theta}\mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T}R'(s^i_t,a^i_t,\mathbf{n}_t)|\pi_\theta] \tag{2.11}$$

The benefit of the above representation is that now we can apply various techniques developed for collective Dec-POMDPs to optimize (2.11) using stochastic gradient ascent. To optimize (2.11), we consider a fictitious play based approach to compute policy gradient for policy π_θ of agent i over all possible local state s and individual action a. Using the standard policy gradient [103] with respect to an individual agent i, we can perform the update for θ as follows:

$$\begin{aligned}\theta' =&\theta+\alpha_\theta\sum_{t=1}^{T}\sum_{s,a,\mathbf{n}_t}\mathbb{E}_{\boldsymbol{s}_{t:T},\boldsymbol{a}_{t:T}}[\mathbb{I}(s^i_t=s,a^i_t=a)\mathbb{I}(\mathbf{n}_t\sim\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{s}_{t+1})\\ &\times\sum_{t'=t}^{T}R'(s^i_{t'},a^i_{t'},\mathbf{n}_{t'})|\pi_\theta]\nabla_\theta\log\pi_\theta(a|s,\mathbf{n}_t)\end{aligned} \tag{2.12}$$

where $\mathbb{I}(\mathbf{n}_t\sim\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{s}_{t+1})$ is an indicator function for whether the count table of the joint transition $\boldsymbol{s}_t,\boldsymbol{a}_t,\boldsymbol{s}_{t+1}$ is identical to $\mathbf{n}_t$. Applying results from [76] for collective Dec-POMDPs, we can sample the counts (using the current policy) and use these counts to compute the gradient term in (2.12) as:

$$
\begin{aligned}
&\mathbb{E}_{\boldsymbol{s}_{t:T},\boldsymbol{a}_{t:T}}[\mathbb{I}(s^i_t = s, a^i_t = a)\mathbb{I}(\mathbf{n}_t \sim \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) \sum_{t'=t}^{T} R'(s^i_{t'}, a^i_{t'}, \mathbf{n}_{t'})|\pi_\theta] \\
=&\sum_{\mathbf{n}'_{1:T}} P(\mathbf{n}'_{1:T})\mathbb{I}(\mathbf{n}'_t = \mathbf{n}_t)\frac{\mathbf{n}'_t(s,a)}{N}\sum_{t'=t}^{T}\sum_{s^i_{t'},a^i_{t'}}\frac{\mathbf{n}'_t(s^i_{t'},a^i_{t'})}{N}R'(s^i_{t'},a^i_{t'},\mathbf{n}'_{t'}) \qquad (2.13)
\end{aligned}
$$

The above expected value can be estimated by a Monte-Carlo approximation $\hat{Q}^i_t(s,a,\mathbf{n}_t)$ with samples $\xi = 1,\ldots,K$ of counts [76]. For a given count sample $\mathbf{n}^\xi_{1:T}$:

$$
\begin{aligned}
V^\xi_T(s,a) &= R'_T(s,a,\mathbf{n}^\xi_T(s)) && (2.14)\\
V^\xi_t(s,a) &= R'_t(s,a,\mathbf{n}^\xi_t(s)) + \sum_{a'}\frac{\mathbf{n}^\xi_t(s,a',s')}{\mathbf{n}^\xi_t(s)}V^\xi_{t+1}(s',a') && (2.15)\\
Q^\xi_t(s,a) &= \frac{\mathbf{n}^\xi_t(s,a)}{N}\times V^\xi_t(s,a) && (2.16)\\
\hat{Q}^i_t(s,a,\mathbf{n}_t) &= \frac{1}{K}\sum_{\xi|\mathbf{n}^\xi_t=\mathbf{n}_t} Q^\xi_t(s,a) && (2.17)
\end{aligned}
$$

Continuous actions We highlight that even though we have formulated FICO for discrete action spaces, our method works for continuous action space also as long as the policy gradient analogue of (2.12) is available for continuous actions. Empirically, we do test on the maritime traffic control problem where action space is continuous, and policy gradient is derived in [100].

Lagrange multiplier update Given a fixed policy π_θ, the Lagrange multiplier λ_k for each constraint k is optimized by solving the following problem.

$$
\min_{\lambda_k\geq 0}\mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\lambda_k(\frac{c_k}{N}-\sum_{t=1}^{T}C^i_k(s^i_t,a^i_t,\mathbf{n}_t[S_k]))|\pi_\theta] \qquad (2.18)
$$

Algorithm 2.1: FICO

```
1  Initialize policy network parameters $\theta$ and Lagrange multipliers $\lambda_k = 0, \forall k$
2  Set learning rates $\alpha_k, \forall k$ and $\alpha_\theta$ with $\alpha_k < \alpha_\theta$.
3  repeat
4      Sample count statistics $\mathbf{n}_{t=1:T}$
5      for agent $i = 1$ to $N$ do
6          Compute individual value for agent $i$ using Eq. 2.14–Eq. 2.17
7          Update policy parameters $\theta$ using Eq. 2.12
8          Update $\lambda_k, \forall k$ using Eq. 2.20
9      end
10 until convergence
11 return $\theta$
```

We re-write the objective in the collective way as follows.

$$
\begin{aligned}
=&\lambda_k(\frac{c_k}{N} - \sum_{s' \in S_k} \sum_{t=1}^{T} \sum_{\boldsymbol{s}^i_{1:t}, \boldsymbol{a}^i_{1:t}, \mathbf{n}_{1:t}} \mathbb{I}(s^i_t = s') P(\boldsymbol{s}^i_{1:t}, \boldsymbol{a}^i_{1:t}, \mathbf{n}_{1:t}) \frac{C_k(\mathbf{n}_t[S_k])}{|\mathbf{n}_t[S_k]|}) \\
=&\lambda_k(\frac{c_k}{N} - \frac{1}{N}\mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t[S_k])|\pi_\theta]) \\
=&\frac{\lambda_k}{N}(c_k - \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t[S_k])|\pi_\theta]) \qquad (2.19)
\end{aligned}
$$

By applying gradient descent, we have.

$$
\lambda'_k = \max\Big(\lambda_k - \alpha_k \frac{1}{N}(c_k - \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t[S_k])|\pi_\theta]), 0\Big) \qquad (2.20)
$$

where α_k is the learning rate for the update of λ_k.

The pseudocode for our approach FICO is provided in Algorithm 2.1.

2.4 Experiments

In the section, we evaluate our proposed approach FICO on two real-world tasks: Maritime traffic management (MTM) and vehicular network routing problem with a large scale of agent population. For MTM problem, we compare FICO with two baseline approaches LR-MACPO and LR-MACPO+. LR-MACPO is a Lagrangian based approach without any credit assignment. The policy in this case is trained with global reward and global cost signal, similar to RCPO algorithm [106]. LR-MACPO+ is also a standard Lagrangian based approach with credit assignment only for the reward signal but not for the cost function in constraint. The detailed problem formulations for LR-MACPO and LR-MACPO+ are as follows. We compare FICO with CMIX [68] in the vehicular network routing problem. Since CMIX only deals with discrete action space, we did not evaluate CMIX in the MTM problem where the action space is continuous.

2.4.1 Formulation for LR-MACPO and LR-MACPO+

In LR-MACPO , there is no credit assignment technique. Hence, the constrained optimization problem is given as follows,

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T}\sum_{m=1}^{M} R_m(\mathbf{n}_t)|\pi_\theta] \tag{2.21a}$$

$$\text{s.t.} \quad \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t)|\pi_\theta] \leq c_k, \qquad \forall k \tag{2.21b}$$

The Lagrangian is,

$$\mathcal{L}(\pi_\theta, \lambda) = \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T}\sum_{m=1}^{M} R_m(\mathbf{n}_t)|\pi_\theta] - \sum_{k=1}^{K}\lambda_k(\mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t)|\pi_\theta] - c_k) \tag{2.22}$$

where $\lambda_k \geq 0, \forall k$. The update for θ and λ_k are as follows,

$$\theta' = \theta + \alpha_\theta \nabla_\theta \mathcal{L}(\pi_\theta, \lambda) \tag{2.23}$$

$$\lambda'_k = \max\Big(\lambda_k - \alpha_k \nabla_{\lambda_k}\mathcal{L}(\pi_\theta, \lambda), 0\Big) \qquad \forall k \tag{2.24}$$

where α_θ and α_k are the learning rates for updating of θ and λ_k respectively.

LR-MACPO+ performs credit assignment only to reward signal but not cost signal. The constrained optimization problem is given as follow,

$$\max_{\theta} \quad \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T}\sum_{m=1}^{M} R^i_m(s^i_t, a^i_t, \mathbf{n}_t[S_m])|\pi_\theta] \tag{2.25a}$$

$$\text{s.t.} \quad \mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t)|\pi_\theta] \le c_k, \qquad \forall k \tag{2.25b}$$

The Lagrangian is,

$$\mathcal{L}(\pi_\theta, \lambda) = \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T}\sum_{m=1}^{M} R^i_m(s^i_t, a^i_t, \mathbf{n}_t[S_m])|\pi_\theta] - \sum_{k=1}^{K} \lambda_k(\mathbb{E}_{\mathbf{n}_{1:T}}[\sum_{t=1}^{T} C_k(\mathbf{n}_t)|\pi_\theta] - c_k) \tag{2.26}$$

where $\lambda_k \ge 0, \forall k$. The update for θ and λ_k are as follows,

$$\theta' = \theta + \alpha_\theta \nabla_\theta \mathcal{L}(\pi_\theta, \lambda) \tag{2.27}$$

$$\lambda'_k = \max\Big(\lambda_k - \alpha_k \nabla_{\lambda_k} \mathcal{L}(\pi_\theta, \lambda), 0\Big) \qquad \forall k \tag{2.28}$$

where α_θ and α_k are the learning rates for updating of θ and λ_k respectively.

2.4.2 Maritime traffic management

The main objective in the MTM problem is to minimize the travel delay incurred by vessels while transiting busy port waters and also to reduce the congestion developed due to uncoordinated movement of vessels. The previous formulations of the MTM problem in [99, 100] involved unconstrained policy optimization—the objective is a weighted combination of the delay and congestion costs. This requires an additional tuning of the weight parameters in the objective. We propose a new MTM formulation using constrained ℂDec-POMDP as in (2.1). In our formulation, we introduce constraints that the cumulative congestion cost should be within a threshold for each zone and minimize the travel delay as the main objective. The new formulation with constraint is more interpretable, and avoids the intensive search over weight parameters to formulate a single objective as in previous models [100]. We evaluate our constrained based approach of the MTM problem in both synthetic and real-data instances. In synthetic data experiments we test the scalability and robustness of our proposed algorithm. Real-data instances are used to measure the effectiveness of the approach in a real-world problem.

MTM formulation Let Z denote the set of all zones. We penalize each vessel for not reaching its destination zone at time step t with a penalty 1, i.e., $R^i_t = -1$. Let $\mathbf{n}_t$ denote the count statistics at time t. The reward at time step t is then defined as,

$$R(\mathbf{n}_t) = \sum_{z\in Z} R_z(\mathbf{n}_t) = \sum_{z\in Z} - \mathrm{n}^{\mathrm{tot}}_t(z) \tag{2.29}$$

where $\mathrm{n}^{\mathrm{tot}}_t(z)$ denotes the total number of vessels in zone z at time t.

We consider that each zone z has a capacity c_z. The cost for zone z at time step t is defined as,

$$C_z(\mathbf{n}_t) = \max(\mathrm{n}^{\mathrm{tot}}_t(z) - c_z, 0) \tag{2.30}$$

The cost function reflects the resource violation at time t. The constraint for zone z is defined as the cumulative resource violation over time horizons is within a predefined threshold β_z. Formally, we have,

$$\mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}\Big[\sum_{t=1}^{T} C_z(\mathbf{n}_t)|\mathbf{n}_{1:T} \sim (\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}), \pi_\theta\Big] \le \beta_z \tag{2.31}$$

The MTM problem is formulated as:

$$\max_{\theta} \quad \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}\Big[\sum_{t=1}^{T} R(\mathbf{n}_t)|\mathbf{n}_{1:T} \sim (\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}), \pi_\theta] \tag{2.32a}$$

$$\text{s.t.} \quad \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{H} C_z(\mathbf{n}_t)|\mathbf{n}_{1:T} \sim (\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}), \pi_\theta] \le \beta_z, \qquad \forall z \tag{2.32b}$$

The individual constrained problem is defined for each zone z as follows,

$$\max_{\theta} \quad \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T} - \mathrm{n}^{\mathrm{tot}}_t(z)|\mathbf{n}_{1:T} \sim (\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}), \pi_\theta] \tag{2.33a}$$

$$\text{s.t.} \quad \mathbb{E}_{\boldsymbol{s}_{1:T},\boldsymbol{a}_{1:T}}[\sum_{t=1}^{T} \max(0, \mathrm{n}^{\mathrm{tot}}_t(z) - c_z)|\mathbf{n}_{1:T} \sim (\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}), \pi_\theta] \le \beta_z \tag{2.33b}$$

Synthetic data instances For synthetic data experiments, we first randomly generate

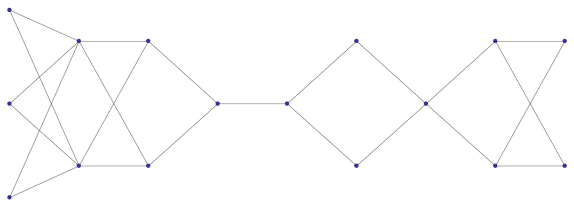
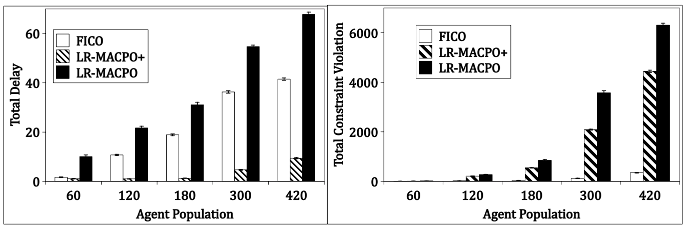


Figure 2.2: Synthetic instance map with 23 zones



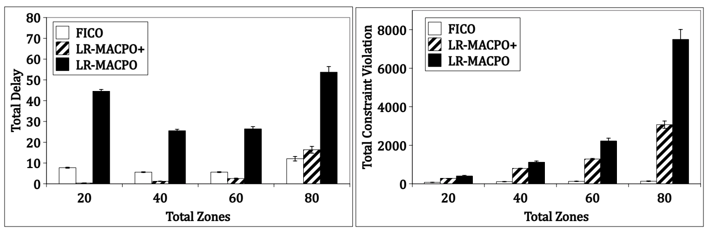
(a) Total delay **(b)** Total constraint violation

Figure 2.3: Results on the map with 23 zones and different agent population. The lower is the better for both metrics.

directed graphs similar to the procedure described in [99]. Figure 2.2 shows a generated directed graph with total 23 zones. The edge of the graph represents a zone, vessels move from left to right through the zones. Each zone has some capacity i.e the maximum number of vessels the zone can accomodate at any time. Each zone is also associated with a minimun and maximum travel time to cross the zone. Vessels arrive at the source zone following an arrival distribution, and its next heading zone is sampled from a pre-determined distribution.

Over all experiments, the maximum capacity of each zone is uniformly sampled between [5,10]. The arrival time for vessels at source zones is also uniformly sampled between [1,20]. Minimum and maximum travel time for source zones are 1 and 30. Minimum travel time for planning zones is 1, and maximum travel time is randomly sample between [5, 10]. The total time horizon is 100 time steps.

We first evaluate the scalability of our approach with varying agent population size from 60 agents to 420 agents on the map with 23 zones. We show the results on total delay and total constraint violation respectively as in Figure 2.3(a) and Figure 2.3(b).

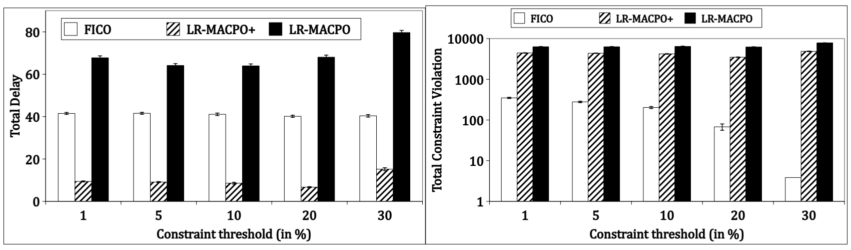


(a) Total delay **(b)** Total constraint violation

Figure 2.4: Results on agent population 300 and different maps

Delay is computed as the difference between actual travel time and minimum travel time in the zone. Total violation computes the total constraint violations over all zones. X-axis denotes the agent population size in both the subfigures, and y-axis denotes the total delay and total constraint violation in (a) and (b) respectively. We observe that LR-MACPO baseline performs poorly than other approaches in terms both the metric of delay and violation. This is because LR-MACPO is trained with global system reward and global cost function, which is without any credit assignment technique. LR-MACPO+'s performance on delay metric is superior than our approach FICO, but it suffers severely on violation metric in Figure 2.3(b). This is an expected result because in LR-MACPO+ credit assignment is provided only for the reward signal not for the cost signal. This makes each agent's credit for the cost component to be noisy resulting in ineffective handling of the constraints. Also, low delay benefits from the high constraint violation in LR-MACPO+.

We next evaluate the robustness of our approach with different maps with varying number of zones (from 20 zones to 80 zones). As in Figure 2.4, x-axis denotes different maps with varying number of zones in both the subfigures, and y-axis denoted the total delay and total constraint violation in (a) and (b) respectively. In this experiment the complexity of the problem increases with the increasing number of zones. We observe that our approach FICO is able to reduce the violation consistently for all the settings as shown in Figure 2.4(b). In settings with less than 80 zones, LR-MACPO+ beats our approach in terms of total delay. However, it fails to satisfy the constraints poorly. We see that at the most difficult setting with 80 zones, our approach performs better than LR-MACPO+



(a) Total delay (b) Total constraint violation

Figure 2.5: Results on agent population 420 and 23 zones with different constraint threshold

in terms of both total delay and constraint violation.

Finally, we evaluate the robustness of our approach with different constraint thresholds on the map with 23 zones and 420 agents. The constraint threshold specifies the upper bound of cumulative resource violation over the horizons. The constraint threshold is defined as a percentage of the total resource violations over the horizons when agents are moving with the fastest speed. As shown in Figure 2.5, x-axis denotes different constraint thresholds in both the subfigures, and y-axis denotes the total delay and total constraint violation in (a) and (b) respectively. We observe that FICO performs better than LR-MACPO baseline in terms of both the metric of delay and constraint violation. In Figure 2.5(b), we see that FICO performs better than other baselines consistently over different constraint thresholds. With the increase of constraint threshold, the total constraint violation is decreasing. FICO is almost able to make the constraint satisfied with the loosest constraint threshold (30%). The constraint violation comes from the zone in the middle of the map which is the busiest zone.

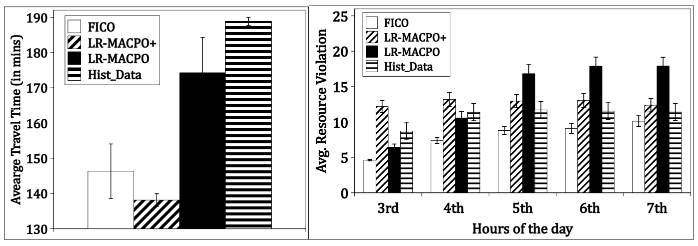
Real data instances We also evaluated our proposed approach on real-world data instances from Singapore strait. The strait is considered to be one of the busiest in the world. It connects the maritime traffic of South China Sea and Indian Ocean. We use 6 months (2017-July - 2017-December) of historical AIS data of vessel movement in Singapore strait. Each AIS record consists of vital navigation information such as lat-long, speed over ground, heading etc, and is logged every 15 seconds. In our evaluation we mainly focus on tankers and cargo vessel types because majority of traffic belongs to these two types.

Figure 2.8(a) shows the electronic navigation chart of Singapore strait. Vessels enter and leave the strait through one-way sea lanes called traffic separation scheme (TSS). It is created for easy transit of vessels in the strait and helps in minimizing any collision risks. TSS is further sub-divided into smaller zones for better management of the traffic. From the total datasets of 6 months (180 days), 150 days are used for training and 30 days for testing.

Training From the historical data, we first estimate the problem instance parameters such as capacity of each zones, minimum and maximum travel time in each zone. The simulator that we use is the same as in [100], and is publicly available at [98].

The capacity of a zone is computed as 60% of the maximum number of vessels present at any time in the zone overall all days. Each zone can have a different capacity value. We treat the physical sea space in a zone as a resource. Each vessel occupies 1 unit of resource of that zone. The constraint for each zone is expressed as the cumulative resource violation over time should be within a threshold. There are also other problem parameters which are specific to a particular day such as vessels' arrival time on the strait and initial count distribution of vessels present at the strait in beginning of the day. For each training day we estimate the two parameters. Our constrained based policy FICO is trained on varying scenarios of training days. From historical data we observe that there are peak hour periods of traffic intensity during 3rd - 7th hour of the day. Therefore, in our evaluation we focus on optimizing the peak hour periods.

Testing We test our trained policy on separate 30 testing days. Figure 2.6(a) shows the results of average travel time of vessels crossing the strait averaged over 30 test days. We observe that all the three baselines achieve better travel time than the historical data baseline Hist-Data. LR-MACPO performs poorly among the three. This is because LR-MACPO is trained with the system reward and cost signals which are without any credit assignment techniques. We also observe that LR-MACPO+ baseline is able to further reduce the travel time slightly better than our approach FICO but at the expense of higher resource violation as seen in Figure 2.6(b).



(a) Average travel time (b) Average resource violation

Figure 2.6: Results on maritime real-data over 30 testing days. (lower is better for both metrics)

.

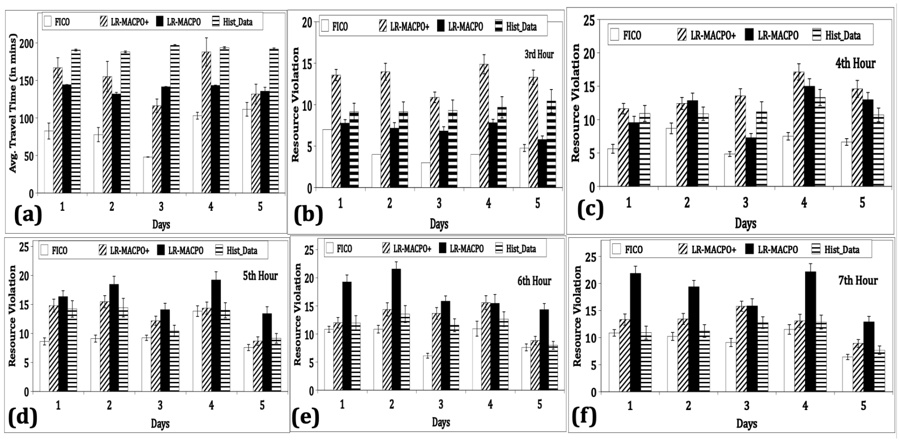


Figure 2.7: (a) Average travel time(lower is better) (b-f) Resource violation for peak hours 3rd-7th (lower is better)

Results in Figure 2.6(b) show the average violation of resource over 30 testing days. X-axis denotes the peak hour periods. During the peak hour period, FICO achieves reduced violation of resource among all the baselines. Since LR-MACPO+ lacks the credit assignment signal on the cost function it performs sub-optimally than FICO. The results in Figure 2.6 validate the benefit of providing efficient credit assignment technique to both reward and cost function.

In Figure 2.7 we show the results of top 5 busiest testing days. Figure 2.7(a) shows the results for travel time, x-axis denotes days and y-axis denotes average travel time for crossing the strait. Figure 2.7(b-f) show the results of resource violation during peak hour periods (3rd-7th hour), x-axis denotes the days and y-axis denotes the resource

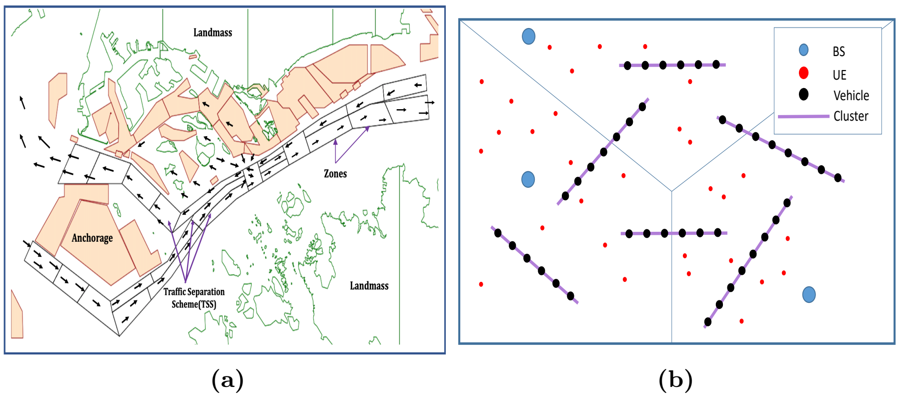


Figure 2.8: (a) Electronic navigation chart(ENC) of Singapore Strait; (b)Vehicular network model (adapted from [59])

violation. In all 5 days and during the peak hour periods, our approach achieves an improved reduction in violation of resource while also reducing the travel time better than other baselines.

2.4.3 Vehicular network routing problem

We compare our approach with CMIX [68] in their cooperative vehicular networking problem, which is their largest tested domain. Figure 2.8(b) shows a network model with three cells and six clusters. There are two types of cluster - inter cluster (between two cells) and intra cluster. Each cluster contains well connected vehicles that can communicate with high throughput via V2V (Vehicle-to-Vehicle) links. The base station (BS) cell is shared by other mobile user equipments (UEs) and can communicate with vehicles via the direct V2I (Vehicle-to-Infrastructure) links. In this paper, we consider the problem of downlink data transmission where the data are transmitted from BSs to vehicles in the clusters. The objective for all vehicles/agents here is to find the network routes such that the total throughput is maximized (i.e., delivering high volumes of data to destination vehicles), while satisfying both the peak and average latency constraints. Peak constraint means that the latency due to the execution of an agent's action at any time step should be bounded. In CMIX, each agent requires an individual policy to perform action selection. In contrast, agents that belong to the same cluster share the

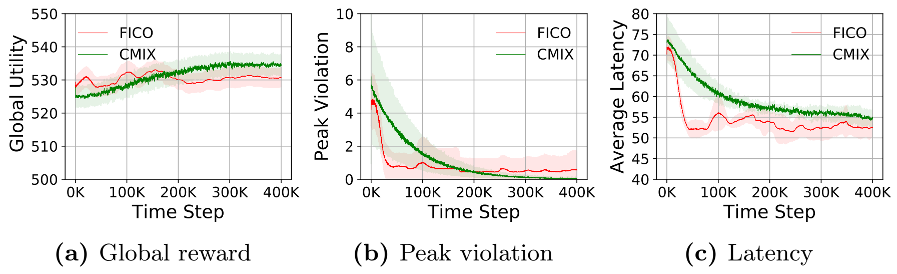
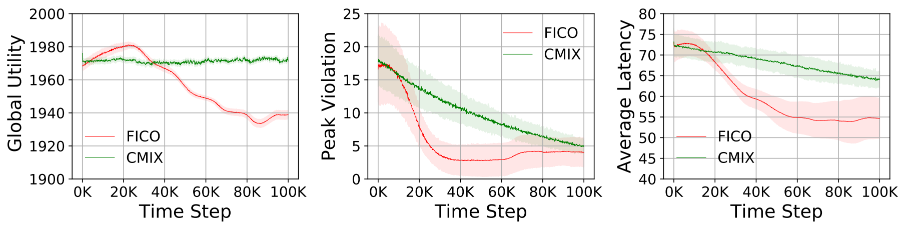


Figure 2.9: Convergence results over time steps with total 30 $\sim$ 60 agents

same policy in our collective method. Time limit is set to 180 mins.

We first follow the same experimental settings as in the CMIX paper to evaluate the performance. There are total three cells and six clusters that are randomly distributed over cells. The number of vehicles in each cluster is randomly generated between a range $[5, 10]$ so that there will be total 30 $\sim$ 60 vehicles. The throughput and latency of these V2V and V2I links are also randomly generated. Figure 2.9 shows the learning curves of global reward, peak violation and latency over time steps. The average latency over time steps in one episode is bounded by 60. The threshold for peak constraint is also 60. From Figure 2.9(c), we can see that the average constraint is satisfied when convergence occurs in both approaches. However, our approach converged much faster and is more sample efficient than the CMIX. In Figure 2.9(b), the peak constraint is almost satisfied in our approach. Only one agent's latency is greater than 60 in average. The reason is that we use shared policy for agents in the same cluster. And it is challenging for the policy to consider each agent's peak constraint. In Figure 2.9(a), the global rewards in both approaches are increasing (higher is better), and converged to almost the same values (530 in our approach v.s. 535 in CMIX). It shows that our approach is also able to maximize the delivered volumes of data.

We next evaluate the scalability of our approach. We increase agents in each cluster to $[5, 10]$; total number of agents are between 150 and 180. CMIX is only able to train around 100K steps within the limit and our approach can finish 400K steps. Therefore, we show the learning process over 100K steps. Figures 2.10(b) and (c) show that peak



(a) Global reward **(b)** Peak violation **(c)** Latency

Figure 2.10: Convergence results over time steps with total 150 $\sim$ 180 agents

violation and average latency are decreasing in both two approaches. However, the average latency and peak violation in FICO are decreasing with a much faster speed than CMIX. Also, our approach is able to find a policy to satisfy the latency constraint within 100K steps, confirming the effectiveness of our method for large scale problems. The average latency in CMIX is still greater than the threshold 60. The global rewards over time steps are almost unchanged in CMIX, and decreased slightly from 1970 to 1940 in our approach as our approach results in lower constraint violations versus CMIX.

Chapter 3

Zone-based Multi-agent Pathfinding

The chapter introduces a framework called Zone-Based Multi-Agent Pathfinding (ZBPF), which addresses the multi-agent sequential decision making problem with constraints, specifically motivated by the drone delivery application. The ZBPF framework extends the standard Multi-Agent Pathfinding (MAPF) model, which has practical applications in various domains such as game character movements [73], warehouse logistics [113], robotics [121], and vessel routing in maritime traffic [105]. In the previous chapter, we presented the Constrained Collective Dec-POMDP model for large-scale multi-agent systems. In this chapter, our focus is on a variant of the standard Dec-POMDP model that is well-suited for the ZBPF framework. The standard multi-agent pathfinding (MAPF) model is first introduced in Section 3.1, and its three key limitations are identified. The ZBPF model is presented in Section 3.2, which features zones with capacities where agents move and uncertain travel times. The chapter then presents a novel difference-of-convex functions (DC) programming approach in Section 3.3 to solve the policy optimization problem in ZBPF. The goal is to minimize travel time for agents from their respective sources to destinations while satisfying the zone capacity constraints. A multi-agent credit assignment scheme is also proposed to speed up the learning approach's convergence. Finally, the chapter evaluates the effectiveness of the approach in reducing constraint violations in zones while ensuring agents reach their final destinations through 2D and 3D instances in Section 3.4.

3.1 Relation to standard multi-agent pathfinding

The problem of standard multi-agent pathfinding (MAPF) entails multiple agents finding collision free paths in a collaborative fashion from their respective sources to destination nodes in a graph [38, 73]. Agents can only choose to stay at the current node or move to one of the adjacent nodes. Collision free means that two (or more) agents cannot be present in a node at the same time. There are multiple objectives possible in MAPF including minimizing sum of arrival time of all the agents at their destination nodes, or the makespan, which is the arrival time of the agent that reaches its destination last. Both these variants are known to be NP-hard [121]. Various solution approaches (both approximate and optimal) have been developed for MAPF. Reduction based approaches translate MAPF to other well known problems such as SAT [101, 102], integer programming [121], and constraint satisfaction [111] among others. There are several approaches that take a search based perspective of MAPF and develop multi-agent search algorithms such as conflict based search [92, 91] and its variant [16]. Reinforcement learning based approaches are also developed for MAPF [86].

In the standard MAPF formulation, there are three key assumptions: (1) each graph node can only accommodate at most one agent at a time; (2) agents move at a fixed uniform speed; (3) there is no movement uncertainty. However, these assumptions do not hold any more in real-world applications e.g., drone delivery. Several recent attempts aim to generalize standard MAPF to realistic settings. Based on resource constrained activity scheduling, the MAPF problem has been extended to incorporate nodes/edges with higher capacity more than one agent, and non-uniform travel time for different edges [6, 102]. However, this problem setting is still deterministic as there is no uncertainty in the movement of agents.

To handle movement uncertainty and imperfect plan execution by agents, delay probabilities are introduced in [74] in which an agent's movement to the next node succeeds with probability p, and with $1-p$ probability agent stays at its current node. Such agent movement essentially follows a geometric distribution, which may not be general enough for several practical problems. Their solution strategy relies on modifying the plan execution during run time to avoid collisions, rather than using a full multi-agent planning model such as a decentralized partially observable MDP (Dec-POMDP). Another limitation is that their approach considers capacity of each node as one agent. Another robust plan execution strategy is presented in [53]. Their approach post-processes the output of a MAPF solver to create a schedule that can be executed safely (i.e., collision-free) by (robotic) agents. Their approach considers minimum and maximum time needed by an agent to move along a graph edge, and ensures that the post-processed output is safe for all agent movements within the allowed range. Their approach also considers nodes having the capacity of one agent. A velocity planner is developed in [100]. Their approach addresses uncertainty in movement, but is limited to optimizing only the temporal aspect of movement (not the spatial movement of agents), and assumes that all the agents are homogeneous.

To summarize, there are generalizations of MAPF that consider graph nodes and edges having capacity of more than one agent, but may not model uncertainty in agents' movement, and there are other generalizations that model movement uncertainty, but assume nodes and edges have a capacity of one agent. Our zone based multi-agent pathfinding framework addresses the three key limitations of the standard MAPF by allowing for nodes (or zones) with capacity of more than one agent, model uncertainty (by using the highly expressive class of exponential family distributions to model variable duration movements of agents), and partial observability wherein agents take decisions based on their local observations.

3.2 Zone based pathfinding model

We next present a model for ZBPF by using a variant of the constrained Dec-POMDP model, which is a popular framework for multi-agent sequential decision making under uncertainty and partial observability. The set of all zones z is Z. Given a directed graph $G = (V, E)$, each node is a zone, and edges represent connections between neighboring zones. We assume that each zone has a capacity c_z and a constraint that the maximum number of agents present in the zone at a time cannot exceed its capacity. If more agents

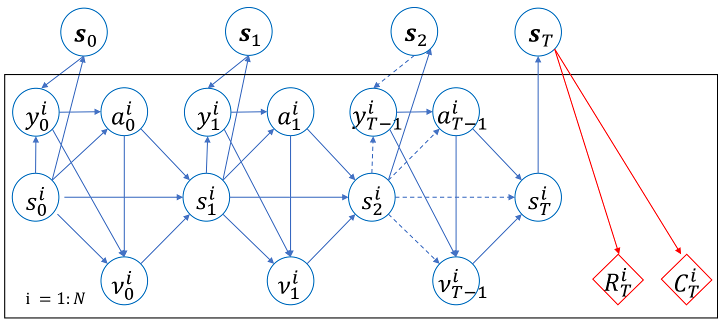


Figure 3.1: Dynamic Bayes Net (DBN) for T-step reward and cost

are present in a zone than its capacity, it creates congestion affecting safety of movement. Crossing a zone z to a neighboring zone z' requires a minimum and maximum travel time–$t_{\text{min}}, t_{\text{max}}$. Our approach can also handle the case when each agent has a unique $t_{\text{min}}, t_{\text{max}}$ for each zone intersection, but for simplicity, we assume all zone movements have the same lower/upper limits. We also assume time is discretized, and horizon is T.

Agents' decision model We now describe how an agent's decision making evolves in ZBPF. Instead of describing a specific formulation applied only to ZBPF, we develop a more general decision theoretic model (in Figure 3.1) of which ZBPF is a specific instance. Figure 3.1 describes graphically how different agents interact with each other using a dynamic Bayes net (and plate notation). The arrows in the dynamic Bayes net represent how the state transition, observation, reward, and cost functions of an agent are influenced by both the global and local state, as well as the agent's previous action. Detailed descriptions of these parameters follow next. Similar to MAPF, an agent i has a starting zone z_o^i, and a goal zone z_g^i (multiple agents are allowed to share their start and goal zones).

States We model the behavior of each agent as follows. Let s_t^i denote the state of agent i at time t. Consider an agent i currently inside a zone z. Since the navigation between zones has a variable duration (as described later), this agent can be categorized as *newly arrived* at some zone z at time t or *in-transit* through z at time t.

- [In-Transit] The state of an agent i is defined as a tuple $s_t^i = \langle z, z', \tau \rangle$, where z

denotes the current zone of agent i at time t; z' denotes the next zone agent is heading to; and τ is the remaining time to reach z'.

- [Newly Arrived] When an agent i newly arrives at zone z, its next zone z' and the time-to-reach z' (say τ) are not yet determined (which will be determined by the agent's action sampled from the policy). The state s_t^i is denoted as $\langle z, \Phi, \Phi \rangle$.

Actions The action set of an agent is $A = \{\langle z, \nu \rangle \ \forall z \in V, \nu \in \Re^d\} \cup \{\text{noop}\}$. In a given state s_t^i, only some of the actions are valid.

- If agent is in-transit, $s_t^i = \langle z, z', \tau \rangle$, only valid action is noop.
- If agent is newly arrived at a non-goal zone $z \in Z \setminus \{z_g^i\}$, or $s_t^i = \langle z, \Phi, \Phi \rangle$, then valid action set is $A = \{\langle z, \nu \rangle \ \forall z \in \text{Nb}(z), \nu \in \Re^d\}$, where $\text{Nb}(z)$ is the set of neighbor zones of z where agent can move to. There are additional conditions on possible parameter ν as explained in the transition function.
- If agent is at the start of a goal zone, or $s_t^i = \langle z_g^i, \Phi, \Phi \rangle$, then the only valid action is noop.

Intuitively, for an agent at the start of a non-goal zone z, two action components are needed–$\langle z', \nu \rangle$. First, the agent decides its next destination $z' \in \text{Nb}(z)$ to move to. Second, the agent must decide the speed at which it wants to move. To avoid modeling the control dynamics of agents, we assume that the agent decides the time to take to move to z'. This temporal movement part is controlled by the continuous action component ν (transition function shows a more precise definition). Thus, the model we present contains both discrete and continuous actions, which represents a more challenging setting than standard planning or RL.

Transition function We assume that the movement uncertainty is captured by the transition function. Instead of assuming a specific movement model such as geometric distribution [74], we chose to model travel time uncertainty using a widely applicable yet tractable class of exponential family distributions. Give parameter(s) η, the exponential family of probability distributions is defined as those distributions whose pdf or pmf have the following general form [13, Chapter 2]:

$$p(x|\eta) = h(x)\exp\{\eta^T T(x) - \mathcal{A}(\eta)\} \tag{3.1}$$

The function $T(x)$ is the *sufficient statistic*, η is the *canonical parameter*, and $\mathcal{A}(\eta)$ is the partition function (to make sure probabilities normalize to 1). Using this form, we now frame the transition function of the model in Figure 3.1 as:

$$p(s'^i|s^i, a^i, \nu^i) = f(s^i, a^i, s'^i)\exp\{\nu^i \phi(s^i, s'^i) - \mathcal{A}(\nu^i)\} \tag{3.2}$$

where f is non-negative (plays the role of $h(x)$); $\phi(s^i, s'^i)$ is the sufficient statistic that summarizes the transition information. a^i is the discrete action (analogous to the direction decision in ZBPF), and ν^i is the canonical parameter of the exponential family distribution that governs the transition function of each agent i.

The above representation of the transition function is quite general; it can represent standard categorical distribution (which are widely used as transition function for MDPs), as well as distributions such as Gaussian. The exponential family distributions are also useful to model travel times because they are the maximum-entropy distribution consistent with given constraints (or prior information) about the underlying random variables. As per the principle of maximum entropy, among the distributions that satisfy given constraints, the least informative one must be chosen [56]. E.g., given that we have fixed time limits t_{min} and t_{max} on the movement of an agent between zones, an agent may decide to move in α time steps. The maximum entropy discrete distribution with a given mean (say α) and bounded support (between t_{min} and t_{max}) is the binomial distribution [51], which models the uncertainty in agent movement. In this case, the success probability of the binomial distribution can be computed using ν. We can also similarly interpret the geometric distribution used by [74]. In their approach, an agent's move action succeeds with (a given) probability p. In other words, the average time to cross an edge is $\alpha = 1/p$. The maximum entropy distribution with a given mean is indeed the geometric distribution[1]. Similarly, if time is continuous, then the agent's action ν can be interpreted as the mean of the Gaussian distribution which models the travel duration (we can assume that the agent cannot control the variance, which is known).

To summarize, given an action $\langle z', \nu \rangle$, the first action component denotes the *direction decision* (the zone z') where agent wants to go next, and ν is the *canonical parameter* of the transition function (assuming the transition function is from the exponential family). Specific transition function for ZBPF is given as:

- For a newly arrived agent i, let $s^i_t = \langle z, \Phi, \Phi \rangle$, action taken $\langle z', \nu \rangle$ (which implies agent has decided to go to zone z'), then the time taken to move to z' is sampled from the distribution $\tau \sim p^i_{\text{dur}}(\cdot; \nu, z')$, where p^i_{dur} is an exponential family distribution with finite support between t_{min} and t_{max} (that is, $\tau \in \{t_{\text{min}}, \ldots, t_{\text{max}}\}$). If $\tau = 1$ (i.e., only one time step to-go), then the next state is $\langle z', \Phi, \Phi \rangle$ (or agent reaches the next zone z' at time $t+1$), therefore, $p^i(\langle z', \Phi, \Phi \rangle | \langle z, \Phi, \Phi \rangle, \langle z', \nu \rangle) = p^i_{\text{dur}}(\tau; \nu, z')$. If $\tau > 1$, the next state is $\langle z, z', \tau - 1 \rangle$, and $p^i(\langle z, z', \tau - 1 \rangle | \langle z, \Phi, \Phi \rangle, \langle z', \nu \rangle) = p^i_{\text{dur}}(\tau; \nu, z')$.

- For in-transit agent i, let $s^i_t = \langle z, z', \tau \rangle$, then the only action allowed is noop. If $\tau = 1$, then the state deterministically transitions to $\langle z', \Phi, \Phi \rangle$. If $\tau > 1$, then the state deterministically transitions to $\langle z, z', \tau - 1 \rangle$. That is, the agent has not yet completed its movement action, and is still in zone z en-route to z'.

We treat an agent's goal zone as an absorbing state without any outgoing transitions. For simplicity, we assume well-formed infrastructures where an agent occupying its goal zone does not obstruct other agents from finding their paths [21].

Observation function Each agent i observes y^i_t based on its local state s^i_t, and the global state of all agents $\boldsymbol{s}_t$. For simplicity, we assume a deterministic observation function, which can include observation about all agents present in the same zone as agent i and in neighboring zones.

Reward function There is a negative reward $w_d < 0$ given to each agent for every time

[1] https://en.wikipedia.org/wiki/Exponential_family

step the agent is not at its goal location. When the agent reaches its goal zone for the first time, a positive reward w_g is given. Total system reward R is sum of rewards for all the agents (or $R_t = \sum_{i=1}^{N} R_t^i$).

Cost function In a global state $\boldsymbol{s}$, if the total number of agents in a zone z are more than its capacity c_z, then each agent in zone z receives a cost of $w_c > 0$ for constraint violation. Total system cost C is sum of costs for all the agents (or $C_t = \sum_{i=1}^{N} C_t^i$).

Policy representation Many practical applications of reinforcement learning require agents to select actions from discrete-continuous hybrid spaces [43, 63]. In our ZBPF, we also consider that an agent i has a hybrid policy. First, the policy $\pi^i(a_t^i|s_t^i, y_t^i)$ provides a distribution over discrete actions a (discrete actions are analogous to the direction decisions in ZBPF). Thus π^i is a stochastic policy. We also have a *deterministic* policy μ^i which provides the (often continuous) action component $\nu_t^i = \mu^i(s_t^i, y_t^i, a_t^i)$. These policies can also be parameterized using parameters θ.

3.3 Objective function and DC programming

We first go over the difference-of-convex functions (DC) programming framework [123, 67]. Our goal would be to reformulate the objective function as an instance of DC programming. The DC programming and *concave-convex procedure* (CCCP) [123] are a popular approach to optimize a general non-convex function expressed as a *difference* of two convex functions. We describe it here briefly. Consider the optimization problem:

$$\min\{g(x) : x \in \Omega\} \tag{3.3}$$

where $g(x) = u(x) - v(x)$ is an arbitrary function with u , v being real-valued *convex* functions and Ω being a convex set. The CCCP method provides an iterative procedure that generates a sequence of points x_k by solving the following convex program:

$$x_{k+1} = \arg\min\{u(x) - x^T \nabla v(x_k) : x \in \Omega\} \tag{3.4}$$

Each iteration of CCCP decreases the objective $g(x)$ and is guaranteed to converge to a local optimum [67]. The above formulation is derived by linearizing the convex function v at the point x_k as: $\hat{v}(x) = v(x_k) + \nabla v(x_k)^T(x - x_k)$, and exploiting the property that linearizing a convex function results in a global (valid over the entire domain of function) lower bound on v. The key benefit of CCCP is that Problem (3.4) is often much easier to solve than the original Problem (3.3) as the objective in (3.4) is convex (first term is convex, and second term is linear in x).

We now show how to exploit the DC for our ZBPF problem. Consider the DBN in Figure 3.1. A joint trajectory is denoted as $\tau = \boldsymbol{s}_0, \boldsymbol{y}_0, (\boldsymbol{a}_0, \boldsymbol{\nu}_0), (\boldsymbol{R}_0, \boldsymbol{C}_0), \boldsymbol{s}_1, \boldsymbol{y}_1, (\boldsymbol{a}_1, \boldsymbol{\nu}_1), (\boldsymbol{R}_1, \boldsymbol{C}_1), \ldots$ where subscripts denote time. For deriving the DC formulation, we start by considering policy π^i for each agent is tabular, and μ^i is a deterministic function. Let π and μ denote the joint-policy. We also assume deterministic observations (as highlighted earlier).

The objective to optimize is:

$$J(\pi, \mu) = \sum_{T=0}^{\infty} \sum_{\tau_{0:T}} \gamma^T R(\boldsymbol{s}_T) b_0(\boldsymbol{s}_0) \pi(\boldsymbol{a}_0 | \boldsymbol{s}_0, \boldsymbol{y}_0) \prod_{t=1}^{T} \Big(p(\boldsymbol{s}_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \mu(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{a}_{t-1})) \pi(\boldsymbol{a}_t | \boldsymbol{s}_t, \boldsymbol{y}_t) \Big) \tag{3.5}$$

where we assumed that the reward is discounted using γ, b_0 denotes the initial state distribution.

The zone capacity constraints to satisfy during optimizing the above objective are as follows.

$$\sum_{i=1}^{N} \mathbb{I}(s_t^i = \langle z, \cdot, \cdot \rangle) \leq c_z, \quad \forall z, t \tag{3.6}$$

where $\mathbb{I}(s_t^i = \langle z, \cdot, \cdot \rangle)$ is an indicator function. Its value is 1 when agent i is present in zone z at time step t; otherwise, the value is 0.

When considering movement uncertainty, it becomes impractical to find a path for agents that guarantees no collisions. In MAPF, this constraint is required to be satisfied completely, leading to very conservative solutions and high sum of arrival times for all agents. Therefore, in our work, we treat the zone capacity constraints as soft constraints, aiming to reduce constraint violations as much as possible. Instead of using Lagrange multipliers, which are difficult to learn, to penalize constraint violations, we use a fixed cost $w_c > 0$ to penalize constraint violations and to encourage coordination among agents. To achieve this, we modify the original reward R_t^i for each agent i by deducting the cost C_t^i, i.e., $R'^i_t = R_t^i - C_t^i,\ \forall i, t$. This modification allows us to reduce the negative impact of constraint violations while still considering them during the optimization process.

Without loss of generality, we also assume modified rewards are non-negative (if they are not, we can use $R' - R'_{\text{min}}$ instead of R'). Exploiting the DBN structure in Figure 3.1, we can factorize the joint transition function and modified reward function to get $J(\pi, \mu)$ as:

$$J(\pi,\mu) = \sum_{T=0}^{\infty}\sum_{\tau_{0:T}} \gamma^T R'(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{i=1}^{N}\pi^i(a_0^i|s_0^i,y_0^i)\Big)\prod_{t=1}^{T}\bigg(\Big(\prod_{i=1}^{N} p^i(s_t^i|s_{t-1}^i, a_{t-1}^i, \mu^i(s_{t-1}^i, y_{t-1}^i, a_{t-1}^i))\Big)\Big(\prod_{i=1}^{N}\pi^i(a_t^i|s_t^i,y_t^i)\Big)\bigg) \tag{3.7}$$

The Equation (3.7) is our objective to maximize by optimizing joint-policies (π, μ). The above objective is non-convex. It is also not expressed in the DC form. To reformulate the above objective into a DC form, we first make a number of variable substitutions as below.

First, we substitute $\pi^i(a^i|s^i, y^i)$ with $\exp\big(\lambda^i(a^i, s^i, y^i)\big)$ in $J(\pi, \mu)$. This can always be done without affecting the optimality. Similarly, we use the the exponential family structure of the transition function (3.2) to get $J(\lambda, \mu)$ as:

$$\begin{aligned} J(\lambda,\mu) = &\sum_{T=0}^{\infty}\sum_{\tau_{0:T}} \gamma^T R'(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{t=1}^{T}\prod_{i=1}^{N} f(s_{t-1}^i, a_{t-1}^i, s_t^i)\Big) \\ &\exp\bigg[\sum_{i=1}^{N}\bigg(\Big(\sum_{t=0}^{T}\lambda^i(a_t^i, s_t^i, y_t^i)\Big) + \sum_{t=1}^{T}\Big(\mu^i(s_{t-1}^i, y_{t-1}^i, a_{t-1}^i)\times\phi(s_{t-1}^i, s_t^i) - \mathcal{A}(\mu^i(s_{t-1}^i, y_{t-1}^i, a_{t-1}^i))\Big)\bigg)\bigg] \end{aligned} \tag{3.8}$$

Above expression has non-negative combination of $\exp(\cdot)$ functions (as all rewards, initial beliefs, and functions f are non-negative). The exp function is convex in λ, but *not* convex in μ. Reason is that for exponential family distributions, the partition function $\mathcal{A}(\nu^i = \mu^i(\cdot))$ is a convex function of canonical parameters ν. However, the exponential function has the term $-\mathcal{A}(\nu^i)$, which is concave in ν^i. To circumvent this problem, we use another substitution as $\xi^i(s^i, y^i, a^i) = \mathcal{A}(\mu^i(s^i, y^i, a^i))$. We will also include this constraint as part of our constraint set Ω when writing the final DC formulation. Using these substitutions, our final expression for $J(\lambda, \mu, \xi)$ is:

$$\begin{aligned} J(\lambda,\mu,\xi) &= \sum_{T=0}^{\infty}\sum_{\tau_{0:T}} \gamma^T R'(\boldsymbol{s}_T)\Big(\prod_{i=1}^{N} b_0^i(s_0^i)\Big)\Big(\prod_{t=1}^{T}\prod_{i=1}^{N} f(s_{t-1}^i, a_{t-1}^i, s_t^i)\Big) \\ &\exp\Bigg[\sum_{i=1}^{N}\Bigg(\Big(\sum_{t=0}^{T}\lambda^i(a_t^i, s_t^i, y_t^i)\Big) + \sum_{t=1}^{T}\Big(\mu^i(s_{t-1}^i, y_{t-1}^i, a_{t-1}^i)\times\phi(s_{t-1}^i, s_t^i) - \xi^i(s_{t-1}^i, y_{t-1}^i, a_{t-1}^i)\Big)\Bigg)\Bigg] \end{aligned} \tag{3.9}$$

Now each exp term in the above expression is convex in each of λ, μ, ξ, and non-negative combination of convex functions is also convex. We can now reformulation policy optimization in the DC programming framework as:

$$\min_{\lambda,\mu,\xi} 0 - J(\lambda, \mu, \xi) \tag{3.10}$$

$$\sum_{a^i} e^{\lambda^i(a^i, s^i, y^i)} = 1 \;\; \forall s^i, y^i, \forall i = 1 : N \tag{3.11}$$

$$\xi^i(s^i, y^i, a^i) = A(\mu^i(s^i, y^i, a^i)) \;\; \forall s^i, y^i, a^i, \forall i = 1 : N \tag{3.12}$$

The objective (3.10) is a difference of two convex function, and is therefore a DC function. Notice that the constraints (3.11) and (3.12) are non-convex. However, the DC iteration is still applicable because as highlighted earlier, the DC objective uses linearization of the function v (which is $J(\lambda, \mu, \xi)$ in our case), and linearization based bounds are valid over the entire domain of the function.

DC Iteration for ZBPF The optimization Problem (3.10) is no easier to solve than the original problem (3.7). However, as (3.10) is a DC program, we can solve it iteratively analogous to (3.4). Let the current estimate of parameters be $(\lambda_k, \mu_k, \xi_k)$. To find the improved estimates (λ, μ, ξ), we need to solve the below problem:

$$\begin{aligned} \max_{\lambda,\mu,\xi} &\sum_{i,a^i,s^i,y^i} \nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k)\lambda^i(a^i, s^i, y^i) \\ + &\sum_{i,a^i,s^i,y^i} \nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k)\mu^i(a^i, s^i, y^i) \\ + &\sum_{i,a^i,s^i,y^i} \nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k)\xi^i(a^i, s^i, y^i) \end{aligned} \tag{3.13}$$

subject to constraints (3.11) and (3.12) on variables λ, μ, ξ

We can further simplify the above optimization problem by eliminating constraints as follows. We can replace back $\lambda^i(a^i, s^i, y^i) = \ln \pi^i(a^i|s^i, y^i)$, and substitute back $\xi^i(a^i, s^i, y^i) = \mathcal{A}(\mu^i(a^i, s^i, y^i))$ to get the following optimization problem:

$$\begin{aligned}
\max_{\pi,\mu} & \sum_{i,a^i,s^i,y^i} \nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) \log \pi^i(a^i|s^i, y^i) \\
& + \sum_{i,a^i,s^i,y^i} \nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) \mu^i(a^i, s^i, y^i) \\
& + \sum_{i,a^i,s^i,y^i} \nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) \mathcal{A}(\mu^i(a^i, s^i, y^i)) \qquad (3.14) \\
& \sum_{a^i} \pi^i(a^i|s^i, y^i) = 1, \pi^i(a^i|s^i, y^i) \geq 0;\ \forall s^i, y^i, \forall i \qquad (3.15)
\end{aligned}$$

Problem (3.14) is much easier to solve than original Problem (3.7). This is because the most computationally intensive step is to compute different gradients ∇J, which are with respect to old parameters $(\lambda_k, \mu_k, \xi_k)$, and we show that a closed form expression can be derived for them. An agent i's state, observation and action are denoted using (s^i, y^i, a^i), and (s^{-i}, y^{-i}, a^{-i}) denote the same for all other agents except i. Let $d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i})$ denote the occupancy measure or the total discounted amount of time the system was in the joint setting $(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i})$ (note that action $\boldsymbol{\nu}$ is deterministic given the joint-state, therefore not included in d^k). Expression for measure $d^k(\cdot)$ is as:

$$\sum_{t=0}^{\infty} \gamma^t P(s^i_t = s^i, s^{-i}_t = s^{-i}, y^i_t = y^i, y^{-i}_t = y^{-i}, a^i_t = a^i, a^{-i}_t = a^{-i})$$

We also denote the total discounted reward from a given joint state, action $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$ as $Q^k(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\nu})$. Both d^k and Q^k are computed as per the old policy π_k and μ_k. Using d^k, Q^k, different gradients are as:

$$\nabla_{\lambda^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) = \sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}) Q^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}, \nu^i, \nu^{-i})$$

$$\nabla_{\mu^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) = \gamma \sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}) \times \sum_{\boldsymbol{s}',\boldsymbol{y}',\boldsymbol{a}'} \phi(s^i, s'^i) P(\boldsymbol{s}', \boldsymbol{y}', \boldsymbol{a}'|s^i, s^{-i}, a^i, a^{-i}, \nu^i, \nu^{-i}) Q^k(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\nu}')$$

$$\nabla_{\xi^i(a^i,s^i,y^i)} J(\lambda_k, \mu_k, \xi_k) = -\sum_{s^{-i},y^{-i},a^{-i}} d^k(s^i, y^i, a^i, s^{-i}, y^{-i}, a^{-i}) \times \Big(Q^k(s^i, s^{-i}, a^i, a^{-i}, \nu^i, \nu^{-i}) - r(s^i, s^{-i})\Big)$$

We can substitute back the gradients to our DC iteration (3.14), and get the following optimization problem:

$$\max_{\pi,\mu} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}}\Big[Q^k(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})\Big(\sum_{i=1}^{N}\log \pi^i(a^i|s^i,y^i)\Big)\Big]$$

$$+\gamma\mathbb{E}_{\substack{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu},\\ \boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}'}}\Bigg[Q^k(\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}')\Bigg(\Big(\sum_{i=1}^{N}\phi(s^i,s'^i)\mu^i(a^i,s^i,y^i)\Big)-\Big(\sum_{i=1}^{N}\mathcal{A}(\mu^i(a^i,s^i,y^i))\Big)\Bigg]\quad(3.16)$$

$$\sum_{a^i}\pi^i(a^i|s^i,y^i)=1;\pi^i(a^i|s^i,y^i)\geq 0\ \ \forall s^i,y^i,\forall i\quad(3.17)$$

In the above problem, we have introduced only policy normalization constraints. There may be additional constraints on μ variables for the particular exponential family distribution used. The expectation over variables $(\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}')$ is as per the joint transition probability of states and actions $P(\cdot|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})$. The expectation over variables $(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})$ is as per the corresponding occupancy measure d^k.

Planning The optimization (3.16) is solvable when the policy of each agent has a convenient form (such as a tabular policy), and transition/observation function are known. Using model parameters and the old policy π_k,μ_k we can get a better policy π_{k+1},μ_{k+1}. Note that, it is not required to solve Problem (3.16) exactly. E.g., we can use general purpose nonlinear program solver such as SNOPT [47] to approximately optimize it. Such a process is also guaranteed to monotonically increase the original objective J until convergence to a local optima. In situations, when the exact planning model is not available, or state-space is too large, we next present a gradient based approach to approximately optimize (3.16).

Learning The form of Problem (3.16) is also highly convenient for reinforcement learning. We assume that policy π_θ and μ_θ are parameterized using θ. We can compute the gradient of the DC iteration objective J' (3.16) as:

$$\nabla_\theta J' = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}}\Big[Q^k(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})\Big(\sum_{i=1}^{N}\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big)\Big]$$
$$+\gamma\mathbb{E}_{\substack{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu},\\ \boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}'}}\Bigg[Q^k(\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}')\Bigg(\Big(\sum_{i=1}^{N}\phi(s^i,s'^i)\nabla_\theta\mu^i_\theta(a^i,s^i,y^i)\Big)-\Big(\sum_{i=1}^{N}\nabla_\theta\mathcal{A}(\mu^i_\theta(a^i,s^i,y^i))\Big)\Bigg)\Bigg] \tag{3.18}$$

We can use properties of exponential families to further simplify gradient $\nabla_\theta\mathcal{A}(\nu^i = \mu^i_\theta(a^i,s^i,y^i))$ as $\nabla_\nu\mathcal{A}(\nu^i)\nabla_\theta\nu^i$. The derivative $\nabla_\nu\mathcal{A}(\nu^i)$ is nothing but the expectation of the sufficient statistic ϕ in the transition function expression (3.2)[2], which can also be estimated using sampling if a closed form expression is not known. The Equation (3.18) provides a basis for computing gradients using sampling to approximately optimize (3.16). Once a new estimate θ^{k+1} is known, we can iteratively improve it again until convergence.

Multi-agent credit assignment Performing vanilla gradient ascent based on (3.18) generally suffers from poor convergence, primarily due to the high variance in the policy gradient estimator caused by the involvement of multiple agents. In multi-agent settings, the randomness in the policy gradient arises not only from an individual agent's interactions with the environment but also from the explorations of other agents. Consequently, it becomes challenging to determine the contribution of an agent's action to the global reward since the decisions made by multiple agents collectively influence the team reward. This challenge is commonly referred to as the problem of multi-agent credit assignment [22, 41]. We next show how to extract a clearer signal from empirical returns that performs effective credit assignment. We assume that the joint Q-function (for a fixed policy) is approximated as follows:

$$Q(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}) \approx \sum_{i=1}^{N} h^i_w(s^i,a^i,\nu^i,y^i(\boldsymbol{s})) \tag{3.19}$$

where h^i_w is an individual value function approximator parameterized by w for agent i.

[2]https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/other-readings/chapter8.pdf

We focus on an agent i's contribution to the gradient expression (3.18). Considering first terms depending on $\nabla_\theta \pi^i$, using Q function approximation (3.19), we get:

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}}\Big[Q^k(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big] \approx \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}}\Big[\{\sum_{m=1}^{N} h^m_w(s^m,a^m,\nu^m,y^m)\}\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big] \tag{3.20}$$

We can show that:

$$\mathbb{E}_{\boldsymbol{s}}\Big[\mathbb{E}_{\boldsymbol{a},\boldsymbol{\nu}|\boldsymbol{s}}\Big[\{\sum_{m\neq i} h^m_w(s^m,a^m,\nu^m,y^m)\}\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big]\Big] = 0 \tag{3.21}$$

Therefore, (3.20) simplifies to:

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu}}\Big[Q^k(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\nu})\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big] \approx \mathbb{E}_{\boldsymbol{s}}\Big[\mathbb{E}_{a^i,\nu^i|\boldsymbol{s}}\Big[h^i_w(s^i,a^i,\nu^i,y^i)\nabla_\theta \log \pi^i_\theta(a^i|s^i,y^i)\Big]\Big] \tag{3.22}$$

The above expression has much lower variance than estimating gradients using (3.20). Let us now consider terms depending on deterministic policy gradient $\nabla_\theta \mu^i$:

$$\begin{aligned}&\mathbb{E}\Big[Q^k(\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{\nu}')\Big(\phi(s^i,s'^i)\nabla_\theta\mu^i_\theta(a^i,s^i,y^i) - \nabla_\theta\mathcal{A}(\mu^i_\theta(a^i,s^i,y^i))\Big)\Big]\\ &\approx \mathbb{E}\Big[\{\sum_{m=1}^{N} h^m_w(s^m,a^m,\nu^m,y^m)\}\Big(\phi(s^i,s'^i)\nabla_\theta\mu^i_\theta(a^i,s^i,y^i) - \nabla_\theta\mathcal{A}(\mu^i_\theta(a^i,s^i,y^i))\Big)\Big]\end{aligned} \tag{3.23}$$

Empirically, the above expression also resulted in poor quality credit assignment because the gradient term $\nabla_\theta \mu^i_\theta$ is affected by the value accumulated by all the agents m. We therefore used an approximation where we only add the contributions from those agents which are in the immediate neighborhood of agent i given the joint-state $\boldsymbol{s}$, denoted using $\text{Nb}(i,\boldsymbol{s})$. The expression becomes:

$$\mathbb{E}\Big[\{\sum_{m\in \text{Nb}(i,s)} h^m_w(s^m,a^m,\nu^m,y^m)\}\Big(\phi(s^i,s'^i)\nabla_\theta\mu^i_\theta(a^i,s^i,y^i) - \nabla_\theta\mathcal{A}(\mu^i_\theta(a^i,s^i,y^i))\Big)\Big] \tag{3.24}$$

Using the above credit assignment techniques, our approach worked significantly better than the vanilla gradient based learning. To estimate $h^i_w(s^i,a^i,\nu^i,y^i)$, we used empirical returns. That is, for a given joint-trajectory of all the agents from the simulator, the total discounted reward obtained by an agent i given its current state-action-observation being (s^i, a^i, ν^i, y^i) is treated as the target for $h^i_w(s^i, a^i, \nu^i, y^i)$.

Practical considerations: As number of agents can become quite large in practical ZBPF scenarios, optimizing a unique policy for each agent would require far too many samples to learn within a reasonable time. Therefore, we use policy sharing among agents. We have one set of parameters for each zone or $\theta = \{\theta_z \; \forall z \in Z\}$. Our approach resembles a traffic management system where traffic through a zone z is controlled using $\pi_{\theta_z}, \mu_{\theta_z}$. To differentiate among agents, we also provide agent id as part of the observation to the agent. This approach is suited for settings where number of start and goal zones are relatively fewer than the number of agents. This is a reasonable assumption, for example, in drone delivery, where most drones takeoff and land at few specialized high-rise and safe locations in the city. If each agent has a unique start-goal zone, then the problem becomes highly combinatorial, and policy sharing among agents may not be effective.

3.4 Experiments

We evaluate our approach on several 2D grid maps and 3D maps. Our ZBPF simulator is constructed in the cross platform Unity3D engine which provides ml-agents framework for training intelligent agents, and a simple Python API to access simulation state. We vary several parameters including grid size, number of agents, start and goal zones. We compare our learning approach with credit assignment (‘DCRL’) against the vanilla gradient based approach (‘VPG’) in (3.18) where Q values are estimated by total discounted global modified rewards, and multi-agent Q-learning based approach for discrete-continuous hybrid action spaces (‘MAPQN’) [43]. We have adapted the ‘MAPQN’ approach for the ZBPF setting. We optimize the sum of arrival time of all agents (SOAT) while minimizing the constraint violations (i.e., reducing congestion). For comparison, we also show SOAT when all the agents follow their shortest path (‘SP’). This value is the lowest SOAT any approach can achieve; but it also results in high congestion. All our experiments are run on a Linux machine with 3GHz CPU and 64 GB RAM. Maximum runtime allowed was 10 hours for any approach.

3.4.1 Simulator

Figure 3.2 shows different 2D and 3D Unity maps generated by our simulator. In Unity, we can build complex environments without additional overheads such as developing physics libraries. In our simulator, we use spheres to represent intersection zones where agents are required to decide where to go and use cylinders to represent 1-way normal zones. Agents are represented by orange cubes. If agents have reached their destinations, they will be highlighted in color. In Figure 3.2(a), highlighted spheres represent landmarks which agents are required to visit (landmark constraints are discussed in Chapter 5). In Figure 3.2(c), zones are highlighted in color to indicate congestion.

3.4.2 2D open grids

Figure 3.3 shows the SOAT comparisons among DCRL, VPG and MAPQN. For each grid, starting and goal locations were the top and bottom rows. For each agent, we randomly selected its start and goal zones from the top and bottom rows. As a result, multiple agents had the same start and/or goal zones. This setting is also challenging for our approach—if multiple agents are in the same start zone, the zone based policy has to properly assign different route/travel time to them. If agents follow each other on the shortest path to the goal zone, it creates high congestion. The capacity of each zone was sampled uniformly from [1, 2] for 4x4 grid, [1, 3] for 8x8 grid, and [1, 4] for 10x10 grid. The $t_{\min}, t_{\max}$ for each zone were $1, 5$ respectively. The travel time distribution used

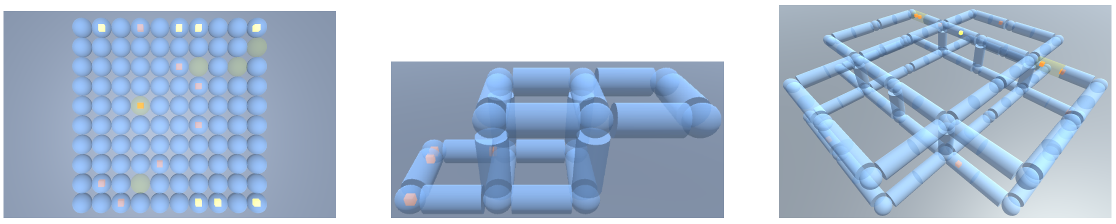

(a) 10x10 grid **(b)** SmallCube **(c)** TwoFloor

Figure 3.2: Different Unity maps generated by our simulator

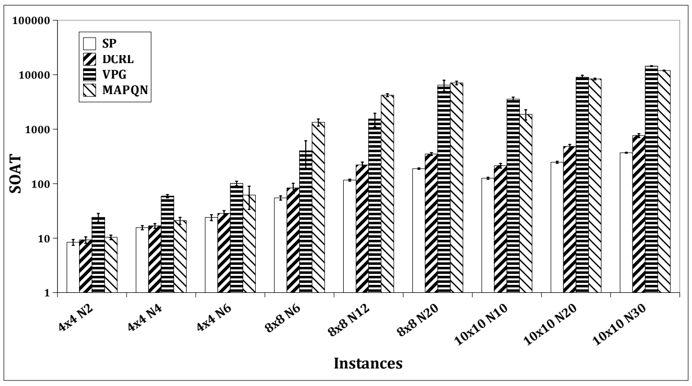


Figure 3.3: SOAT comparisons among DCRL, VPG and MAPQN (log-scale, lower is better). N# denotes number of agents.

was the binomial distribution, which is the maximum entropy distribution with a given support and mean. Each training episode was cutoff after 500 time steps or after 10 hours. We varied both grid sizes and number of agents ranging from 4x4 grid, 2 agents to 10x10 grid 30 agents. For each setting, we generated 5 instances, and the average SOAT (along with standard deviation) is shown in Figure 3.3. This figure clearly shows that as the problem becomes more complex (increasing grid size, number of agents), our approach DCRL provides significantly better solution quality than VPG and MAPQN. This is because DCRL performs credit assignment much more effectively than the VPG, and MAPQN attempts to learn Q-values over joint state and action, which may be inaccurate for large number of agents. For larger grids (e.g., 10x10) and larger density of agents (e.g., 10x10, 30 agents), several agents did not reach their goal zone when using VPG based policy and Q-learning based policy. To provide an estimate of best possible SOAT, we also show the SOAT when all agents move as fast as possible (using $t_{\min}$ time) on their shortest path to goal (denoted using 'SP'). DCRL is worse than SP (as expected), however, not by a large margin. This is expected as DCRL make agents take a longer route or take more time traveling between zones to minimize congestion. For the ZBPF problems we have generated as above, standard MAPF solvers including based on RL (such as by [86]) are not applicable in a straightforward fashion. In our instances, multiple agents can have the same start zone, which is not allowed in standard MAPF, and they also do not incorporate uncertainty in travel time and higher capacity nodes.

Setting	Congestion			Stranded agents		
	DCRL	VPG	MAPQN	DCRL	VPG	MAPQN
4x4 N2	0	0	0	0	0	0
4x4 N4	0.11	0.02	0.17	0	0	0
4x4 N6	0.12	0.04	0.29	0	0	0
8x8 N6	0.06	0.18	0.11	0	0.40	2.08
8x8 N12	0.50	0.67	31.62	0	2.21	6.20
8x8 N20	1.74	4.15	114.70	0	11.45	10.71
10x10 N10	0.44	0.07	0.68	0	6.81	2.83
10x10 N20	0.97	3.21	130.29	0	17.25	14.18
10x10 N30	2.12	35.12	273.16	0	27.33	18.38

Table 3.1: Average congestion level and stranded agents comparisons

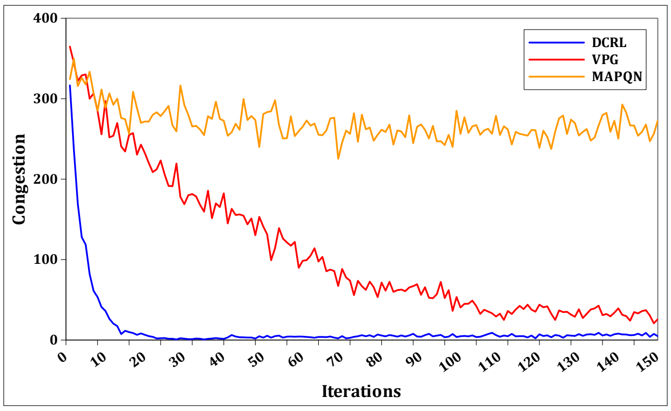


Figure 3.4: Learning curve (showing congestion level) of DCRL, VPG and MAPQN on a 10x10 grid, 30 agents instance

Table 3.1 shows the results comparing congestion level between our approach, VPG and MAPQN . For each time step, if number of agents in a zone z (say N_z) are more than its capacity C_z, than $N_z - C_z$ is added to the congestion level. We also show, on an average, how many agents were unable to reach destination in these three approaches (or ‘stranded agents’). These results clearly show that our approach is able to effectively minimize congestion over VPG and MAPQN , and for all tested instances, agents were able to reach their destinations. The standard deviation in congestion metric was low for DCRL (less than 1 for all problems).

Figure 3.4 shows the learning curve for DCRL, VPG and MAPQN for the largest grid

Metric	**Instance**	**Average**		
		DCRL	VPG	MAPQN
SOAT	SmallCube	155.29	312.00	775.98
	TwoFloor	532.35	1300.73	6346.09
Stranded agents	SmallCube	0	0	0.33
	TwoFloor	0	0.2	8.66
Congestion	SmallCube	3.66	4.37	10.61
	TwoFloor	5.09	7.24	112.49

Table 3.2: Average SOAT, stranded agents, and congestion level comparisons (over 5 instances).

with most number of agents (10x10, 30 agents). This figure clearly shows that with credit assignment, our approach was quite stable during learning and converged much faster and to a better quality than VPG. For this large setting, MAPQN was unable to reduce the congestion level to an acceptable low level.

3.4.3 3D maps

We also evaluated our approach on two generated 3D maps–'SmallCube' and 'TwoFloor' as shown in Figure 3.2 . SmallCube map and TwoFloor map have 30 zones and 142 zones respectively. Total number of agents were 10 and 20 for these two maps respectively. Capacity of each zone was uniformly sampled from [1, 3] and [1,4]. Agents moved from bottom left 5 zones (which were start zones) to top right 5 zones (which were goal zones) for SmallCube, and moved from bottom front zones to bottom back zones for TwoFloor map. Rest of the settings were same as for grid graphs. For such maps also, our approach DCRL worked much better than VPG and MAPQN as expected. Table 3.2 shows the SOAT, stranded agents and congestion level results. For SmallCube map, all agents were able to reach their goals in both DCRL and VPG approaches. But compared with DCRL, VPG gives higher SOAT (lower SOAT is better) and marginally greater congestion level. MAPQN performed slightly worse in terms of stranded agents (which is 0.33), but also gives worst SOAT and congestion level. For TwoFloor map, VPG gives reasonably higher SOAT of 1300.73 versus 532.35 by DCRL. This is because there are slightly more average stranded agents (0.2) versus zero stranded agents by DCRL. Compared with DCRL and VPG, MAPQN gives the worst SOAT (6346.09), and it cannot work well for either making all agents reach goal zones or minimizing the congestion.

CHAPTER 4

Decision Diagrams for Constraint Compilation

In both single-agent and multi-agent constrained RL, one of the main difficulties is identifying all the valid actions that comply with the constraints specified in a CMDP or constrained Dec-POMDP. In fact, since the constraint are known to us, they can be treated as domain knowledge. In this chapter, we present two general decision diagrams that have been recently proposed for compiling domain knowledge. First, in Section 4.1, we introduce the boolean representation of constraints in RL by providing two practical examples. Then, in Section 4.2, we introduce sentential decision diagrams (sdds) that provide a compact representation of a Boolean formula. Finally, in Section 4.3, we present probabilistic sentential decision diagrams (psdds), which are parameterized sdds that can represent a probability distribution over all valid instantiations encoded by the underlying sdd. Compiling constraints using decision diagrams is the symbolic side, which is integrated with neural side in the next two chapters to provide efficient neuro-symbolic methods to solve constrained RL problems.

4.1 Boolean representation for constraint in RL

Let $\boldsymbol{X}_S = \{\beta_1, \ldots, \beta_n\}$ denote the set of n propositional variables defining a state; each $\beta_i \in \{0, 1\}$. Let $\boldsymbol{X}_A = \{\alpha_1, \ldots, \alpha_m\}$ denote the set of m propositional variables defining an action; each $\alpha_i \in \{0, 1\}$. Any state is denoted by an assignment of truth values, s_β, to variables in $\boldsymbol{X}_S$; any action is similarly defined using a_α. Such propositional logic based model representation has also been used for planning in discrete factored state and action spaces [87].

Assume there are K constraints. Each constraint k is defined as a Boolean function f_k over variables sets $\boldsymbol{X}_S$ and $\boldsymbol{X}_A$. Let $f_k|s_\beta$ represent the reduced Boolean subfunction f over variables $\boldsymbol{X}_A$ where variables in $\boldsymbol{X}_S$ are set to their respective instantiation according to the state s_β. Given a state s_β, the set of actions that satisfy all constraints (valid actions), $\mathcal{C}(s_\beta)$, is defined as:

$$\mathcal{C}(s_\beta) = \Big\{ a_\alpha \;\Big|\; \bigwedge_{k=1}^{K} f_k|s_\beta(a_\alpha) = 1 \Big\} \tag{4.1}$$

We next give some examples showing how to represent constraints using Boolean functions in a variety of practical problems such as resource allocation and path planning.

Example 4.1.1 (*Resource allocation constraints*)**.**

Consider a resource allocation setting where we need to allocate P resources to Q entities, assuming a 0/1 allocation setting. For simplicity, we assume that there are more entities than resources and each entity can only have at most one resource(this is not a limitation; in Chapter 6, we can allocate any number of resources to an entity in our tested domain).

We encode the set of all valid allocations using *pseudo-Boolean* (PB) constraints (linear constraints over Boolean variables) [36]. A PB-constraint is an inequality $C_0 p_0 + C_1 p_1 + \ldots + C_{n-1} p_{n-1} \geq C_n$, where each p_i is a literal and C_i is an integer coefficient. A true literal is denoted as 1 and false as 0. Note that it is easy to incorporate $\leq$ and $=$ both as PB-constraints [36].

We create a Boolean variable X_{pq} denoting whether a resource p is assigned to entity q. The set of PB constraints encoding a valid assignment is denoted as:

$$\sum_{q=1}^{Q} \sum_{p=1}^{P} X_{pq} = P; \; \sum_{q=1}^{Q} X_{pq} = 1 \; \forall p = 1 : P \; \sum_{p=1}^{P} X_{pq} \leq 1 \; \forall q = 1 : Q \tag{4.2}$$

It is shown in [36] how PB-constraints can be translated into Boolean formulas which evaluate to true only if any given literal assignment satisfies the PB-Constraints. Furthermore, it is possible to represent the resulting Boolean functions using popular compact data structures such as binary decision diagrams (BDDs) [17]. We can also have a single

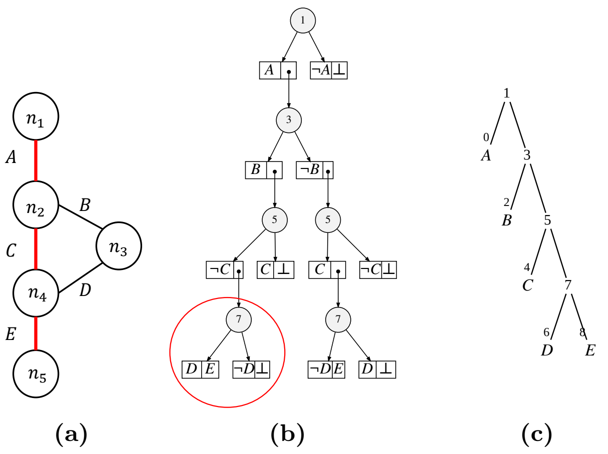


Figure 4.1: (a) An undirected graph; A, B, C, D, E represent the edge variables. A simple path from $s = n_1$ to $d = n_5$ is highlighted in red and can be written as a propositional sentence $A \land C \land E \land \neg B \land \neg D$; (b) An sdd encoded all simple paths in the graph where the encircled node represents a decision node $(D, E), (\neg D, \bot)$ (c) a vtree for the sdd

BDD representing the conjunction of BDDs for their respective PB-constraints, which encodes the set of all the valid resource allocations.

Example 4.1.2 (*Multistep path planning*).

Consider a multistep path planning problem in a grid introduced in [33]. In their environment, the agent receives a fixed-size square window surrounding its current position as the observation. Consider the action set to be the set of all possible simple paths (no loops) in the agent's observation, or all possible simple paths of a fixed length. As shown in [26], a path p in a graph $G = (V, E)$ can be represented as a Boolean formula as follows. We create a Boolean variable X_{ij} $\forall (i, j) \in E$. If an edge (i, j) occurs in path p, we set $X_{ij} = 1$, otherwise 0. Therefore, the conjunctions of all such literals denotes the formula for path p, and Boolean formula for all the paths in a graph is obtained by the disjunction of formulas for all the valid paths in G [26]. To account for blocked cell in the agent's observation window, we set all $X_{ij} = 0$ for any edge (i, j) that is incident to the blocked cell. This is analogous to reducing the Boolean formula for all the paths by using the instantiation of variables that are set according to the agent's current observation.

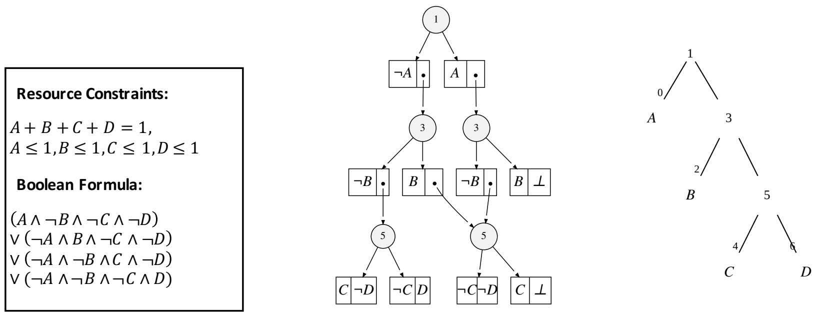
4.2 Sentential decision diagrams

In resource allocation problems, the number of valid assignments can be combinatorial, as there are m indistinguishable resources that must be assigned to n entities, resulting in $|A| = \binom{m+n-1}{n-1}$. Similarly, the set of all simple paths in a graph is also combinatorial. Due to this, a compact representation of the Boolean formula representing resource allocations or paths is necessary. To accomplish this, we use a general data structure called a *sentential decision diagram* (sdd) [30]. An sdd is a succinct and tractable representation of a Boolean formula that generalizes the well-known ordered binary decision diagrams (OBDDs) [17]. Succinctness refers to the size of the compiled knowledge representation, and tractability implies that operations such as conjunction (or conjoin) are polytime operations in the size of sdds [94]. It has also been shown that sdds can be exponentially more succinct than OBDDs [15].

An sdd represents a Boolean function $f(\boldsymbol{A}, \boldsymbol{B})$ on some non-overlapping variable sets $\boldsymbol{A}$ and $\boldsymbol{B}$ [30]. The function f is represented as $f = (p_1(\boldsymbol{A}) \wedge s_1(\boldsymbol{B})) \vee \ldots \vee (p_k(\mathbf{A}) \wedge s_k(\mathbf{B}))$, with each *element* (p_i, s_i), $i = 1 \ldots k$ of the decomposition composed of a *prime* p_i and a *sub* s_i, which themselves are sdds. An sdd represented as a decision diagram describes members of a combinatorial space (e.g., resource allocations, paths in a graph) using propositional logic in a tractable manner. Each model of an sdd is a valid assignment of Boolean variables. It has two kinds of nodes:

- *terminal node*, which can be a literal (X or $\neg X$), always true ($\top$) or always false ($\bot$).
- *decision node* having k branches. Decision node represents $(p_1 \wedge s_1) \vee \ldots \vee (p_k \wedge s_k)$ where all (p_i, s_i) are recursively sdds. The primes for a decision node are always consistent, mutually exclusive and exhaustive [30].

An sdd is characterized by a *full* binary tree, called a *vtree*, which induces a total order on the variables $\boldsymbol{X} = \boldsymbol{A} \cup \boldsymbol{B}$ from a left-right traversal of the vtree. Given a fixed vtree, the sdd is unique. An sdd node n is *normalized* (or associated with) for a vtree node v as follows:



Resource Constraints:

$A + B + C + D = 1,$
$A \leq 1, B \leq 1, C \leq 1, D \leq 1$

Boolean Formula:

$(A \wedge \neg B \wedge \neg C \wedge \neg D)$
$\vee (\neg A \wedge B \wedge \neg C \wedge \neg D)$
$\vee (\neg A \wedge \neg B \wedge C \wedge \neg D)$
$\vee (\neg A \wedge \neg B \wedge \neg C \wedge D)$

(a) **(b)** **(c)**

Figure 4.2: (a) Resource allocation constraints and Boolean formula; (b) An sdd encoded all valid resource allocations; (c) a vtree for the sdd.

- If n is a terminal node, then v is a leaf vtree node containing the variable of n (if any).
- If n is a decision node, then n's primes (subs) are normalized for the left (right) child of v.
- If n is the root node, then v is the root vtree node.

Intuitively, a decision node n being normalized for vtree node v implies that the Boolean formula encoded by n contains only those variables contained in the sub-tree rooted at v. The Boolean formula encoding the domain knowledge can be compiled into a decision diagram using the sdd compiler [80]. The resulting sdd may not be exponential in size even though it represents an exponential number of objects.

Figure 4.1(b) shows an sdd representing for the Boolean formula that encodes all simple path from n_1 to n_5 in a graph as shown in Figure 4.1(a). Figure 4.1(c) shows the vtree for the sdd where the variable order is (A, B, C, D, E). The encircled node is a decision node with two elements (D, E) and $(\neg D, \bot)$. The primes are D and $\neg D$ and subs are E and $\bot$. The Boolean formula representing this sdd node is $(D \wedge E) \vee (\neg D \wedge \bot)$. The Boolean formula encoded by the whole sdd is given by the sdd root node.

Figure 4.2 shows how the constraints in resource allocation are represented using an sdd. We assume there is one resource and four entities, and we create four Boolean

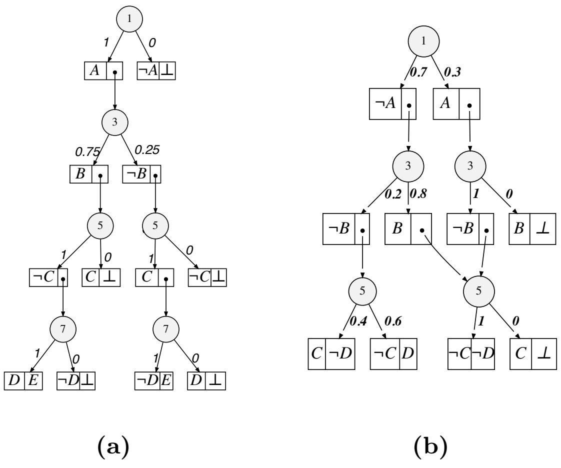


Figure 4.3: (a) A psdd with underlying sdd shown in Figure 4.1(b); (b) A psdd with underlying sdd shown in Figure 4.2(b).

variables $X_{11}, X_{12}, X_{13}, X_{14}$ denoting whether the resource should be assigned to entity $q, q = 1, \ldots, 4$ or not. To have better viewing of the sdd graph, we rename these four Boolean variables as A, B, C, D respectively. The resource constraints and corresponding Boolean formula are shown in Figure 4.2(a). Given the Boolean formula, Figure 4.2(b) shows the constructed sdd, Figure 4.2(c) is the vtree for the sdd. In this example, one model for the sdd is $(\neg A, \neg B, C, \neg D)$, which means the resource is assigned to the third entity. A model can be obtained by traversing the sdd from top to bottom. For each decision node, we select a branch whose sub is not false to visit. Traversal terminates after we have the instantiation for all the variables.

4.3 Probabilistic sentential decision diagrams

An sdd encodes all instantiations that satisfy the constrains represented in Boolean formula. If we parameterize each decision node of the sdd, such that the local parameters form a distribution, the resulting probabilistic structure is called a psdd or a *probabilistic* sdd [61]. psdds are probabilistic extensions of sdds. A psdd induces a probability distribution over the models of the underlying sdd. That is, a psdd assigns a strictly positive probability to an instantiation that satisfies the Boolean formula for the

Instantiation				Probability
A	$\neg B$	$\neg C$	$\neg D$	0.3
$\neg A$	B	$\neg C$	$\neg D$	0.56
$\neg A$	$\neg B$	C	$\neg D$	0.056
$\neg A$	$\neg B$	$\neg C$	D	0.084

Table 4.1: Probability distribution encoded by the psdd of Figure 4.1(b)

underlying sdd; and zero probability to instantiations that do not satisfy the underlying sdd. The structure of a psdd normalized for an sdd is described as follows.

- For each decision node $(p_1, s_1), \ldots, (p_k, s_k)$, there are non negative parameters θ_i such that $\sum_{i=1}^{k} \theta_i = 1$ and $\theta_i = 0$ iff $s_i = \perp$.
- For each terminal node $\top$, there is a parameter $0 < \theta < 1$.

Parameters $\theta_i, i = 1, \ldots, k$ are also called the *local distribution* associated with a decision node. psdds are tractable structures of probability distributions as several probabilistic queries can be performed in poly-time such as computing marginal probabilities, conditional probabilities, and sampling from the distribution $\Pr(\boldsymbol{X})$ represented by the psdd.

Figure 4.3(a) and (b) show psdds with underlying sdds shown in Figure 4.1(b) and Figure 4.2(b) respectively. Table 4.1 shows the probability distribution $\Pr(\boldsymbol{X})$ represented by the psdd shown in Figure 4.1(b). The distribution is defined over four Boolean variables, resulting in a total of $2^4 = 16$ variable instantiations. However, only four instantiations have non-zero probabilities due to the constraints imposed on the space. These four instantiations are models of the underlying sdd. In other words, $\Pr(\boldsymbol{x}) = 0$ for all other instantiations. The probability over all valid instantiations is defined inductively, as described in the work by Kisa [61]. The distribution induced by a terminal node is first defined, followed by the distribution of a decision node, which is defined in terms of the distribution induced by its primes and subs. To be more specific, let $\Pr_n$ denote the distribution represented by node n over the variables of vtree v. This distribution is determined according to the following rules.

- If n is a terminal node, and v contains variables X, we have

n	$\Pr_n(X)$	$\Pr_n(\neg X)$
$X : \theta$	θ	$1-\theta$
$\perp$	0	0
X	1	0
$\neg X$	0	1

- If n is a decision node $(p_1, s_1, \theta_1), \ldots, (p_n, s_n, \theta_n)$, and v contains left variables $\boldsymbol{X}$ and right variables $\boldsymbol{Y}$, then we have,

$$\Pr_n(\boldsymbol{xy}) \overset{def}{=} \Pr_{p_i}(\boldsymbol{x}) \cdot \Pr_{s_i}(\boldsymbol{y}) \cdot \theta_i \qquad for\ i\ where\ \boldsymbol{x} \models p_i$$

Applying this definition to the psdd of Figure 4.3(b) leads to the distribution in Table 4.1 for its root node.

To summarize, sdd and psdd are compact knowledge compilation frameworks for Boolean constraints. sdd represents all valid models/actions, while psdd encodes a probability distribution over these valid actions. One key advantage of these encodings is their tractability, which refers to the succinctness in terms of the number of nodes in an sdd and the number of parameters in a psdd. This tractability allows for efficient representation and manipulation of complex constraints. In Chapter 5, We will demonstrate how to compile various constraints, such as coverage constraint and landmark constraint in ZBPF, using sdds. Additionally, we will discuss methods to improve the sample efficiency of RL algorithms employed in ZBPF by integrating sdds. In Chapter 6, we will illustrate how psdds can be employed to encode the distribution of all valid actions that satisfy complex resource allocation constraints. Furthermore, we will demonstrate how RL algorithms can be leveraged to learn the parameters of the psdd, thereby effectively tackling action-constrained RL problems.

Chapter 5

Neuro-symbolic for Zone-based Multi-agent Pathfinding

In Chapter 3, we presented a zone-based multi-agent pathfinding problem with zone capacity constraints and proposed a MARL algorithm to solve it. However, finding the best policies for agents is a challenging task. One of the significant challenges in ZBPF is the poor quality of samples collected from agents at the initial training stage. For instance, a single agent's path may not be complete, as shown in Figure 5.1(b), where an agent moves towards a deadend. Additionally, agents can move in cycles, as shown in Figure 5.1(a) when an agent loops back to one of its earlier vertices. The poor quality of samples leads to a highly sample inefficient learning approach. To address this issue, in this chapter, we propose a neuro-symbolic method to incorporate a domain constraint on all agents to ensure that the agent's path is complete and simple, with no loops. We refer to this domain constraint as the simple path constraint. Furthermore, we consider two practical domain constraints: the landmark constraint, which requires the agent to visit specific landmark nodes before reaching the destination, as shown in Figure 5.1(c), and the coverage constraint, which requires the agent to visit some nodes at least once every k time steps. We note that the domain constraints mentioned above are explicit cumulative constraints that do not take a form as the expectation of total discount costs being less than a threshold. Furthermore, we consider these constraints as hard constraints, meaning that each path generated through sampling must adhere to these constraints.

The rest of this section is structured as follows. In Section 5.1, we introduce how to

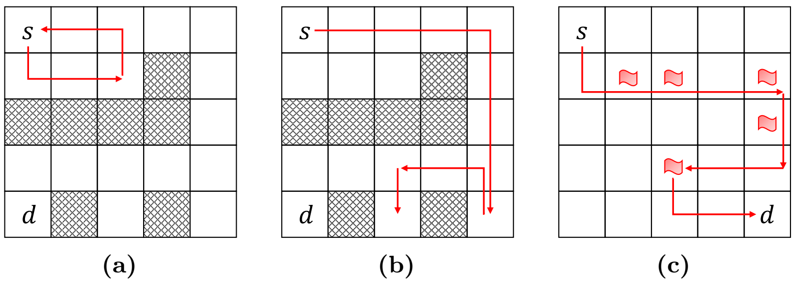


Figure 5.1: (a) Path with a loop; (b) Path to a deadend; (c) Path with landmarks; Dark nodes are blocked. Landmark nodes are flagged.

incorporate valid actions into different RL algorithms at each step when given all valid actions that satisfy domain constraints. In Section 5.2, we explain how to model different domain constraints in ZBPF using propositional logic and compile them in a symbolic manner using sentential decision diagrams (sdds), as introduced in Chapter 4. We also show how to query the set of valid actions efficiently from an sdd. In Section 5.3, we demonstrate the effectiveness of our symbolic knowledge compilation framework in ZBPF and how it improves the performance of MARL algorithms by integrating the compiled knowledge. We show the compilation framework's succinctness, efficiency, and modularity as well.

5.1 Incorporating valid actions in RL

Given an undirected graph $G = (V, E)$ in ZBPF, the set V denotes the nodes (zones) where agents can move, and edges E connect different nodes. An agent i has a start vertex s_i and final goal vertex g_i. At any time step, an agent can be located at a vertex $v \in V$, or in-transit on an edge (u, v) (i.e., moving from vertex u to v). To better capture the domain constraints in ZBPF, we simplify the action set defined in ZBPF by ignoring the continuous action component since it does not control the change of location. We assume that an agent's action set is denoted by $A = A_{\text{mov}} \cup A_{\text{oa}}$. Intuitively, A_{mov} denotes actions that intend to change the location of the agent from the current vertex to a neighboring directly connected vertex in the graph (e.g., move up, right, down, left in a grid graph). The set A_{oa} denotes other actions that do not intend to change the location of the agent (e.g., noop that intends to make the agent stay at the current vertex).

Let p_t^i denote the path taken by an agent i until time t (or the sequence of vertices visited by an agent starting from source s_i). We also assume that it does not contain any cycle. This information can be extracted from agent's history τ_t^i. Let $a \in A_{\text{mov}}$ be a movement action towards vertex a_v. To handle the hard domain constraints in ZBPF, we assume the exists a function validActions that takes an agent's current path p_t^i as input and returns a set of valid actions that satisfy all domain constraints at each RL step. Formally, we have,

$$\{a \in A_{\text{mov}} \text{ s.t. } [\mathrm{p}_t^i, a_v] \rightsquigarrow d_i\} = \text{validActions}(\mathrm{p}_t; s_i, d_i) \tag{5.1}$$

where $[\mathrm{p}_t^i, a_v] \rightsquigarrow d_i$ implies there exists at least one simple path from source s_i to destination d_i that includes the path segment $[\mathrm{p}_t^i, a_v]$, and satisfies all domain constraints. Therefore, starting with $\mathrm{p}_0 = [s_i]$, the RL approach would only sample simple paths that are guaranteed to satisfy the constraints and reach the agent's destination, thereby significantly pruning the search space, and resulting in high reward trajectories. Given the set of valid actions, we next present simple and easy-to-implement modifications to a variety of deep multiagent RL algorithms which can be used in ZBPF.

5.1.1 Policy gradient based MARL

We first provide a brief background of policy gradient approaches for single agent case [104]. An agent's policy π_θ is parameterized using θ. The policy is optimized using gradient ascent on the total expected discounted reward $J^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$. The gradient is given as:

$$\nabla_\theta J^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)] \tag{5.2}$$

The above gradient expression is also extendible to the multi-agent case in an analogous manner [83, 41]. The input to policy are some features of the agent's observation history or $\phi(\tau^i)$. The function ϕ can be either hard-coded (e.g., only last two observations), or can be learned using recurrent neural networks.

For using the function validActions, the only change we require is in the structure of an agent's policy π (we omit superscript i for brevity). The main challenge is addressing the variable sized output of the policy in a differentiable fashion. Assuming a deep neural network based policy π, given the discrete action space A, the last layer of the policy has $|A|$ outputs using the Softmax. However, when using validActions, the probability of actions not in validActions needs to be zero. Also, the set returned by validActions changes as the observation history τ of the agent is updated. However, there is an easy fix similar to proposed in [19]. We use $\tilde{\pi}$ to denote the standard way policy π is constructed with last layer having fixed $|A|$ outputs. The policy π is re-defined as:

$$\pi(a|\tau) = \begin{cases} 0 \text{ if } a \notin \text{validActions}(\text{p}(\tau); s, d) \\ \text{else } \frac{\exp\left(\tilde{\pi}(a|\phi(\tau))\right)}{\sum_{a' \in \text{validActions}(\text{p}(\tau); s, d)} \exp\left(\tilde{\pi}(a'|\phi(\tau))\right)} \end{cases} \tag{5.3}$$

where $\text{p}(\tau)$ denotes the path taken by the agent so far, and s, d are its source and destination. Sampling from π guarantees that invalid actions are not sampled. Furthermore, π is differentiable even when validActions gives different length outputs at different time steps. The above operation can be easily implemented in autodiff libraries without requiring a major change in the policy structure π.

5.1.2 Q-learning based MARL

Deep Q-learning for the single agent case [110] has also been extended to the multiagent setting [85]. In the QMIX approach [85], the joint action-value function $Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \psi)$ is factorized as (non-linear) combination of action-value functions $Q_i(\tau^i, a^i; \theta^i)$ of each agent i. A key operation when training different parameters θ^i and ψ involves maximizing $\max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \phi)$ (for details we refer to [85]). This operation is intractable in general, however, under certain conditions, it can be approximated by maximizing individual Q functions $\max_{a \in A} Q_i(\tau^i, a^i)$ in QMIX. We require two simple changes to incorporate the validActions function in QMIX. First, instead of maximizing over all the actions, we maximize only over valid actions of an agent as $\max_{a \in \text{validActions}(\text{p}(\tau^i); s^i, d^i)} Q_i(\tau^i, a)$. Second, in Q-learning, typically a replay buffer is also used which stores samples from the environment as $(\boldsymbol{\tau}, \boldsymbol{a}, \boldsymbol{\tau}', r)$. In our case, we also store additionally the set of valid actions for the next observation history τ'^i for each agent i as validActions$(\mathrm{p}(\tau'^i); s^i, d^i)$ along with the tuple $(\boldsymbol{\tau}, \boldsymbol{a}, \boldsymbol{\tau}', r)$. The reason is when this tuple is *replayed*, we have to maximize $Q^i(\tau'^i, a)$ over $a \in$ validActions$(\mathrm{p}(\tau'^i); s^i, d^i)$, and storing the set validActions$(\mathrm{p}(\tau'^i); s^i, d^i)$ would reduce computation.

5.2 Implementing validActions using decision diagrams

In this section, we will explain how to implement the validActions function by leveraging sdds as introduced in Chapter 4. To start with, we describe how to model the domain constraints in ZBPF as Boolean formulas and compile them into sdds. To compile a variety of complex constraints on routes, sdd multiplication operation provides a simple and modular way. This makes the framework flexible and general purpose for modeling real world applications. We also present how to query from a compiled sdd to get valid actions in this section.

5.2.1 Modelling and compiling domain constraints

We now present how to use Boolean formulas to model three domain constraints in ZBPF. Given an undirected graph $G = (V, E)$, we associate each edge $(i, j) \in E$ with a Boolean random variable $X_{i,j}$. Let bold upper case letter $(\boldsymbol{X})$ denote a set of Boolean variables and its lower case counterpart $(\boldsymbol{x})$ denote the instantiations.

Simple path constraint

A simple path is defined as a path that connects a source and a destination in a graph without any loops. To represent a path in a graph G, we can assign true or false values to each Boolean variable $X_{i,j}$ for each edge (i, j) in G. If an edge (i, j) occurs in the path, then $X_{i,j}$ is set to true; otherwise, it is set to false. By taking the conjunction of these *literals*, we can represent a specific path. To represent all possible paths, we disjoin the Boolean formulas for all paths [26]. Let us consider the graph shown in Figure 5.3(a). We assign Boolean variables A, B, C, D and E to represent the edges $(n_1, n_2), (n_2, n_3), (n_2, n_4), (n_3, n_4)$ and (n_4, n_5), respectively. By using the conjunction and disjunction operations, we can construct a Boolean formula that represents all simple paths connecting the source node n_1 and the destination node n_5 as follows.

$$(A \wedge C \wedge E \wedge \neg B \wedge \neg D) \vee (A \wedge \neg C \wedge E \wedge B \wedge D)$$

Since the number of simple paths between two nodes can be exponential, we use sdd as introduced in Chapter 4 as a compact representation of the Boolean formula representing paths. To construct an sdd encoding all simple paths, we use the `GRAPHILLION` [55] package to first construct a ZDD and then convert it to an sdd [78, 79]. We also note that a modified depth-first search algorithm [90] is also able to find out all simple paths. However, using sdds, we can model much richer constraints in a tractable manner, and for these constraints, there may not be a straightforward graph heuristic such as shortest path.

Hierarchical clustering for large graphs To increase the scalability of our framework, we take motivation from [24, 95]. These previous results show that by suitably partitioning the graph G among clusters in a hierarchical way, we can keep the size of the sdd tractable even for very large graphs. Such partitioning does result in the loss of expressiveness as the sdd for the partitioned graph may omit some simple paths, but empirically, we found that this approach worked well.

To partition a map represented as an undirected graph $G = (V, E)$, we divide the nodes V into clusters or regions $c_1, \ldots, c_m$. Each cluster c_i consists of both internal edges within the cluster and external edges that cross into the cluster. We create a graph G_p by considering these clusters as nodes. To enforce constraints on the variable $\boldsymbol{X}$, we utilize both G and G_p such that paths that are simple in G_p remain simple with respect to G. Specifically, the paths cannot enter a region more than once, and they also cannot visit any nodes within the clusters multiple times. We represent all the simple paths within the clusters $c_1, \ldots, c_m$ and across the clusters using sdds. This hierarchical representation captures the paths in a structured manner, employing two levels of hierarchy: one for paths across the clusters and another for paths within the clusters.

Figure 5.2(a) illustrates the partitioning of a 4x4 grid map into clusters $c_1, \ldots, c_4$, where

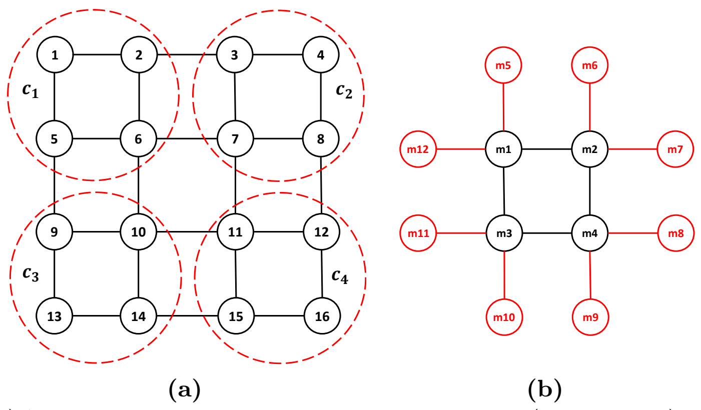


Figure 5.2: (a)A 4x4 grid map partitioned into regions (or clusters) $c_1, ..., c_4$. $c_1, ..., c_4$ form a 2x2 grid map G_p (b) Inside of a cluster, e.g., c_1. The black edges are the inernal edges and the red edges are the external edges.

the resulting graph G_p is constructed using these clusters as nodes. In Figure 5.2(b), we zoom in on the interior of a cluster c_1 where $m1$ corresponds to node 1 in the original map, $m2$ corresponds to node 2, and so on. Similarly, edge $(m2, m7)$ corresponds to edge $(2, 3)$ in the original map, $(m4, m8)$ corresponds to $(6, 7)$, and so forth. The black edges represent internal edges within the cluster, while the red edges represent external edges connecting the cluster with other clusters. For each cluster, we construct an sdd to represent all simple paths between the red nodes within the cluster. Additionally, an sdd is constructed to represent all simple paths between nodes in graph G_p.

Visiting landmarks constraint

In Figure 5.1(c), let us consider a scenario where an agent needs to navigate along a simple path from a source to a destination while also visiting certain landmarks. Assume that there are N landmark nodes denoted as $1, \ldots, N$. We define $\text{inc}(k)$ as the set of variables corresponding to the edges incident on landmark k, represented as $\text{inc}(k) = X_{i,k} \mid (i, k) \in E$. To ensure that the agent visits landmark k, we need to ensure that at least one variable in $\text{inc}(k)$ is set to true. This constraint can be expressed as $\text{visit}(k) = \bigvee_{X \in \text{inc}(k)} X$. To guarantee the visitation of all landmarks, we can take the conjunction of such constraints for each landmark, resulting in a conjunction of constraints over all landmarks.

$$\text{visit} = \wedge_{k=1}^{N} \text{visit}(k) \tag{5.4}$$

The visit Boolean formula can be represented using an sdd, denoted as sdd_1. However, sdd_1 does not incorporate any information about routes or paths. To address this, we introduce another sdd, sdd_2, representing the simple path constraints from the source to the destination. We can combine these two sdds so that the resulting sdd has models (satisfiable instantiations) from both the sdds. This can be achieved by the conjoin operation on sdds, and it also has polytime complexity in the size of component sdds [94]. This problem has applications in settings such as taxi pick-up and drop-off [32] and logistics [64].

Coverage constraint

Consider the simple graph shown in Figure 5.3(b), where the agent's task is to visit node 3 at least once every k time steps over a horizon T. To model this problem, we construct a time-indexed graph based on the given graph. In this time-indexed graph, the agent chooses one edge at each time step t. For example, if the agent selects edge (i, j) between time t and $t + 1$, it signifies a movement from node i to node j. If $i = j$, it means the agent stays at node i. The graph is designed to only allow movement from left to right, aligning with the flow of time. To represent this constraint, we focus on the edges between time steps t and $t + 1$, which are represented by the variables in the set $E(t, t + 1)$. Each of these edges can be taken *exactly once* so that the agent does not move backward in time. For each variable $X_i \in E(t, t{+}1)$, we make sure that only one of them is true using the formula $\text{only}(X_i) = X_i \wedge_{j \neq i} \neg X_j$. The time constraint between time steps t and $t{+}1$ is given by:

$$\text{tc}(t, t{+}1) = \vee_{X \in E(t,t+1)} \text{only}(X) \tag{5.5}$$

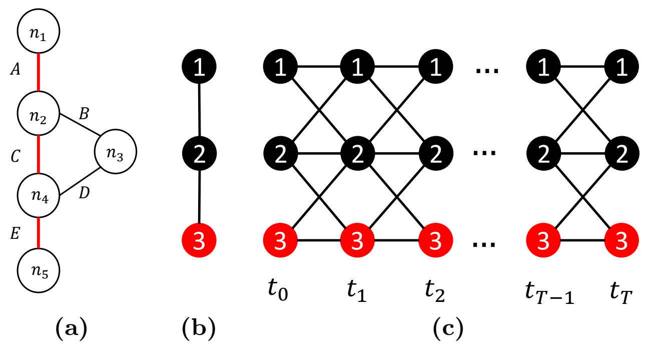


Figure 5.3: (a) An undirected graph A; (a) An undirected graph B; (b) Time-indexed graph for graph B

The time constraint over all the time steps is given as follows.

$$\wedge_{t=0}^{T-1} \mathrm{tc}(t, t+1) \tag{5.6}$$

To model constraints for visiting node 3 every k time steps, we can adopt a similar strategy to the landmarks constraint visit but apply it between consecutive time steps t and $t+k$. By taking the conjunction of all these constraints over the time horizon T, we obtain the final Boolean formula. This problem can be easily generalized to visit N nodes at least once every k time steps. In literature, this is called the coverage problem [120] and can be leveraged for applications like patrolling [49, 50]. We can compile separate sdds for all such constraints including an sdd for the simple path constraints, and conjoin them (as in the landmark constraint) to get one sdd that represents all the constraints. In this fashion, we can represent complex constraints in a simple and modular fashion.

5.2.2 Querying decision diagrams for valid actions

Assuming that the agent's current sampled path is denoted by p, in the context of sdd, we can assume that p is a set of edges in the graph G that the agent has traversed from the source node s, with $\boldsymbol{X}_\mathrm{p}$ representing the variables for the edges in p. It's worth noting that the compiled sdd encodes all paths that satisfy the domain constraints, rather than all valid actions given p, as described in Eq. 4.1 in Chapter 4. Let v_p denote the current vertex of the agent in G (assuming v_p is not the destination). We can use $\mathrm{Nb}(v_\mathrm{p})$

to represent all direct neighbors of v_p in G. Given the current p, the set of paths that satisfy all domain constraints after visiting v' is given as follows.

$$\mathcal{C}(\boldsymbol{e}^{p'}) = \left\{a \;\middle|\; f_r|\boldsymbol{e}^{p'}(a) = 1\right\} \tag{5.7}$$

where f_r denote the Boolean function represented by the sdd root node r and $\boldsymbol{e}^{p'} = \boldsymbol{x}_\mathrm{p} \cup \{X_{v_\mathrm{p},v'} = \mathit{true}\}$ and $v' \in \mathrm{Nb}(v_\mathrm{p})$. $f_r|\boldsymbol{e}^{p'}$ is a subfunction resulting from setting variables of f_r to their values in $\boldsymbol{e}^{p'}$. If this set is not empty, it means that there exists at least one path that leads to the destination while satisfying all constraints after the edge (v_p, v'). Formally, we have,

$$\mathrm{validActions(p)} = \{v' \mid v' \in \mathrm{Nb}(v_\mathrm{p}),\ (v_\mathrm{p}, v') \notin \mathrm{p},\ \mathcal{C}(\boldsymbol{e}^{p'}) \neq \emptyset\} \tag{5.8}$$

It is important to note that the number of paths in $\mathcal{C}(\boldsymbol{e}^{p'})$ could potentially be exponential. In larger graphs, it can be particularly challenging to enumerate all of these paths. However, in fact, it is unnecessary to find all possible paths. The main goal is to determine whether $f_r|\boldsymbol{e}^{p'}$ is satisfiable or not. We define a function $\mathrm{SAT}(f_n|\boldsymbol{e}_v)$ as follows.

$$\mathrm{SAT}(f_n|\boldsymbol{e}_v) = \begin{cases} \mathit{true} & \text{if } f_n|\boldsymbol{e}_v \text{ is satisfiable,} \\ \mathit{false} & \text{otherwise} \end{cases} \tag{5.9}$$

And the function validActions is revised as:

$$\mathrm{validActions(p)} = \{v' \mid v' \in \mathrm{Nb}(v_\mathrm{p}), (v_\mathrm{p}, v') \notin \mathrm{p}, \mathrm{SAT}(f_r|\boldsymbol{e}^{p'}) = \mathit{true}\} \tag{5.10}$$

If $\mathrm{SAT}(f_r|\boldsymbol{e}^{p'})$ is false, then v' can be pruned from the action set, as it implies that there is no simple path to destination d that takes the edge (v_p, v') after taking the partial path p while also satisfying domain constraints.

The above query, as such, can be easily performed for an sdd. We do not require a general purpose Boolean satisfiability solver. The sdd package [1] is equipped with a model counting routine that can also take into account the evidence set. This routine has polytime complexity in the number of sdd nodes. If model count is zero, then we can prune v', otherwise it is valid. However, this method was still slow with RL, as a query is made to the sdd at each step of episode sampling. To address this, we present a fast algorithm that performs inference in the sdd in a top-down search fashion, and empirically, is much faster than the standard model counting approach. We also highlight that the inference method we present is general purpose and is not limited to sdds that represent paths or any specific constraint.

Algorithms to check sdd satisfiability Algorithm 5.1 describes a way to compute $\text{SAT}(f_r|\boldsymbol{e})$, and is motivated by the sdd model counting approach. Table 5.1 shows $\text{SAT}(f_n|\boldsymbol{e}_v)$ for a *terminal* sdd *node* n that is normalized for vtree node v (assume v has the variable X). In this case, $\boldsymbol{e}_v$ is either a literal (X or $\neg X$) or an empty instantiation $\emptyset$ (which means the variable X is not in the evidence $\boldsymbol{e}$). Entries in this table can be justified using standard Boolean logic. If X is not in $\boldsymbol{e}$, then we can set X to either true or false.

n	$\text{SAT}(f_n\|X)$	$\text{SAT}(f_n\|\neg X)$	$\text{SAT}(f_n\|\emptyset)$
$\top$	$true$	$true$	$true$
$\bot$	$false$	$false$	$false$
X	$true$	$false$	$true$
$\neg X$	$false$	$true$	$true$

Table 5.1: $\text{SAT}(f_n|\boldsymbol{e}_v)$ for a terminal node given evidence

When n is a decision node with k elements, $\text{SAT}(f_n|\boldsymbol{e}_v)$ is computed as $\bigvee_{i=1}^{k}\big(\text{SAT}(f_{p_i}|\boldsymbol{e}_l)\wedge \text{SAT}(f_{s_i}|\boldsymbol{e}_r)\big)$, where $\boldsymbol{e}_l$ and $\boldsymbol{e}_r$ denote the subsets of evidence $\boldsymbol{e}$ that pertain to the variables of the left and right subtree of v respectively. We prove the correctness of Algorithm 5.1 by induction.

Proof. Base Case: *n is a terminal node.* If n is a terminal node, then $\text{SAT}(\cdot|\cdot)$ is given

[1]https://github.com/wannesm/PySDD

Algorithm 5.1: BU-SAT: bottom-up method

```
Input: SDD $r$ and evidence $\boldsymbol{e}$
// visit children before parents
for node $n$ in the SDD do
    if $n$ is a terminal node then
        $n.value \leftarrow \text{SAT}(f_n|\boldsymbol{e}_v)$
    else
        $n.value \leftarrow \bigvee_{i=1}^{k} p_i.value \wedge s_i.value$
        // $(p_i, s_i)$ is the $i^{th}$ element of node $n$
    end
end
return $r.value$
```

by Table 5.1.

<u>Inductive Step</u>: *n is a decision node.* Assume it has k elements, and is denoted as $(p_1 \wedge s_1) \vee \ldots \vee (p_k \wedge s_k)$. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ denote variables in the left and the right subtree of v respectively. Notation $\vee_{\boldsymbol{x} \models \boldsymbol{e}}$ implies disjunction over all instantiations $\boldsymbol{x}$ that are consistent with evidence $\boldsymbol{e}$.

$$\begin{aligned}
\text{SAT}(f_n|\boldsymbol{e}_v) &= \bigvee_{\boldsymbol{x}\models \boldsymbol{e}_l} \bigvee_{\boldsymbol{y}\models \boldsymbol{e}_r} \text{SAT}(f_n|\boldsymbol{x}\boldsymbol{y}) \\
&= \bigvee_{i=1}^{k} \Big(\bigvee_{\boldsymbol{x}\models \boldsymbol{e}_l} \bigvee_{\boldsymbol{y}\models \boldsymbol{e}_r} \Big(\text{SAT}(f_{p_i}|\boldsymbol{x}) \bigwedge \text{SAT}(f_{s_i}|\boldsymbol{y}) \Big) \Big) && (5.11) \\
&= \bigvee_{i=1}^{k} \Big(\bigvee_{\boldsymbol{x}\models \boldsymbol{e}_l} \text{SAT}(f_{p_i}|\boldsymbol{x}) \Big) \bigwedge \Big(\bigvee_{\boldsymbol{y}\models \boldsymbol{e}_r} \text{SAT}(f_{s_i}|\boldsymbol{y}) \Big) && (5.12) \\
&= \bigvee_{i=1}^{k} \Big(\text{SAT}(f_{p_i}|\boldsymbol{e}_l) \wedge \text{SAT}(f_{s_i}|\boldsymbol{e}_r) \Big) && (5.13)
\end{aligned}$$

Equation (5.11) holds because of the decomposability property of sdd, i.e., variables in prime p_i and sub s_i are disjoint. □

The time complexity of Algorithm 5.1 is linear in the size of sdd as every node n is visited only once. However, in RL, the inference for validActions needs to be done at each time step for each training episode. We observed empirically that this method was slow, and difficult to scale. We next develop our inference method that implements the same logic as in Algorithm 5.1, but in a top-down fashion. Our approach does not always need to visit each sdd node, which makes it faster.

In Algorithm 5.2, for a decision node n (which has elements (p_i, s_i)), we recursively check if $\text{SAT}(f_{p_i}|\boldsymbol{e}_l)=\text{true}$ and if $\text{SAT}(f_{s_i}|\boldsymbol{e}_r)=\text{true}$ for all i. As we only need to answer whether the formula $f_n|\boldsymbol{e}_v$ is satisfiable or not, we can terminate the procedure once we have found that both the conditions are true. We note that we check whether $f_{s_i}|\boldsymbol{y}$ is satisfiable only if $f_{p_i}|\boldsymbol{x}$ is satisfiable. We also store values of terminal and decision nodes that we have already visited and reuse them during the recursion. Each edge in sdd will be visited at most once. Hence, Algorithm 5.2 has linear complexity in the worst case. This simple approach gives us significant improvement in inference speed as compared to Algorithm 5.1. We also optimize this approach over an episode. That is, assume that the edge (v_p, v') is selected at time step t, we can reuse the values of visited nodes during computing $\text{SAT}(f_r|\boldsymbol{e}^{p'})$ for checking validActions at time step $t+1$ (details omitted). These optimizations are not easily integrable in the bottom-up or model counting approach.

5.3 Experiments

We first present results to show the succinctness, efficiency and modularity of our symbolic knowledge compilation framework. Then we show how our knowledge compilation framework works in ZBPF when integrated with previous multi-agent deep-RL methods based on policy gradient(e.g., DCRL) and Q-learning (e.g., MAPQN) on a number of different maps and with different number of agents. We also show this knowledge compilation framework can be integrated with another MARL algorithm (PRIMAL) for solving stand MAPF problem.

Algorithm 5.2: TD-SAT: top-down search

```
Input: sdd node $n$ and evidence $\boldsymbol{e}_v$
if $n.visited$ is false then
    if $n$ is a terminal node then
        $n.value \leftarrow$ SAT($f_n|\boldsymbol{e}_v$)
    else
        $n.value \leftarrow false$
        for element($p_i$,$s_i$) of node $n$ do
            if TD-SAT($p_i, \boldsymbol{e}_l$) is true then
                if TD-SAT($s_i, \boldsymbol{e}_r$) is true then
                    $n.value \leftarrow true$
                    Break
                end
            end
        end
    end
    $n.visited \leftarrow true$
end
return $n.value$
```

5.3.1 Statistics of our symbolic compilation method

Number of paths encoded

Table 5.2 shows that an sdd can encode an exponential number of paths demonstrating its succinctness. For large maps (e.g., 10x10, 20x20), we use hierarchical clustering as noted in Section 5.2. The table shows the number of paths encoded in maps of sizes 10x10 and 20x20 and with landmark constraints. The largest sdd contained 127K nodes (for 20x20, open grid). We were able to compile sdd with very large number of landmarks (80 landmarks in 20x20 grid), the resulting sdd has ∼22K nodes, and is thus tractable to represent and reason with.

Size	Open grid	5 Landmarks
10x10	1.08E+13	2.25E+12
20x20	4.59E+51	1.21E+50

Table 5.2: Number of paths encoded

Simulation Speed

We compare the sampling speeds of TD-SAT, BU-SAT with the speed of two graph-heuristics based inference approaches on open grid maps (5x5, 10x10, and 20x20) and maps with five landmarks. For grid maps, the graph heuristic (GH1) uses Dijkstra's algorithm to determine whether the agent can take the edge next (v_p, v') by checking if there exists a path between v' and the destination d. For maps with landmark constraints, we use an approximation algorithm (GH2) [109] to check if there exists a simple path from the next possible node v' to destination d while going through unvisited landmark(s).

We randomly generated 10K paths given the source (top right node) and destination (bottom left node) using each approach for both open grid maps and maps with 5 landmarks. We run the simulation for each approach on all instances 5 times. We report the average simulation speed (for a total of 10K paths). Table 5.3 shows that our inference approach is faster by an order of magnitude than the model counting based approach BU-SAT. The graph heuristic on open grid maps performs well as checking whether the edge (v_p, v') can be taken by Dijkstra's algorithm runs in time $\Theta((|V| + |E|)\log |V|)$. However, we can use hierarchical clustering to divide 10x10 and 20x20 into smaller graphs so that the simulation speed of our approach is comparable to the graph heuristic. For landmarks, Table 5.4 shows that our approach is the fastest among all approaches. GH2 runs extremely slow on large maps (even with 5 landmarks), and cannot be scaled up to the high density setting.

In Table 5.5, we show the simulation speeds of BU-SAT and TD-SAT for the coverage problem on a straight line graph with 7 nodes with number of landmarks to visit as $N = 3$. The agent needs to visit all the landmarks every $K = 15$ time steps. The simulations are

Approach	5x5	10x10	20x20
BU-SAT	979.7	38873.6	730031.1
TD-SAT	153.1	1313.0	26915.3
GH1	**6.9**	**42.9**	**473.4**

Table 5.3: Simulation speed on open grid maps (in sec)

Approach	5x5	10x10	20x20
BU-SAT	4513.6	49135.6	1282995.0
TD-SAT	**201.9**	**2061.9**	**43619.2**
GH2	1461.0	85142.9	494469.9

Table 5.4: Simulation speed on maps with 5 landmarks (in sec)

done for time horizons $T = 30, 60, 90$. We run each simulation 10,000 times and report the average time. Clearly, TD-SAT performs much faster than BU-SAT in all the instances.

5.3.2 Results of incorporating compiled constraints in RL

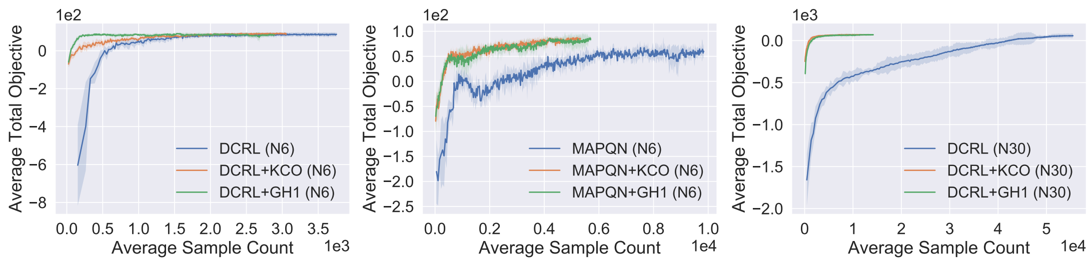
Simple path constraint

We next evaluate the integration of our knowledge-compiled framework (KCO) with DCRL and Q-learning based approach MAPQN [44], which are introduced in Chapter 3, on several open grid maps with varying number of agents. For these experiments, we use the simple path constraints using an sdd. We follow the same ZBPF model and experimental settings as in Chapter 3. The total objective is to minimize sum of arrival time of all agent (SOAT) combined with penalties for congestion. We report the average total objective over all agents vs the average cumulative sample count over all instances on each map.

Open grids We evaluate KCO, GH1 with DCRL and MAPQN on open grid maps. Figure 5.4 clearly shows that both KCO and GH1 make the training process faster and more sample efficient. We note that the difference of the solution quality between KCO and GH1 is not distinct. Moreover, the solution quality is not affected by hierarchical

Approach	$T = 30$	$T = 60$	$T = 90$
BU-SAT	25764.9	54841.6	80584.0
TD-SAT	**2636.6**	**5483.8**	**8272.3**

Table 5.5: Simulation speed for coverage problem (in sec)



(a) 4x4 open grid (DCRL) **(b)** 4x4 open grid (MAPQN) **(c)** 10x10 open grid (DCRL)

Figure 5.4: Sample efficiency results on open grids (N# denotes number of agents)

clustering (10x10 case). Based on this observation, we do not show results on the integration of GH1 with RL approaches on maps with obstacles. As noted in Chapter 3, MAPQN cannot work well on large maps with a large number of agents due to the huge state space. Therefore, we do not run MAPQN on grid sizes more than 5x5.

Obstacles We evaluate KCO with DCRL and MAPQN on a 10x10 map with randomly generated obstacles (density 0.35). Figure 5.6 clearly shows that DCRL and MAPQN can converge much faster with the integration of KCO. Specifically for MAPQN, several agents did not reach their destinations (8.8 agents on average, for N10 case), whereas in MAPQN+KCO, all agents reached destination, which explains much better solution quality by MAPQN+KCO.

Simple path + landmark constraints

In the setting with landmark constraints, to encourage agents to visit all landmarks, we give a higher positive reward if the agent visits a landmark for the first time. Similarly, the agent receive a higher reward if it reaches the destination after visiting a set of landmark nodes in any order via a simple path.

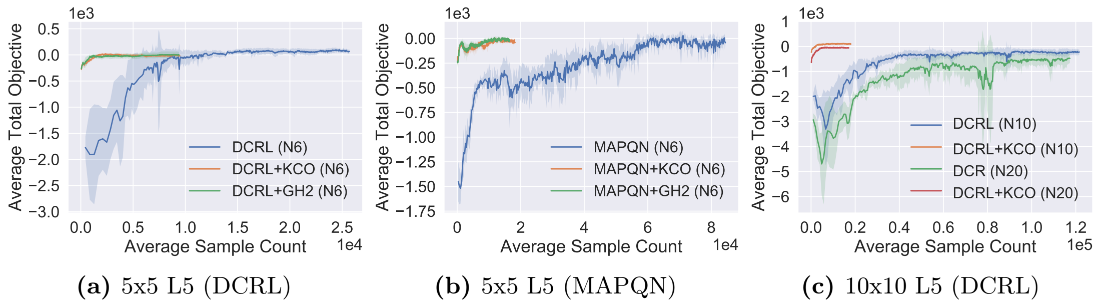


Figure 5.5: Sample efficiency results on maps with landmarks (L# denotes number of landmarks)

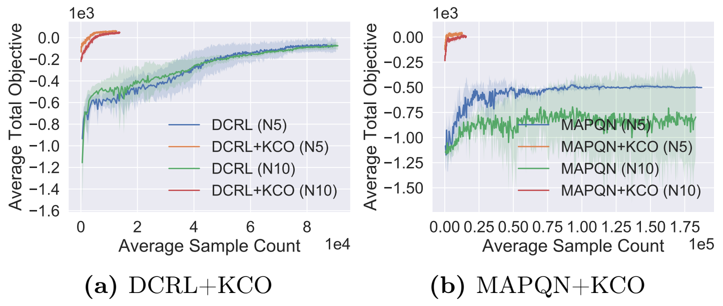
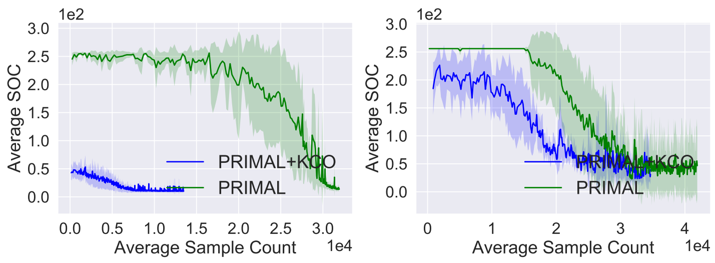


Figure 5.6: Efficiency–10x10 grid with obstacles

We evaluate KCO and GH2 based inference approaches with DCRL and MAPQN algorithms on maps with randomly generated landmarks. GH2 with RL on large maps runs extremely slow, so we omit those results as well. Figure 5.5 shows that DCRL and MAPQN converge using much less samples when integrated with KCO and GH2. The results also confirm that an agent needs many exploration to find a path that goes through all the landmarks when there is no integration of compiled knowledge. We also show the results of the setting where there is a large number of agents. This setting is very challenging as all the agents need to visit the same set of landmarks while also avoiding congestion.

Additional experiments in standard MAPF domain

we also evaluated the effectiveness of integrating our proposed knowledge compilation framework KCO with PRIMAL, a RL approach used to solve standard MAPF problems.



(a) 2 agents **(b)** 4 agents

Figure 5.7: Efficiency–10x10 grid with obstacles (PRIMAL)

It has been noted in previous research that high obstacle density can pose a challenge for PRIMAL. In our experiments, as shown in Figure 5.7, we observed that PRIMAL+KCO outperformed PRIMAL in terms of sample efficiency. However, we noticed that the average sum of costs (SOC) was initially high for both PRIMAL and PRIMAL+KCO, particularly in the N4 case, as agents took actions to avoid congestion.

Chapter 6

Neuro-symbolic for Action Constrained RL

Learning a policy in constrained combinatorial action spaces can be quite challenging as the agent must explore the action space through trial and error to find feasible actions. One possible approach is to solve a Mixed Integer Quadratic Program (MIQP) in the projection step to obtain feasible actions. However, this is intractable in RL as feasible actions must be computed in each time step, and MIQP can be NP-hard. To address this, an approach using QP with integer rounding as action projection has been proposed. While this can obtain an approximate solution, rounding to the nearest integers can cause constraint violations. In Chapter 5, we addressed a constrained RL problem by using an sdd to represent all paths that satisfy the domain constraints and query valid actions through paths. However, we did not handle the general setting of constrained RL with combinatorial action spaces, although the number of paths encoded by sdd is combinatorial, the action space is not.

This chapter introduces a novel neuro-symbolic method for solving constrained RL problems with discrete and combinatorial action spaces. The approach guarantees never to violate any constraint and does not require any costly projection step over the constraint space. Section 6.1 presents the advantages of using probabilistic sentential decision diagrams (psdd) to represent the probability distribution over all valid actions and how to integrate psdd with a policy network in RL. The following section, Section 6.2, details the optimization of psdd parameters using a Q-learning-based approach for factored action spaces. In Section 6.3, we apply the sdd structure, underlying the psdd, to compile various resource constraints efficiently. Finally, in Section 6.4, we compare the performance of our approach with previous methods for action constrained RL, which often violate constraints and are not scalable, demonstrating the superior performance and scalability of our approach.

6.1 Probability distribution over valid actions using decision diagrams

In Chapter 4, we introduced a propositional logic-based framework for defining states, actions, and constraints in RL. Specifically, we assume that a state and an action can be defined using sets of propositional variables denoted as $\boldsymbol{X}_S = \beta_1, \ldots, \beta_n$ and $\boldsymbol{X}_A = \alpha_1, \ldots, \alpha_m$, respectively. Additionally, we assume that there are K constraints, with each constraint represented as a Boolean function $f_k(\boldsymbol{X}_S, \boldsymbol{X}_A)$. Given a state s_β, which is an instantiation of the variables in $\boldsymbol{X}_S$, we define the set of actions that satisfy all constraints, denoted as $\mathcal{C}(s_\beta)$, as follows:

$$\mathcal{C}(s_\beta) = \left\{ a_\alpha \;\middle|\; \bigwedge_{k=1}^{K} f_k|s_\beta(a_\alpha) = 1 \right\} \tag{6.1}$$

where $f_k|s_\beta$ is a subfunction resulting from setting variables of f_k to their values in s_β.

We represent a Boolean action constraint $f_k(\boldsymbol{X}_S, \boldsymbol{X}_A)$ using a general data structure sdds as introduced in Chapter 4. An sdd is a succinct and tractable representation of a Boolean formula. More importantly, we can construct a separate sdd for each f_k $\forall k = 1 : K$. Then we can conjoin all such sdds to represent the single Boolean function (in an sdd form) that represents the set of all valid actions. Such an approach is useful as it allows splitting complex constraints into multiple simpler ones, which can be compactly represented using sdds. Once we conjoin the sdds for all the action constraints f_k $\forall k{=}1{:}K$, we get the final sdd $\mathcal{S}$ for the Boolean formula over (X_S, X_A) encoding all the action constraints. By parameterizing the sdd $\mathcal{S}$, we obtain a *Probabilistic sentential decision diagram* (psdd) as introduced in Chapter 4, which represents a probability distribution over the models of the underlying sdd. Given a world state s_β, we can set variables X_S as per β as evidence $\boldsymbol{e}$. To obtain the distribution $\Delta\mathcal{C}(s_\beta)$, we need to compute the probability of evidence for every node in the psdd [61], and renormalize the psdd parameters based on the probability of evidence.

Theorem 1. *Consider a decision node $n = (p_1, s_1, \theta_1), \ldots, (p_k, s_k, \theta_k)$ that is normalized for vtree node v, with left child l and right child r. Given evidence $\mathbf{e}$, the renormalized parameters θ'_i are given as follows.*

$$\theta'_i = \frac{\Pr_{p_i}(\boldsymbol{e}_l) \cdot \Pr_{s_i}(\boldsymbol{e}_r) \cdot \theta_i}{\sum_{j=1}^{k} \Pr_{p_j}(\boldsymbol{e}_l) \cdot \Pr_{s_j}(\boldsymbol{e}_r) \cdot \theta_j} \tag{6.2}$$

where $\Pr_{p_i}(\boldsymbol{e}_l)$ and $\Pr_{s_i}(\boldsymbol{e}_r)$ denote the probability of evidence for node p_i and node s_i respectively.

Proof. Given evidence $\boldsymbol{e}$, the renormalized parameter θ'_i is defined as conditional probability $\Pr([p_i]|\boldsymbol{e}_v)$ (from Theorem 2 in [61]), which can be written as follows.

$$\theta'_i = \Pr_n([p_i]|\boldsymbol{e}_v) = \frac{\Pr_n(\boldsymbol{e}_v|[p_i]) \cdot \Pr_n([p_i])}{\Pr_n(\boldsymbol{e}_v)} \tag{6.3}$$

Let $\boldsymbol{X}$ denote the variables of the left vtree l, and $\boldsymbol{Y}$ denote the variables of right vtree r. We have,

$$\begin{aligned}
&\Pr_n(\boldsymbol{e}_v|[p_i]) \cdot \Pr_n([p_i]) \\
&= \Big(\sum_{\boldsymbol{x} \models \boldsymbol{e}_l} \sum_{\boldsymbol{y} \models \boldsymbol{e}_r} \Pr_n(\boldsymbol{xy}|[p_i]) \Big) \cdot \theta_i \\
&= \Big(\sum_{\boldsymbol{x} \models [p_i] \wedge \boldsymbol{e}_l} \sum_{\boldsymbol{y} \models \boldsymbol{e}_r} \Pr_n(\boldsymbol{x}|[p_i]) \cdot \Pr_n(\boldsymbol{y}|[p_i]) \Big) \cdot \theta_i && \text{By Proposition 2 in [61]} \\
&= \Big(\sum_{\boldsymbol{x} \models [p_i] \wedge \boldsymbol{e}_l} \sum_{\boldsymbol{y} \models \boldsymbol{e}_r} \Pr_{p_i}(\boldsymbol{x}) \cdot \Pr_{s_i}(\boldsymbol{y}) \Big) \cdot \theta_i && \text{By Proposition 1 in [61]} \\
&= \Big(\sum_{\boldsymbol{x} \models [p_i] \wedge \boldsymbol{e}_l} \Pr_{p_i}(\boldsymbol{x}) \Big) \cdot \Big(\sum_{\boldsymbol{y} \models \boldsymbol{e}_r} \Pr_{s_i}(\boldsymbol{y}) \Big) \cdot \theta_i \\
&= \Pr_{p_i}(\boldsymbol{e}_l) \cdot \Pr_{s_i}(\boldsymbol{e}_r) \cdot \theta_i
\end{aligned}$$

From Theorem 6 in [61], we have

$$\Pr_n(\mathbf{e}_v) = \sum_{i=1}^{k} \Pr_{p_i}(\mathbf{e}_l) \cdot \Pr_{s_i}(\mathbf{e}_r) \cdot \theta_i$$

Therefore, we have the renormalized parameter θ'_i as follows,

$$\theta'_i = \frac{\Pr_n(\boldsymbol{e}_v|[p_i]) \cdot \Pr_n([p_i])}{\Pr_n(\boldsymbol{e}_v)} = \frac{\Pr_{p_i}(\boldsymbol{e}_l) \cdot \Pr_{s_i}(\boldsymbol{e}_r) \cdot \theta_i}{\sum_{j=1}^{k} \Pr_{p_j}(\boldsymbol{e}_l) \cdot \Pr_{s_j}(\boldsymbol{e}_r) \cdot \theta_j}$$

□

6.1.1 Benefits of psdd representation in RL

There are several significant benefits of using a psdd for constrained, combinatorial action spaces in RL. *First*, we can decompose complex constraints over valid actions into multiple simpler constraints. We can construct an sdd *independently* for each constraint, and later conjoin such sdds in a tractable manner to get the final sdd representing all the action constraints [94]. *Second*, we can sample from the distribution $\Delta\mathcal{C}$ over valid actions by following a simple top-down procedure in the psdd $\mathcal{S}$ that has complexity linear in the depth of the psdd [61], which is crucial for fast simulation in an RL environment. *Third*, each generated action sample is *guaranteed* to satisfy all the action constraints by design. This is a major benefit over previous ACRL methods [12, 66] where there is no guarantee that sampled actions will always satisfy constraints, and guaranteeing action constraint satisfaction requires solving expensive mixed-integer math programs for each RL step. *Finally*, the number of parameters in a psdd is linear in the number of edges in the psdd [61]. Thus, for compact psdds, we can represent the distribution over an exponentially large number of models it encodes using a tractable number of parameters. Therefore, psdd parameters can also be optimized relatively easily using RL algorithms compared with directly optimizing over the space of combinatorial actions. We also note that the whole sdd-based knowledge compilation can be done offline before the training of the RL agent starts. Thus, during training, there is no overhead involving manipulation of psdds other than updating its parameters manipulation of psdds other than updating its parameters using the RL algorithm and sampling actions from it. We also show in Section 6.3 that sdd for common types of resource constraints remains tractable, which shows the usefulness of this method for practical applications.

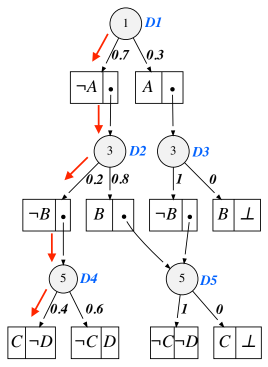
Resource Constraints:

$A + B + C + D = 1,$
$A \leq 1, B \leq 1, C \leq 1, D \leq 1$

Boolean Formula:

$(A \wedge \neg B \wedge \neg C \wedge \neg D)$
$\vee (\neg A \wedge B \wedge \neg C \wedge \neg D)$
$\vee (\neg A \wedge \neg B \wedge C \wedge \neg D)$
$\vee (\neg A \wedge \neg B \wedge \neg C \wedge D)$

(a)



(b)

Figure 6.1: (a) Resource allocation constraints and Boolean formula; (b) Parameterized psdd. Red arrows denote a sampled action from this psdd. We associate a unique id for each decision node ($D1$–$D5$).

6.1.2 Integrating psdd in policy network

The parameters of a psdd compactly encode the distribution over all the valid actions. Our key insight is that, as optimizing directly over all valid actions is intractable, we optimize parameters of the psdd for action constraints, which is tractable given that number of parameters in a psdd is linear in the number of psdd edges [61]. For this purpose, we embed psdd parameters in a deep neural network, which can be later optimized using RL.

We reconsider the resource allocation example (assigning one resource to four entities) as introduced in Chapter 4. The constraints and Boolean formula are describe in Figure 6.1(a). Figure 6.1(b) shows the psdd encoding a probability distribution over all valid actions. We use this psdd as an example to show how psdd parameters can be embeded in a deep neural network.

Figure 6.2 shows the neural network policy which has integrated the parameters. The policy network takes the state as input. The input is followed by two fully connected layers with ReLU activation function. Here, we consider fully connected layers for simplicity, in practice, any other kind (and number) of hidden layers can be used. At the end of the policy network, we create different heads for different psdd decision nodes. Each head has a fully connected layer with *Softmax* activation function, and it outputs the parameters of its corresponding decision node. For example, $D1$ (in Figure 6.1(d)) has two branches, as per psdd semantics, there are two parameters of $D1$, and the Softmax layer for $D1$ has 2 units in policy network. Each policy network head must output a valid probability distribution since the parameters of a decision node are non negative and sum to 1, and determine a distribution (as noted in Chapter 4). Here, each head is also independent of others given the fact that each decision node has its separate local probability distribution (as noted in Chapter 4). In this example, there are three heads for $D1, D2$, and $D4$ respectively in Figure 6.2. $D3$ and $D5$ only have 2 branches, and the *sub* of their second branch is false (in Figure 6.1(d)). Therefore, the parameter for their second branch is always zero, and we do not have to create a Softmax layer for $D3$ and $D5$.

6.1.3 Sampling valid action from psdd distribution

Sampling a valid action from the probability distribution encoded by the psdd can be done using a fast and top-down procedure, which is critical for fast simulation in RL. We take the psdd for resource allocation constraints and its parameters as shown in Figure 6.1(d) as a running example. We follow the below process for action sampling [61]. We start from the psdd root node i.e., $D1$. Since $D1$ is a decision node, we sample one of its branches according to its local distribution $(0.7, 0.3)$. Suppose we have selected the first branch, then we get $\neg A$ (which is a terminal node) and decision node $D2$. We conduct the same sampling process for $D2$. Assume we have selected the first branch of $D2$ with probability 0.2, we get $\neg B$ and decision node $D4$. We continue the sampling process for $D4$. Assume that we sample the first branch of $D4$ with probability 0.4, and get C and $\neg D$. We have now completed the sampling process since we have obtained a model (assignment for all the literals). In this example, the sampled model is $(\neg A, \neg B, C, \neg D)$. To compute the probability of this model, we just need to multiply all parameters that we have encountered during sampling, and we have $Pr(\neg A, \neg B, C, \neg D) = 0.7 \times 0.2 \times 0.4 = 0.056$. The complexity of the sampling is linear

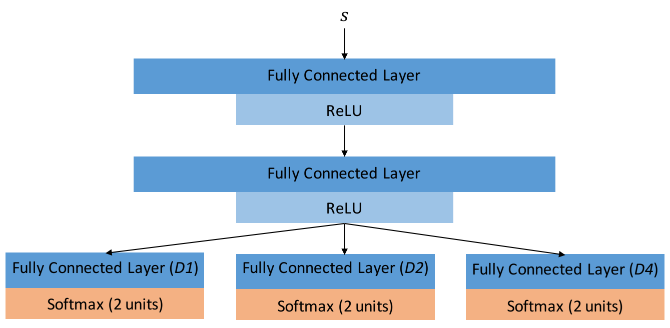


Figure 6.2: psdd integrated Policy Network. Each softmax output head is a distribution over different branches of the corresponding psdd decision node

in the depth of the psdd [61], which is significantly cheaper than sampling naively from a tabular distribution over the combinatorial action space.

Proposition 2. *Let the sampled decision branches for all decision nodes from the policy network be denoted using a_{psdd}. There exists a unique environment action a corresponding to a_{psdd} which satisfies all the action constraints encoded by the underlying* psdd.

Proof. The sampled action a_{psdd} tells which branch to follow for each decision node in the psdd. Consider the sampling process in Section 6.1.3. Starting from the root node of the psdd, we choose the branch as denoted in a_{psdd}. This gives us a *unique* path from root to a leaf node in the psdd. All the literals encountered on this path constitute the unique environment action a corresponding to a_{psdd}. We also know that $\Pr(a) > 0$ (i.e., the action a satisfies all the constraints), as probability of action a is the multiplication of the local probabilities of the chosen branches in a_{psdd} (as noted earlier, details in [61]), which are non-zero by definition (as they are the output of a Softmax layer). □

6.2 Putting it all together: factored action Q-learning

We have shown that the policy network outputs the parameters for each decision node in a psdd and that sampling decision branch of each decision node gives us a psdd action a_{psdd}. Technically, this action cannot be accepted by the RL simulator. However, based on Proposition 2, psdd action a_{psdd} can be mapped into a unique environment action

a. Environment actions are the actions that can be fed into RL simulator. At each step, given the current state s, we can obtain the parameters of the psdd from the policy network. Then we sample the psdd action a_{psdd} and get the uniquely mapped environment action a. We pass a to the RL simulator and receive a reward r. The environment transitions to a new state s'. The transition sample (s, a, r, s') can be stored in an experience replay buffer, and used to train a critic i.e., the Q function. The critic will guide the update of psdd parameters using the policy network as in Figure 6.2.

Policy update Next, we show how to update the psdd parameters using a recently proposed Q-learning based approach for factored action spaces (named AQL) [108]. The high-level idea is as follows. Given a state s, we first search for the psdd action a_{psdd} with the highest Q-value approximately [108], where the Q-value is computed using a_{psdd}'s corresponding environment action a. As the action space is combinatorial, the AQL algorithm uses two heuristics to sample actions:

- Sample actions a_{psdd} using the current policy network.
- Sample actions uniformly from the action space. This is challenging as the action space is both constrained and combinatorial. However, this can be addressed easily using the underlying sdd. The open source library [1] provides functionality to sample variables (environment actions in our case) uniformly from a given sdd, which is fast and tractable in the size of the sdd.

From the action samples collected as above, we can get the best action a^*_{psdd} that has the highest Q-value. We then update the psdd parameters by maximizing the probability of getting the best action a^*_{psdd}. To prevent the output distribution of each decision node from becoming deterministic, we also add an entropy regularization. Assume the policy network π which outputs the psdd parameters is parameterized by θ. The loss function for training the policy is given as follows.

$$\mathcal{L}(\pi_\theta; s) = -\log \pi(a^*_{\text{psdd}}|s, \theta) - \lambda H(\pi(\cdot|s, \theta)) \tag{6.4}$$

[1]https://github.com/art-ai/pypsdd

The first term in the above loss function is the negative log-likelihood with a^*_{psdd} as the target (similar to the cross-entropy for classification task). By minimizing it, it is more likely to sample a psdd action from the policy network with the highest Q-value. The second term is the regularization term which makes the policy network output more diverse distributions for decision nodes. λ is a hyperparameter for the regularization term.

Critic update As we mentioned, the Q-value is computed using the environment action a instead of a_{psdd} due to the deterministic mapping shown in Proposition 2. We use function g to denote the mapping i.e., $a = g(a_{\text{psdd}})$. Assume the Q function is parameterized by ϕ, and we have the loss function for updating the Q function as:

$$\mathcal{L}(\phi^Q) = \mathbb{E}_\pi\Big[\big((r_t + \gamma Q(s_{t+1}, a^*_{t+1})) - Q(s_t, a_t; \phi^Q)\big)^2\Big] \tag{6.5}$$

where $a^*_{t+1} = g(a^*_{\text{psdd},t+1})$ and $a_t = g(a_{\text{psdd},t})$.

The original AQL algorithm is designed for large action spaces. However, it is not specialized to handle combinatorial action spaces *with* complex logical constraints. A key benefit of our combination of AQL with psdd is that we can address rich *constrained* and combinatorial action spaces. Key insights that have helped are to re-interpret psdd parameters as actions to be optimized by AQL, use Proposition 2 to extract the unique environment action, and modify the policy and critic loss functions appropriately. Finally, we emphasize that AQL algorithm is not the only algorithm that can be used to train the policy network in our case. Most RL algorithms designed for factored action spaces can be applied as the sampled action a_{psdd} maps deterministically to an environment action a. It also shows the generality of our developed techniques to optimize psdd parameters.

6.3 Compact sdd encoding of resource constraints

In this section, we describe different resource constraints on integer variables. We also introduce how to translate integer variables to Boolean variables, and constraints to Pseudo-Boolean constraints in a scalable fashion.

Types of resource constraints Three types of resource constraints we consider in this paper are global, regional, and local constraints, as introduced by [12]. *Global constraints* are also known as equality constraints. They ensure that all available resources m are allocated to the n entities: $\sum_{d=1}^{n} a_d = m$ where $d \in D$ is one of the entities and $a_d \in \mathbb{Z}_{\geq 0}$ represents the number of resources allocated to it. *Regional constraints* require a grouping of entities G, so that constraints can be enforced on each group $G_j \in G$. E.g., the case of a regional minimum bound G_j^{min}: $\sum_{d \in G_j} a_d \geq G_j^{min}$. *Local constraints* limit the number of resources that can be allocated to a single entity. E.g., an entity has a maximum capacity d^{max}: $a_d \leq d^{max}$. For both local and regional constraints, maximum, minimum, and equality constraints are possible. Other sensible constraints do also exist, such as resource flow constraints. They are not presented here, even though it is possible to represent them using an sdd.

Resource constraints as Boolean constraints We translate the action representation from an integer vector a to a Boolean representation X_A by creating a Boolean variable b_{di} for every integer $i \in \{1, ..., u_d\}$, where u_d is the highest value a_d can take. We do this for all entities $d \in D$. If we consider a global constraint for m resources and n entities, this will lead to $O(m \times n)$ Boolean variables b_{di} where $d \in \{1, ..., n\}$ and $i \in \{1, ..., m\}$, as $u_d = m$, $\forall d$. A corresponding less than or equal to constraint is $\sum_{d,i} b_{di} \leq m$. Local and regional constraints can be constructed in a similar fashion using analogous Boolean variables.

Translating Boolean constraints into decision diagrams We show a scalable method to compile these Boolean constraints into a compact sdd. Translating these constraints into Boolean formulas and then compiling these into sdds using standard packages [23, 93] did not scale; The resulting sdd was extremely large in size. Instead, we developed a way to create sdds for these Boolean constraints which is inspired by pseudo-Boolean constraints [36]: First, we create a table that represents the Boolean constraint. In the case of a *less than or equal to* constraint, it has $m + 2$ rows and $m * n - m + 1$ columns. An example for $m = 3$ (total resources) and $n = 2$ (total entities) is shown in Table 6.1.

b_{11}	b_{12}	b_{13}	$\top$
b_{12}	b_{13}	b_{21}	$\top$
b_{13}	b_{21}	b_{22}	$\top$
b_{21}	b_{22}	b_{23}	$\top$
$\perp$	$\perp$	$\perp$	

Table 6.1: Table for Boolean constraint for the setting with total 3 resources and 2 entities. Each entity can get 0-3 resources.

In order to check whether an instantiation of all Boolean variables is valid from this table, we start in the top left corner. If we want to set the variable in the current cell to $\top$ (true) (i.e., one unit of resource is consumed), we move one cell downwards. If we want to set it to $\perp$ (false) (i.e., no resource is consumed), we move to the right. This process is repeated until we reach a cell with $\top$ (true) or $\perp$ (false), which tells us whether the variable assignment we tested is valid. We encode this logic into an sdd structure.

This table representation allows us to efficiently compute the information needed to construct an sdd. The total number of decision nodes in this sdd is $(m+1)*(n*m-m)-1$. With this sdd, we can encode $\sum_{i=0}^{n*m-m}\binom{n*m}{m+i}$ models. Thus, in a polynomial-sized sdd (in the number of resources and entities), we can represent combinatorial number of valid resource allocations. For all the constraints, we create sdds in this fashion. We then conjoin (or multiply) all such sdds (which is also a poly-time operation as noted earlier) to encode all the constraints into a single sdd. We parameterize the resulting sdd to obtain the psdd, which is incorporated into our algorithm.

6.4 Experiments

We evaluate our approach on two developed simulation environments for resource allocation: an emergency response system (ERS) [10] and a bike sharing system (BSS) [9]. These environments are used in evaluating the previous proposed resource allocation approaches in [12, 66]. Our code is publicly available in GitHub repository [2].

[2]https://github.com/lingkaching/kcac

Emergency response system This system was originally proposed in [122]. The task here is to provide a target allocation of ambulances to stations in a day. The allocation of ambulances depends on the observation of emergency request demand, the current allocation of ambulances to stations, and the time of the day. This environment also considers adding Gaussian noise (called surge) to the demand request, occurring at a random zone (which a station belongs to) and random time once per day.

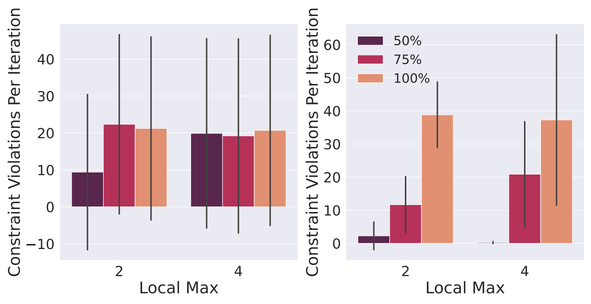
Bike sharing system The bike sharing system was originally introduced in [46]. The task is to provide a target allocation of bicycles to stations to minimize the lost customer demand (no bicycle or no empty parking spot at the station) in a day.

Baselines We compare our proposed framework KCAC against two previous approaches: DDPG with projection using Quadratic Programming [12] and NFWPO [66] where MIQP is used to enforce constraints on resource allocations. The implementations for both approaches are publicly available. There are two other methods developed in [12]: DDPG with constrained Softmax and DDPG approximate OptLayer. However, these two approaches are restricted in the types of resource constraints they can handle (i.e., they cannot be easily extended to more complicated regional constraints). Therefore, we compare against the most general DDPG with QP-based projection.

6.4.1 Emergency response system

In ERS, we consider total 32 ambulances and 25 stations as in [12]. We consider three types of resource constraints as follows.

- Global sum: This constraint says the total resource assigned to all stations must be equal to total available resources.
- Local min and max: This constraint specifies the lower bound and upper bound for the resources assigned to a station.
- Group min and max: In this constraint, several stations are grouped together (which is common in reality), and total resource assigned to stations belonging to the same group should be bounded.

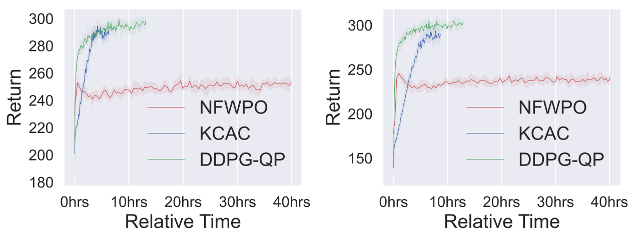


(a) Non-surge **(b)** Surge

Figure 6.3: The number of average constraint violations in DDPG-QP

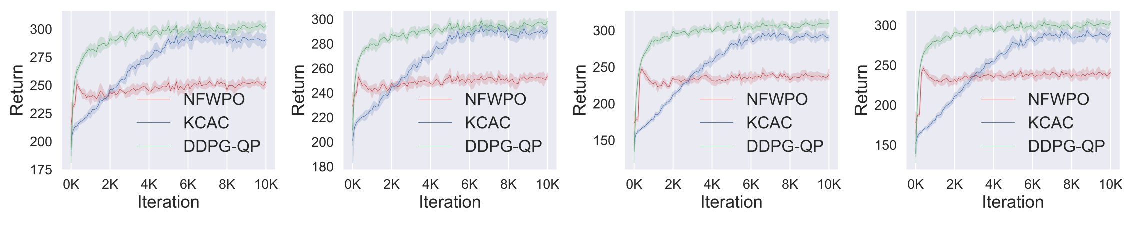
In our experiments, the local min is always 0, and local max is 2 or 4 in different instances for all stations. We also generate different group minimal values in a systematic way as follows. We start with assuming all ambulances are uniformly allocated over all stations so that each station will be assigned 1.28 (32/25) ambulances. Then we modify this average assignment with different percentages e.g., 50%, 75%, and 100%, and we get the group min by multiplying the modified average assignment with total number of stations in the group. Intuitively, the higher the percentage is, the tighter the group min constraint is. In the experiments, we do not consider the group max since we assume a group is able to accommodate a large number of ambulances. Considering the combination of these local max and group min constraints, we have total 6 instances e.g, *2-50%*, *4-50%* etc. For each instance, we first generate different psdds to encode different constraints separately as described in Section 6.3. And we multiply all generated psdds to obtain a single psdd which encodes the actions that satisfy all constraints. In the experiments, we keep the neural network architecture of our policy network and critic network the same as the architecture in DDPG-QP except for the last layer in the policy network for our approach (as noted in Figure 6.2). For NFWPO, we use their default neural network architecture (we changed their architecture to the one used in DDPG-QP, however, it performed worst). We also tune the learning rates for all approaches using grid search over $\{10^{-2}, 10^{-3}, 10^{-4}\}$.

We evaluate all approaches on two ERS scenarios: non-surge and surge, and we run each approach on all instances with 5 different random seeds and report the average. In



(a) Local max = 2 **(b)** Local max = 4

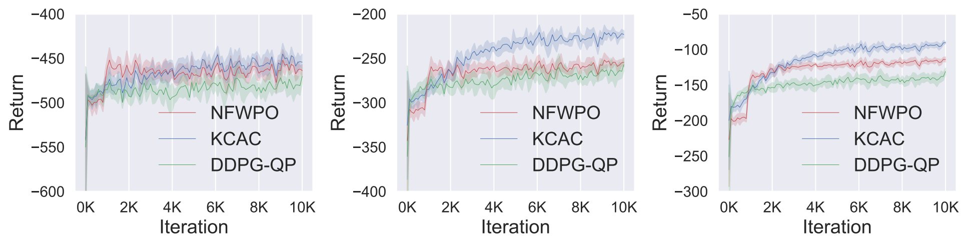
Figure 6.4: Learning process w.r.t run time on ERS with surge



(a) 2-75% **(b)** 2-100% **(c)** 4-75% **(d)** 4-100%

Figure 6.5: Learning process w.r.t iteration on ERS with surge

each run, there are 10k iterations. Figure 6.3 shows the number of average constraint violations per iteration during testing using the final trained policy in DDPG-QP. In both Figure 6.3(a) and (b), x-axis denotes the local maximum resource threshold, y-axis denotes the number of constraint violations, and different bars denote different group min threshold values (50%, 75%, 100%). We can see that there are constraint violations in all instances. The number of constraint violations is getting higher when the group min constraints are becoming tighter. When the percentage for group min is 100%, we see the most constraint violations (except for the non-surge, '2' as local max case). The constraint violations are caused by the rounding procedure after solving a QP in each RL step, and it is very hard to control the rounding given the combinatorial action space. In KCAC and NFWPO, there are zero constraint violations during testing. However, zero constraint violation is achieved via different methods. In our approach KCAC, the action is sampled from the psdd which encodes all valid actions, and it will never violate constraints. However, in NFWPO, MIQP is used to project the action from the policy network to a valid action. Solving a MIQP is NP-hard and is computationally expensive.



(a) 100 bikes **(b)** 150 bikes **(c)** 200 bikes

Figure 6.6: Learning process w.r.t iteration on BSS

Therefore, it takes more time in NFWPO to obtain the same solution quality compared with our approach. In Figure 6.4, we show the learning process of all approaches on the most difficult environments for DDPG-QP (surge, and 100% used in computing group min). In both Figure 6.4(a) and (b), x-axis denotes the relative time elapsed during training w.r.t the starting time, and y-axis denotes the return. We can see that it takes 5 times longer for NFWPO to finish the whole 10k iterations compared with our approach KCAC (∼40hrs v.s. ∼8hrs). To achieve the same solution quality, NFWPO will take more time as well. In other instances for both surge and non-surge cases, NFWPO was 4-5 times slower on average than our method KCAC. Finally, we show the learning process w.r.t. the iteration for all approaches on the surge environment in Figure 6.5. The results on the non-surge environment are very similar to the results on surge environment. As we can see in Figure 6.5, our approach KCAC and DDPG-QP can achieve very close average returns when convergence occurs. However, there are many constraint violations during testing using the final policy in NFWPO. When compared with NFWPO, our approach achieves better solution quality and runs much faster.

6.4.2 Bike sharing system

In BSS experiments, we use the same number of bike stations as in [66] i.e, 5 stations, but we vary the number of bikes e.g, 100, 150 and 200. We consider the global sum constraint and local max constraint in BSS environment. The local max is 23, 35 and 47 for the instance with 100 bikes, 150 bikes and 200 bikes respectively. We evaluate the final trained policy of all three approaches for 1K episodes. In testing, all three approaches can achieve zero constraint violation since the constraints are not complex . However, our approach KCAC performs better in terms of average return. Figure 6.6 shows the learning process by different methods. Our method provides the best return over both DDPG-QP and NFWPO.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The field of sequential decision making with constraints is an active area of research with numerous real-world applications, including air-delivery by drone and resource allocation. In this thesis, we focused on studying the challenges associated with sequential decision-making under various types of constraints in both single-agent and multi-agent settings. To address these challenges, we examined the use of constrained Markov decision processes (CMDP) and constrained decentralized partially observable Markov decision processes (Dec-POMDP) for modeling such problems. However, a major challenge in solving CMDP and constrained Dec-POMDP using RL is finding valid actions that satisfy the constraints. This is especially difficult in large state and action spaces, where trial and error exploration is not efficient, leading to the problem of sample inefficiency. In multi-agent systems, learning control policies even without constraints in a decentralized setting is challenging due to partial observability, uncertainty, and scalability issues. While several scalable multi-agent RL methods have been proposed, few approaches have been developed for large-scale constrained MARL settings since they require credit assignment for both the primary objective and constraints.

The thesis proposes an innovative approach to tackle large-scale multi-agent constrained reinforcement learning (RL) in a collective setting. In addition, a neuro-symbolic method is introduced to address the sample-inefficiency problem resulting from the difficulty in identifying valid actions in each RL step. The contributions of the research are divided into three parts.

- Firstly, the thesis considers a constrained collective Dec-POMDP model and presents a fictitious play based approach, coupled with Lagrangian Relaxation to assign credit to both the primary objective and the constraints.
- Secondly, a zone-based multi-agent pathfinding (ZBPF) framework, which takes inspiration from the drone delivery application, is proposed. The framework is designed to handle a variety of constraints, including capacity, simple path, landmark, and coverage. The thesis proposes a neuro-symbolic method that treats constraints as domain knowledge, which is compiled using a sentential decision diagram (sdd). The compiled knowledge is then integrated with RL algorithms to ensure that agents only select valid actions at each RL step.
- Thirdly, the thesis tackles the problem of action constrained RL with discrete and combinatorial action spaces. Another neuro-symbolic method is presented that uses a probabilistic sentential decision diagram (psdd) to encode a probability distribution over valid actions for any given state. A neural network policy of an agent is learned by optimizing the psdd parameters.

The methods proposed in this thesis provide a significant contribution to solving the challenging problem of large-scale multi-agent constrained RL and address the issue of sample inefficiency when finding valid actions. However, there are still areas where further research could be conducted. The first direction is to extend the current knowledge compilation framework, which utilizes probabilistic sentential decision diagrams, to settings with continuous action spaces. Additionally, we propose an alternative neuro-symbolic integration method that can speed up the policy learning process. The second direction is to tackle infeasible constrained RL problems that arise from the misspecification of constraint thresholds. These future research directions have the potential to significantly enhance the effectiveness and practicality of solving sequential decision making problems using RL.

7.2 Future work

7.2.1 Action constrained RL with continuous action space

In Chapter 6, we proposed a neuro-symbolic method for solving constrained RL problems with discrete and combinatorial action spaces. However, in many real-world applications, constraints are imposed on continuous actions. An example of such an application is the Bipedal walking robots, as mentioned in 1, where the movements of all joints must be synchronized to ensure stable and safe locomotion.

In the Mujoco environments [107], a similar task called Walker2D is presented. The walker is a two-dimensional two-legged figure composed of four main body parts: a single torso at the top, two thighs in the middle, two legs in the bottom, and two feet attached to the legs. The objective is to coordinate the movements of both sets of feet, legs, and thighs to move in the forward (right) direction by applying torques on the six joints that connect the body parts. In this example, we consider state-dependent action constraints, which are expressed as follows:

$$\sum_{i=1}^{6} |v_i w_i| \leq 20 \tag{7.1}$$

where $v_i, i = 1, \ldots, 6$ is the i^{th} dimension of an action and denotes torque applied on the i^{th} joint. w_i denotes the angular velocity of the i^{th} joint and is part of the state.

Since the constraint is defined over continuous variables and is non-linear. The sdd compilation framework for resource allocation constraints, based on pseudo-Boolean constrains, cannot be applied directly. In our future work, we propose a new sdd compilation framework for general actions constraints in continuous action space. The proposed framework consists of four steps. Firstly, we generate data from the action space, which may or may not satisfy the constraint. Next, we binarize the generated data and associate each data point with a label of either 0 or 1 based on whether it satisfies the constraint. Thirdly, we train a neural network using the generated binarized data and labels with binary inputs and step activation functions. Finally, we compile the trained Impneural network into an sdd using the methodology described in [25].

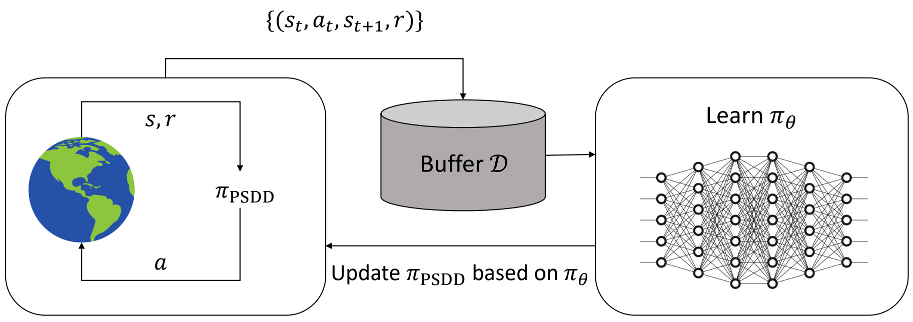

$\{(s_t, a_t, s_{t+1}, r)\}$
s, r
π_{PSDD}
a
Buffer $\mathcal{D}$
Learn π_θ
Update π_{PSDD} based on π_θ

Figure 7.1: The integration of psdd and neural network

7.2.2 Improving the neuro-symbolic integration

In Chapter 6, we presented a factored action Q-learning approach for updating the psdd parameters. However, this method requires sampling a large number of actions from the policy network and uniformly from the action space at each RL step, which can be time-consuming. Additionally, the more actions we sample, the higher the probability that the action with the highest Q-value has already been sampled at least once, leading to a trade-off between simulation speed and algorithm optimality.

To address this limitation, we propose a new approach for integrating psdd and deep neural networks. Let $\pi_{\text{psdd}}(a|s)$ denote the probability of taking an action a in a state s, which is encoded by the psdd. The proposed framework, depicted in Figure 7.1, involves learning another policy π_θ parameterized by θ through three steps. First, we collect transition samples by interacting with the environment using policy π_{psdd}, and we store the samples in a replay buffer. Second, we leverage SAC algorithm to train an offline agent with policy π_θ using the samples in the replay buffer. Finally, we update the psdd parameters by minimizing the KL divergence between π_θ and π_{psdd}, while freezing θ. This process is repeated until convergence. By integrating the strengths of both psdd and deep neural networks, this approach offers a more efficient and effective way of leaning a policy.

7.2.3 Meta-Gradient for infeasible constrained optimization

When applying constrained reinforcement learning to real-world applications, setting constraint thresholds can be difficult due to environmental complexity or the lack of offline verification methods such as simulators. This can result in solutions where all constraints specified in a CMDP cannot be satisfied. However, in some applications, violating constraints is acceptable if the violated behavior is not catastrophic, highlighting the need for soft-constrained RL approaches [34].

Meta reinforcement learning is an active area in the RL community, focusing on learning different meta-parameters to improve RL algorithm performance. These parameters include hyperparameters [117], loss functions [54], exploration strategies [48], and etc. Cross-validation is commonly used to optimize the meta-objective, which can include the expected return or critic loss.

In [20], a meta-gradient approach is proposed to address cases where a constrained RL problem is unsolvable due to misspecified constraint thresholds. When solving a constrained RL problem using Lagrangian Relaxation, they suggest using a meta-parameter to scale the learning rate used in Lagrange multiplier updates. If the problem is unsolvable, the meta-parameter reduces the scaled learning rate to zero, so Lagrangian multipliers are no longer updated. The meta-parameter is learned by optimizing a meta-objective defined in the form of a critic function. However, learning a critic function in RL can be challenging due to the instability problem of using function approximation. It becomes more difficult in multi-agent setting since credit assignment must be performed.

To address this issue, our ongoing work proposes the use of the expected return as the meta-objective for learning the meta-parameter that controls the update of Lagrange multipliers. In this section, we will provide a high-level idea of deriving the meta-gradient for the meta-parameter. Specifically, we consider an unsolvable constrained policy optimization problem as follows.

$$\begin{aligned} \max_{\pi_\theta} \quad & J_R^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \\ \text{s.t.} \quad & J_C^{\pi_\theta} = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)] \le c \end{aligned} \tag{7.2}$$

where c is the misspecified constraint threshold. We note that there does not exist a policy that satisfies the constraint. A modified Lagrangian of Problem 7.2 is given as follows,

$$\mathcal{L}(\pi_\theta, \lambda, \eta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] - \exp(\eta)\lambda(\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t)] - c) \tag{7.3}$$

where η is the meta-parameter. λ is scaled by an exponential term $\exp(\eta)$. The above Lagrangian is a function of three parameters, namely θ, λ, and η. We first perform inner Lagrangian optimization to update the policy parameters θ and Lagrange multiplier λ using gradient based approaches. After we have performed inner optimization. We then proceed with outer optimization to update the meta-parameter η, where the goal is to maximize a meta-objective $J(\tau', \theta', \lambda', \bar{\eta})$ while keeping the meta-parameter $\bar{\eta}$ fixed as a reference value. The meta-objective is defined as follows.

$$J(\tau', \theta', \lambda', \bar{\eta}) = \mathbb{E}_{\tau' \sim \pi_{\theta'}}[\sum_{t=0}^{\infty} \gamma^t (R_t(s_t, a_t) - \exp(\bar{\eta})\lambda' C_t(s_t, a_t))] \tag{7.4}$$

where $\tau' = (s_0, a_0, s_1, a_1, \ldots)$ is the trajectory sampled from the new policy $\pi_{\theta'}$, and λ' is the updated Lagrange multiplier. The meta-gradient of $J(\tau', \theta', \lambda', \bar{\eta})$ w.r.t η is computed using the chain-rule as follows,

$$\frac{\partial J(\tau', \theta', \lambda', \bar{\eta})}{\partial \eta} = \big(\nabla_{\theta'} J(\tau', \theta', \lambda', \bar{\eta})\big)^T \nabla_{\lambda'} \theta' \frac{\partial \lambda'}{\partial \eta} + \frac{\partial J(\tau', \theta', \lambda', \bar{\eta})}{\partial \lambda'} \frac{\partial \lambda'}{\partial \eta}$$

We note that computing $\frac{\partial J(\tau', \theta', \lambda', \bar{\eta})}{\partial \lambda'}$ and $\nabla_{\theta'} J(\tau', \theta', \lambda', \bar{\eta})$ is straightforward and can be done directly. However, computing $\frac{\partial \lambda'}{\partial \eta}$ and $\nabla_{\lambda'} \theta'$ requires the use of gradients used in updating θ and λ.

References

[1] Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. In *International Conference on Machine Learning* (2017), pp. 22–31.

[2] Altman, E. *Constrained Markov Decision Processes.* Chapman and Hall, 1999.

[3] Ames, A. D. Human-inspired control of bipedal walking robots. *IEEE Transactions on Automatic Control* (2014), 1115–1130.

[4] Amos, B., and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning* (2017), pp. 136–145.

[5] Banbara, M., Kaufmann, B., Ostrowski, M., and Schaub, T. Clingcon: The next generation. *Theory and Practice of Logic Programming* (2017), 408–461.

[6] Barták, R., Švancara, J., and Vlk, M. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *International Joint Conference on Autonomous Agents and Multiagent Systems* (2018), pp. 748–756.

[7] Bellman, R. A markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[8] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research 27*, 4 (2002), 819–840.

[9] Bhatia, A. Bike Sharing System (BSS) simualtor gym environment, 2018.

[10] Bhatia, A. Emergency Response System (ERS) simualtor gym environment, 2018.

[11] Bhatia, A., Varakantham, P., and Kumar, A. Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2019), pp. 610–620.

[12] Bhatia, A., Varakantham, P., and Kumar, A. Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2019), pp. 610–620.

[13] Bishop, C. M. *Pattern recognition and machine learning.* Springer-Verlag, Berlin, Heidelberg, 2006.

[14] Bohez, S., Abdolmaleki, A., Neunert, M., Buchli, J., Heess, N., and Hadsell, R. Value constrained model-free continuous control. *arXiv preprint arXiv:1902.04623* (2019).

[15] Bova, S. Sdds are exponentially more succinct than obdds. *arXiv preprint arXiv:1601.00501* (2016).

[16] Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., and Shimony, E. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence* (2015), pp. 740–746.

[17] Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* (1986), 677–691.

[18] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on* (1986), 677–691.

[19] Buffet, O., and Aberdeen, D. The factored policy-gradient planner. *Artificial Intelligence* (2009), 722–747.

[20] Calian, D. A., Mankowitz, D. J., Zahavy, T., Xu, Z., Oh, J., Levine, N., and Mann, T. Balancing constraints and rewards with meta-gradient d4pg. In *International Conference on Learning Representations* (2020).

[21] Cap, M., Novak, P., Kleiner, A., and Selecky, M. Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots. *IEEE Transactions on Automation Science and Engineering 12*, 3 (jul 2015), 835–849.

[22] Chang, Y. H., Ho, T., and Kaelbling, L. P. All learning is local: Multi-agent learning in global reward games. In *Advances in Neural Information Processing Systems* (2004), pp. 807–814.

[23] Choi, A. The PyPSDD Package, 2018.

[24] Choi, A., Shen, Y., and Darwiche, A. Tractability in structured probability spaces. In *Advances in Neural Information Processing Systems* (2017), pp. 3477–3485.

[25] Choi, A., Shi, W., Shih, A., and Darwiche, A. Compiling neural networks into tractable boolean circuits. *intelligence* (2017).

[26] Choi, A., Tavabi, N., and Darwiche, A. Structured features in naive bayes classification. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2016), pp. 3233–3240.

[27] Chow, Y., Ghavamzadeh, M., Janson, L., and Pavone, M. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research* (2017), 6070–6120.

[28] Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. A lyapunov-based approach to safe reinforcement learning. *Advances in Neural Information Processing Systems* (2018).

[29] Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757* (2018).

[30] Darwiche, A. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI* (2011), pp. 819–826.

[31] Diddigi, R. B., Danda, S. K. R., Bhatnagar, S., et al. Actor-critic algorithms for constrained multi-agent reinforcement learning. *arXiv preprint:1905.02907* (2019).

[32] Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research 13* (2000), 227–303.

[33] Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[34] Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881* (2020).

[35] Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning* (2021), 2419–2468.

[36] Eén, N., and Sörensson, N. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation* (2006), 1–26.

[37] Estlin, T., Gaines, D., Chouinard, C., Castano, R., Bornstein, B., Judd, M., Nesnas, I., and Anderson, R. Increased mars rover autonomy using ai planning, scheduling and execution. In *International Conference on Robotics and Automation* (2007), IEEE, pp. 4911–4918.

[38] Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., Sturtevant, N., Wagner, G., and Surynek, P. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Annual Symposium on Combinatorial Search, SoCS* (2017), pp. 29–37.

[39] Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. Dl2: training and querying neural networks with logic. In

International Conference on Machine Learning (2019), pp. 1931–1941.

[40] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018).

[41] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018), pp. 2974–2982.

[42] Frank, M., and Wolfe, P. An algorithm for quadratic programming. *Naval Research Logistics Quarterly* (1956), 95–110.

[43] Fu, H., Tang, H., Hao, J., Lei, Z., Chen, Y., and Fan, C. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. In *International Joint Conference on Artificial Intelligence* (2019), pp. 2329–2335.

[44] Fu, H., Tang, H., Hao, J., Lei, Z., Chen, Y., and Fan, C. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. In *IJCAI* (2019), pp. 2329–2335.

[45] Gattami, A., Bai, Q., and Agarwal, V. Reinforcement learning for multi-objective and constrained markov decision processes. *arXiv preprint arXiv:1901.08978* (2019).

[46] Ghosh, S., and Varakantham, P. Incentivizing the use of bike trailers for dynamic repositioning in bike sharing systems. In *Proceedings of the International Conference on Automated Planning and Scheduling* (2017), vol. 27, pp. 373–381.

[47] Gill, P. E., Murray, W., and Saunders, M. A. SNOPT: An SQP algorithm for large-scale constrained optimization (Reprinted from SIAM Journal Optimization, vol 12, pg 979-1006, 2002). *Siam Review 47*, 1 (2005), 99–131.

[48] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. Meta-reinforcement learning of structured exploration strategies. *arXiv preprint arXiv:1802.07245* (2018).

[49] Gupta, T., Kumar, A., and Paruchuri, P. Planning and learning for decentralized mdps with event driven rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018), pp. 6186–6194.

[50] Gupta, T., Kumar, A., and Paruchuri, P. Successor features based multi-agent rl for event-based decentralized mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 6054–6061.

[51] Harremoes, P. Binomial and poisson distributions as maximum entropy distributions. *IEEE Transactions on Information Theory 47*, 5 (2001), 2039–2041.

[52] Hoernle, N., Karampatsis, R. M., Belle, V., and Gal, K. Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2022), pp. 5700–5709.

[53] Honig, W., Kumar, T. K., Cohen, L., Ma, H., Xu, H., Ayanian, N., and Koenig, S. Multi-agent path finding with kinematic constraints. In *International Conference on Automated Planning and Scheduling* (2016), pp. 477–485.

[54] Houthooft, R., Chen, R. Y., Isola, P., Stadie, B. C., Wolski, F., Ho, J., and Abbeel, P. Evolved policy gradients. *arXiv preprint arXiv:1802.04821* (2018).

[55] Inoue, T., Iwashita, H., Kawahara, J., and Minato, S.-i. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer 18*, 1 (2016), 57–66.

[56] Jaynes, E. T. Information theory and statistical mechanics. *Phys. Rev. 106*, 4 (May 1957), 620–630.

[57] Jin, M., Ma, Z., Jin, K., Zhuo, H. H., Chen, C., and Yu, C. Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2022), pp. 7042–7050.

[58] Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. Unity: A General Platform for Intelligent Agents. *CoRR abs/1809.0* (2018).

[59] Kassir, S., de Veciana, G., Wang, N., Wang, X., and Palacharla, P. Enhancing cellular performance via vehicular-based opportunistic relaying and load balancing. In *INFOCOM IEEE Conference on Computer Communications* (2019), pp. 91–99.

[60] Kautz, H. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine* (2022), 105–125.

[61] Kisa, D., Van Den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In *Principles of Knowledge Representation and Reasoning* (2014), pp. 558–567.

[62] Kormushev, P., Ugurlu, B., Calinon, S., Tsagarakis, N. G., and Caldwell, D. G. Bipedal walking energy minimization by reinforcement learning with evolving policy parameterization. In *International Conference on Intelligent Robots and Systems* (2011), IEEE, pp. 318–324.

[63] Li, B., Tang, H., Zheng, Y., Hao, J., Li, P., Wang, Z., Meng, Z., and Wang, L. Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation. *arXiv preprint arXiv:2109.05490* (2021).

[64] Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S., and Koenig, S. Lifelong multi-agent path finding in large-scale warehouses. In *International Joint Conference on Autonomous Agents and Multiagent Systems* (2020), pp. 1898–1900.

[65] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[66] Lin, J.-L., Hung, W., Yang, S.-H., Hsieh, P.-C., and Liu, X. Escaping from zero gradient: Revisiting action-constrained reinforcement learning via frank-wolfe

policy optimization. In *Uncertainty in Artificial Intelligence* (2021), pp. 397–407.

[67] Lipp, T., and Boyd, S. Variations and extension of the convex-concave procedure. *Optimization and Engineering 17*, 2 (2016), 263–287.

[68] Liu, C., Geng, N., Aggarwal, V., Lan, T., Yang, Y., and Xu, M. Cmix: Deep multi-agent reinforcement learning with peak and average constraints. In *ECML PKDD* (2021).

[69] Liu, Y., Ding, J., and Liu, X. Ipo: Interior-point policy optimization under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), pp. 4940–4947.

[70] Liu, Y., Halev, A., and Liu, X. Policy learning with constraints in model-free reinforcement learning: A survey. In *International Joint Conference on Artificial Intelligence* (2021).

[71] Lu, S., Zhang, K., Chen, T., Basar, T., and Horesh, L. Decentralized policy gradient descent ascent for safe multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021).

[72] Lyu, D., Yang, F., Liu, B., and Gustafson, S. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 2970–2977.

[73] Ma, H., Koenig, S., Ayanian, N., Cohen, L., Hoenig, W., Kumar, T. K. S., Uras, T., Xu, H., Tovey, C., and Sharon, G. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios, 2017.

[74] Ma, H., Kumar, T. K., and Koenig, S. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2017), pp. 3605–3612.

[75] Meyers, C. A., and Schulz, A. S. The complexity of congestion games. *Networks* (2012).

[76] Nguyen, D. T., Kumar, A., and Lau, H. C. Policy gradient with value function approximation for collective multiagent planning. In *International Conference on Neural Information Processing Systems* (2017), pp. 4322–4332.

[77] Nguyen, D. T., Kumar, A., and Lau, H. C. Policy gradient with value function approximation for collective multiagent planning. In *International Conference on Neural Information Processing Systems* (2017), pp. 4322–4332.

[78] Nishino, M., Yasuda, N., Minato, S.-i., and Nagata, M. Zero-suppressed sentential decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2016), pp. 1058–1066.

[79] Nishino, M., Yasuda, N., Minato, S.-i., and Nagata, M. Compiling graph substructures into sentential decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2017), pp. 1213–1221.

[80] Oztok, U., and Darwiche, A. A top-down compiler for sentential decision diagrams. In *IJCAI* (2015), pp. 3141–3148.

[81] Pant, Y. V., Abbas, H., Quaye, R. A., and Mangharam, R. Fly-by-logic: control of multi-drone fleets with temporal logic objectives. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems* (2018), IEEE, pp. 186–197.

[82] Paternain, S., Chamon, L. F., Calvo-Fullana, M., and Ribeiro, A. Constrained reinforcement learning has zero duality gap. In *International Conference on Neural Information Processing Systems* (2019), pp. 7555–7565.

[83] Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. Learning to Cooperate via Policy Search. In *UAI* (2000), pp. 489–496.

[84] Pham, T.-H., De Magistris, G., and Tachibana, R. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *International Conference on Robotics and Automation* (2018), IEEE, pp. 6236–6243.

[85] Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. QMIX: monotonic value function factorisation for deep

multi-agent reinforcement learning. In *ICML* (2018), pp. 4292–4301.

[86] Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Satish Kumar, T. K., Koenig, S., and Choset, H. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters* (2019), 2378–2385.

[87] Say, B., and Sanner, S. Compact and efficient encodings for planning in factored state and action spaces with learned binarized neural network transition models. *Journal of Artificial Intelligence Research 285* (2020).

[88] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning* (2015), pp. 1889–1897.

[89] Schwarting, W., Alonso-Mora, J., and Rus, D. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems* (2018), 187–210.

[90] Sedgewick, R. *Algorithms in C, part 5: graph algorithms.* Pearson Education, 2001.

[91] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence 219* (2015), 40–66.

[92] Sharon, G., Stern, R., Goldenberg, M., and Felner, A. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *International Joint Conference on Artificial Intelligence* (2011), pp. 662–667.

[93] Shen, Y. PSDD, 2020.

[94] Shen, Y., Choi, A., and Darwiche, A. Tractable operations for arithmetic circuits of probabilistic models. In *Advances in Neural Information Processing Systems* (2016), pp. 3943–3951.

[95] Shen, Y., Goyanka, A., Darwiche, A., and Choi, A. Structured bayesian networks: From inference to learning with routes. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 7957–7965.

[96] Silva, A., and Gombolay, M. Encoding human domain knowledge to warm start reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021), pp. 5042–5050.

[97] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *International Conference on Machine Learning* (2014), Pmlr, pp. 387–395.

[98] Singh, A. J. Multiagent decision making for maritime traffic management. https://github.com/rlr-smu/camarl/tree/main/PG_MTM, 2019.

[99] Singh, A. J., Kumar, A., and Lau, H. C. Hierarchical multiagent reinforcement learning for maritime traffic management. In *the International Conference on Autonomous Agents and Multiagent Systems* (2020).

[100] Singh, A. J., Nguyen, D. T., Kumar, A., and Lau, H. C. Multiagent decision making for maritime traffic management. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), pp. 6171–6178.

[101] Surynek, P. Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. *Annals of Mathematics and Artificial Intelligence 81*, 3-4 (dec 2017), 329–375.

[102] Surynek, P., Kumar, T. K. S., and Koenig, S. Multi-Agent Path Finding with Capacity Constraints, 2019.

[103] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[104] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems* (2000), pp. 1057–1063.

[105] Teng, T.-H., Lau, H. C., and Kumar, A. Coordinating Vessel Traffic to Improve Safety and Efficiency. In *Internation Conference on Autonomous Agents and Multiagent Systems* (2017), pp. 141–149.

[106] Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. In *International Conference on Learning Representations* (2018).

[107] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *International conference on intelligent robots and systems* (2012), IEEE, pp. 5026–5033.

[108] Van de Wiele, T., Warde-Farley, D., Mnih, A., and Mnih, V. Q-learning in enormous action spaces via amortized approximate maximization. *arXiv preprint arXiv:2001.08116* (2020).

[109] Vardhan, H., Billenahalli, S., Huang, W., Razo, M., Sivasankaran, A., Tang, L., Monti, P., Tacca, M., and Fumagalli, A. Finding a simple path with multiple must-include nodes. In *IEEE MASCOTS* (2009), pp. 1–3.

[110] Volodymyr, M., Koray, K., David, S., Rusu Andrei A, Joel, V., Bellemare Marc G, Alex, G., Martin, R., Fidjeland Andreas K, and Georg, O. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[111] Wang, J., Li, J., Ma, H., Koenig, S., and Kumar, S. A New Constraint Satisfaction Perspective on Multi-Agent Path Finding: Preliminary Results. In *International Joint Conference on Autonomous Agents and Multiagent Systems* (2019).

[112] Wang, W., Wu, G., Wu, W., Jiang, Y., and An, B. Online collective multiagent planning by offline policy reuse with applications to city-scale mobility-on-demand systems. In *International Joint Conference on Autonomous Agents and Multiagent Systems* (2022).

[113] Wurman, P. R., D'Andrea, R., and Mountz, M. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine 29*, 1 (2008), 9–19.

[114] Xu, H., and Mannor, S. Probabilistic goal markov decision processes. In *International Joint Conference on Artificial Intelligence* (2011), Citeseer, p. 2046.

[115] Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Broeck, G. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning* (2018), PMLR, pp. 5502–5511.

[116] Xu, T., Liang, Y., and Lan, G. Crpo: A new approach for safe reinforcement learning with convergence guarantee. In *International Conference on Machine Learning* (2021), pp. 11480–11491.

[117] Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801* (2018).

[118] Yang, F., Lyu, D., Liu, B., and Gustafson, S. Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *arXiv preprint arXiv:1804.07779* (2018).

[119] Yang, T.-Y., Rosca, J., Narasimhan, K., and Ramadge, P. J. Projection-based constrained policy optimization. In *International Conference on Learning Representations* (2019).

[120] Yehoshua, R., and Agmon, N. Multi-robot adversarial coverage. In *Proceedings of the European Conference on Artificial Intelligence.*

[121] Yu, J., and Lavalle, S. M. Planning optimal paths for multiple robots on graphs. *IEEE International Conference on Robotics and Automation* (2013), 3612–3617.

[122] Yue, Y., Marla, L., and Krishnan, R. An efficient simulation-based approach to ambulance fleet allocation and dynamic redeployment. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2012), pp. 398–405.

[123] Yuille, A. L., and Rangarajan, A. The Concave-Convex Procedure (CCCP), url = http://papers.nips.cc/paper/2125-the-concave-convex-procedure-cccp, year = 2001. In *Advances in Neural Information Processing Systems*, pp. 1033–1040.