1-24-2024

Unmanned Aerial Systems (UAS) Traffic Density Prediction and Multi-Agent Task Allocation

Chen Luo
Syracuse University

Abstract

In recent years, the field of Multi-Agent Systems (MAS) has garnered increasing attention. The essence of Multi-Agent Systems lies in orchestrating the coordination and collaboration of autonomous agents, endowing them with the ability to work collectively towards common goals. This characteristic makes MAS particularly germane in addressing the challenges posed by complex and dynamic environments, where the adaptability and decentralized decision-making of autonomous agents prove advantageous. To delve into the intricacies of MAS, three primary research directions have emerged. Firstly, we aim to develop a model rapidly and accurately to forecast future Unmanned Aerial Systems (UAS) traffic density patterns while simultaneously simplifying model complexity. The second direction involves the study of real-time task allocation and trajectory planning algorithms, considering constraints imposed by the capabilities of individual agents. The third direction focuses on investigating multi-agent cooperative game using Multi-Agent Reinforcement Learning (MARL).

The success of the deep learning-based approach spans various domains, notably in areas such as density and trajectory prediction. In our earlier work, we introduced an innovative trajectory prediction model, which forecasts instantaneous traffic density using mission schedule information. However, one of the main drawbacks of the Deep Neural Network (DNN) is the high computational cost, which prevents us from applying the model to search for the best mission plan or best locations of launching and landing zones, because it requires exponentially large numbers of predictions based on different input combinations. To reduce the complexity of the Convolutional Neural Network (CNN) model, we developed a Neural Architecture Search (NAS) optimization framework. This framework systematically

identifies the optimal compression ratio for each layer, resulting in a streamlined neural architecture. As a result, we achieved a 50% reduction in the size of the instantaneous traffic density prediction model.

Multi-agent task allocation challenges can be accomplished through the application of the Consensus-Based Bundle Algorithm (CBBA). The distributed algorithm exhibits provable convergence and ensures 50% optimality when the score function adheres to the conditions of Diminishing Marginal Gain (DMG). While prior research has primarily focused on the unconstrained optimization of rewards, our work addresses the challenges posed by real-world dynamic environments by incorporating specific constraints. These constraints encompass considerations such as limitations on agent capabilities, communication restrictions, and budget constraints. Our work is to applying CBBA for task allocation while considering budget constraints using various heuristics extensions to the bidding algorithm. In determining the most suitable heuristic extension, we introduce a Graph Convolutional Neural Network (GCN) model to extract and analyze features of constrained optimization problems presented as graphs, predicting the potential performance (i.e., global reward) of different heuristic extensions. Experimental results affirm a correlation exceeding 0.98 between predicted and actual rewards. The prediction-guided selection consistently identifies the most effective heuristic extension in 70% of cases for budget-constrained task allocation problems.

In a MAS, agents share their local observations to gain global situational awareness for decision making and collaboration using a message passing system. When to send a message, how to encode a message, and how to leverage the received messages directly affect the effectiveness of the collaboration among agents. When employing Reinforcement Learning (RL) to train a multi-agent cooperative game, optimizing the message passing system becomes integral to agent policy enhancement. We propose the Belief-map Assisted Multi-agent System (BAMS), which leverages a neuro-symbolic belief map to enhance training. Compared to the sporadic and delayed feedback coming from the reward in RL, the feedback from the belief map is more consistent and reliable. Agents utilizing BAMS can learn a more

effective message passing network, enhancing mutual understanding and improving overall game performance. We assess BAMS in a cooperative predator and prey game with varying map complexities, comparing its performance to previous multi-agent message passing models. Simulation results demonstrate that BAMS reduces training epochs by 66%, and agents employing the BAMS model complete the game with 34.62% fewer steps on average.

UNMANNED AERIAL SYSTEMS (UAS) TRAFFIC DENSITY PREDICTION AND MULTI-AGENT TASK ALLOCATION

By

Chen Luo

B.S., Hunan University, 2015

M.S., Syracuse University, 2018

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

December 2023

Acknowledgements

This long and challenge journey finally is over, a new chapter is on the horizon. At the age of thirty, I entered into marriage, completed my Ph.D, and embarked on my professional journey. I will forever cherish the moment when, in April 2018, Dr. Qinru Qiu offered the opportunity for me to pursue doctoral studies as I took a significant step forward in academia. This marked the beginning of a transformative journey, and I consider myself incredibly fortunate to have been her student and a member of the Advanced Microprocessor and Power-aware Systems (AMPS) lab. Along this path, I've been fortunate to receive generous support from numerous individuals. Their unwavering assistance has been indispensable to the realization of this thesis. Therefore, I would like to extend my deepest gratitude to those who have guided and supported me wholeheartedly, enabling me to successfully navigate the path to the completion of my Ph.D.

Firstly, I would like to express my sincere gratitude to my doctoral advisor, Dr. Qinru Qiu, for her guidance, support and endless encouragement that helped me through my study, research, and life during this doctoral endeavour. During this long period, I have gained much experience in research, and completed many studies in the area of computer vision, multi-agent system and reinforcement learning.

Secondly, I would like to thank the rest of my doctoral committee members, Dr. Carlos Enrique Caicedo Bastidas, Dr. Garrett Ethan Katz, and Dr. Mustafa Cenk Gursoy for their encouragement and insightful comments.

Then, I would like to extend my sincere thanks to all collaborators, labmates and friends at Syracuse University: Dr. Zhe Li, Dr. Haowen Fang, Dr. Krittaphat Pugdeethosapol, Dr. Yilan Li, Dr. Ziyi Zhao, Dr. Amar Shrestha, Zhao Jin, Mingyang Li, Zaidao Mei, Daniel

Patrick Rider, Yue Ma, Qinwei Huang, Jiayang Liu, Zhenhang Zhang and Rui Zuo. I am very grateful for their help and teamwork.

My gratitude also goes to many friends both on the other side of the earth and on the west coast, who have served as receptacles for my negative emotions over the years. Thank you for your unwavering support, attentive listening, encouragement, and tolerance. I appreciate your understanding and helps.

And most importantly, I would like to extend enduring gratitude to my family for their unwavering support and encouragement. This thesis is dedicated to my wife, Suhong Tan, and my parents, Xia Chen and Jian Luo, who have been by my side, offering encouragement and steadfast support throughout these years. I am deeply thankful for the love and patience they have generously shown me during this significant undertaking. Their encouragement and support have meant the world to me. Thank you wholeheartedly!

Abbreviations

AUROC Area Under the Receiver Operating Characteristics

BAMS Belief-map Assisted Multi-agent System

CBBA Consensus-Based Bundle Algorithm

CNN Convolutional Neural Network

DMG Diminishing Marginal Gain

DNN Deep Neural Network

GCN Graph Convolutional Neural Network

GPU Graphics Processing Unit

IOT Internet of Things

LSTM Long Short-Term Memory

MARL Multi-Agent Reinforcement Learning

MAS Multi-Agent Systems

NAS Neural Architecture Search

RL Reinforcement Learning

RNN Recurrent Neural Network

SoC System on Chip

SSL Structured Sparsity Learning

sUAS Small Unmanned Aerial Systems

UAS Unmanned Aerial Systems

UAV Unmanned Aerial Vehicles

UTM Unmanned Aerial System Traffic Management

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the landscape of artificial intelligence and autonomous systems, the concept of Multi-Agent Systems (MAS) has emerged as a pivotal paradigm for modeling complex, decentralized interactions among autonomous entities. A Multi-Agent System comprises multiple agents, each possessing individual capabilities to perceive, reason, and act autonomously. These agents interact with one another, often in a decentralized fashion, to collectively achieve goals that may be beyond the capacity of any single agent. The rich diversity of applications, from robotics and economics to traffic management and beyond, underscores the versatility and relevance of multi-agent systems in addressing real-world challenges.

The roots of multi-agent systems trace back to the need for modeling and simulating interactions in scenarios where multiple entities, each with its own objectives and decision-making processes, coexist. Traditional approaches to problem-solving often fall short when confronted with the intricacies of real-world complexities, where interactions are dynamic, information is distributed, and decision spaces are vast. Multi-agent systems provide a flexible and scalable framework for representing and analyzing such scenarios.

The appeal of multi-agent systems lies in their ability to capture the dynamics of systems where autonomy, interaction, and adaptation are paramount. In robotics, multi-agent systems enable the coordination of autonomous robots to accomplish tasks collectively, fos-

tering collaborative approaches to complex challenges. In economics, these systems serve as powerful tools for modeling market dynamics, where diverse agents engage in transactions and negotiations. Furthermore, in decentralized control and coordination problems, such as traffic management and industrial automation, multi-agent systems offer a distributed solution that can enhance efficiency and adaptability.

1.1 Motivations

1.1.1 UAS Traffic Density Pattern Prediction

Recently, many companies devote themselves to develop Small Unmanned Aerial Systems (sUAS). Complicated and high density UAS traffic imposes significant burden on air traffic management, city planning and communication resource allocation. In this environment, traffic management has shown significant importance. Many existing works study issues such as sUAS navigation, obstacle avoidance or UAS traffic management, by developing a corresponding simulator with fair time complexity. In [1], the authors presented an indoor algorithm to navigate single sUAS to avoid collisions. [2] proposed a solution to avoid collisions in a static environment by importing geometrical constraints. Other single sUAS classical approaches applied rapidly-exploring random trees [3] and Voronoi graphs [4, 5]. Multiple sUAS trajectory simulation has been studied as a multi-agent cooperative system and solved in a rolling horizon approach using dynamic programming [6] or mixed integer linear programming [7]. Other strategies [8, 9] involved real-time routing algorithms with communication and airspace safety considerations. The major goals of my research consists of the following aspects: (1) Coordinate and control Unmanned Aerial Vehicles (UAV) missions to avoid potential confliction; (2) ensure timely completion of mission with minimum flight energy with low-cost real-time centralized or decentralized management; and (3) provide an accurate forecast of UAV traffic.

UAS density prediction is a critical and challenging problem in the Unmanned Aerial

System Traffic Management (UTM) system. Most existing studies focus on simulation-based approaches. Although accurate, they usually take a long time to deliver results. Neural networks have been used to predict traffic density [10]. However, most such studies require the sampling of the traffic density from the past data and predict the future density using past density information. These models assume a static environment. For example, the source (i.e. the location where the sUAS enters the air space) and sink (i.e. the location where the sUAS leaves the air space) of the traffic flow are assumed to remain the same, and air space constraints, such as no-fly zones, are fixed. Based on these assumptions, the traffic in the future will exhibit a similar pattern as the traffic in the past and can be predicted from the historical data. A constant environment may be reasonable for road traffic, however, the operational environment of sUAS features higher dynamics and flexibility. The model based on historical data will become obsolete as soon as the environment changes. New data must be collected and a new model needs to be trained, which can take days or weeks. Furthermore, most of the existing models consider traffic distribution as a stationary process, and focus on predicting the steady states. For resource provisioning or safety assurance, we need to know not only the steady state traffic but also the worst case traffic. Hence the ability to predict the transient behavior of air traffic distribution is highly desirable.

1.1.2 UAS Density Prediction Model Compression and Architecture Search

Model compression has been an active area of research with various techniques developed to address the challenges of reducing the size of a DNN model without significantly compromising its performance. This is crucial for deploying models on edge devices, Internet of Things (IOT) devices [11] and System on Chip (SoC) devices [12], where memory and computational resources are limited. Techniques for model compression include quantization [13], pruning [14, 15], and knowledge distillation [16], which aim to make models more efficient in terms of storage and inference speed while maintaining their accuracy to the extent possible.

Although our previous model gives high quality traffic density prediction, the complexity is also extremely high. Most of the computation is on the 6 CNNs for mission feature extraction. Our goal is to reduce the size of the CNNs to lower the model complexity and accelerate the prediction without sacrificing its prediction accuracy. This is achieved by pruning (unnecessary) filters in the CNN layers. Filter pruning [17] is chosen here because it results in a dense weight matrix, thereby facilitating Graphics Processing Unit (GPU) acceleration. Furthermore, pruning a filter in layer i results in the removal of a channel in layer $i + 1$ without creating sparsity. We aims to search for the best prune strategy for the UAV traffic density prediction model to facilitate fast traffic prediction.

1.1.3 Multi-agent Task Allocation with Budget Constraints

Task allocation in missions is a pivotal aspect of optimizing the collaborative efforts of multiple entities toward a shared goal. This process involves strategically assigning tasks or responsibilities to individual components within a system. The objective is to leverage the unique skills, capabilities, and current states of each entity to achieve maximum efficiency and effectiveness in mission execution.

The original CBBA algorithm only considers unconstrained optimization. It assumes that each task has a reward (a) and a cost (p), and tries to maximize the score defined as the difference of the rewards and costs (i.e., $a - p$.) But the real-world dynamic environments always have specific constraints considerations such as limitations on agent capabilities, communication restrictions, and budget constraints.For example, a team of unmanned aerial vehicles (UAVs) set out to dispatch life supplies in a rescue mission. Each UAV has a limited battery capacity. By incorporating real-world considerations like budget constraint, the task allocation algorithm becomes a more robust and applicable solution for optimizing task allocation in dynamic and complex environments.

1.1.4 Multi-agent Cooperative Game

We consider a fully cooperative multi-agent game as a *decentralized partially observable Markov Decision Process(DEC-POMDP)* [18]. DEC-POMDP is defined as a tuple $\langle N, S, P, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, Z, \boldsymbol{\gamma} \rangle$, where N denotes the number of agents; S is a finite state space; $P(s'|s,a) : S \times \boldsymbol{\mathcal{A}} \times S \to [0,1]$ stands for the state transition probabilities; $\boldsymbol{\mathcal{A}} = [\mathbf{A_1} \dots \mathbf{A_N}]$ is a finite set of actions, where $\mathbf{A_i}$ represents the set of local actions $\mathbf{a_i}$ that agent i can take; $\boldsymbol{\mathcal{O}} = [\mathbf{O_1} \dots \mathbf{O_N}]$ is a finite set of observations controlled by the observation function $Z : S \times \boldsymbol{\mathcal{A}} \to \boldsymbol{\mathcal{O}}$; $\boldsymbol{\mathcal{R}} : S \times \boldsymbol{\mathcal{A}} \to \mathbb{R}$ is the reward function; and $\gamma \in [0,1]$ is the discount factor.

According to the DEC-POMDP model, each agent i takes an action a_i based on its local observation o_i. When all agents applied their actions $[a_0, a_1, \dots, a_N]$ to the environment, the environment moves to a new state s' and returns a joint reward r. The MARL trains policies $\pi_i(a_i|o_i) : \boldsymbol{\mathcal{O}_i} \to \boldsymbol{\mathcal{A}_i}$, $\forall i$, that maximizes the expected discounted reward $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r^t]$, where γ is the discount factor.

Sharing observation improves the performance and helps agents learn a better action policy. Efficient communication allows agents to obtain more information about the global environment and reduces the negative impact of partial observations. We aims to answer the following three questions: What is the appropriate time to send a message by agent? How to encode a message? How to utilize the received information messages? As these factors directly influence the efficacy of collaboration among agents.

1.2 Outline and Contributions

The organization and contributions are summarize as the follows:

- In Chapter 2, we proposed a technique to search for the best prune strategy for the UAV traffic density prediction model to facilitate fast traffic prediction. To reduce the complexity of the neural network model. A neural architecture optimization framework that searches for the best compression ratio for each layer is developed. Overall, we are

able to reduce the size of the traffic prediction model by 50%. Furthermore, because the pruning adds more regularization to the model and reduces the potential of overfitting, the compressed model also achieves small improvements in the prediction accuracy.

- In Chapter 3, we aim to apply the Consensus-Based Bundle Algorithm (CBBA) to task allocation with budget constraints. Several heuristics were proposed to build the bundle and calculate the bidding scores as improvements to the original CBBA algorithm. Also, to decide which heuristic extension should be used for a given task allocation problem, a graph convolutional neural network (GCN) model is trained to extract and analyze the features of the constrained optimization problem as a graph, and predict the potential performance (i.e., global reward) of different heuristic extensions.

- In Chapter 4, we propose the Belief-Aided Message System (BAMS), which employs a neuro-symbolic belief map to augment training. The belief map decodes an agent's hidden state, providing a symbolic representation of its understanding of the environment and other agents' statuses. This symbolic representation simplifies the gathering and comparison of ground truth information, offering an additional, consistent, and reliable channel of feedback compared to sporadic and delayed rewards in RL.

- In Chapter 5, we conclude this thesis with a summarization of the results and discuss the future research directions.

Chapter 2

Neural Network Architecture Search and Model Compression for Fast Prediction of UAS Traffic Density

2.1 Introduction

New applications and services based on small Unmanned Aircraft Systems (sUAS) have gradually been introduced into urban city environments. The number of daily sUAS operations in uncontrolled low altitude airspace is expected to reach into the millions in the future. Complicated and high density UAS traffic imposes a significant burden on air traffic management, city planning and communication resource allocation. Fast and accurate UAS traffic density prediction is necessary for centralized management to coordinate and control UAS missions to avoid potential conflicts, ensure timely completion of missions and enhance operational safety through guaranteed connectivity to communication networks.

The UAS traffic density at a future time T can be considered as a function of the current UAS traffic density distribution, the flight environment (e.g. the location of the no-fly-zones), and the schedule of the future UAS missions within the target air space up to time

T. Recent research showed that conventional neural networks with at least one hidden layer satisfy the universal approximation property [19] in that they can approximate an arbitrary continuous or measurable function given enough number of neurons in the hidden layer. In our previous work [20] a DNN model has been developed to predict the traffic density. Compared to previous machine learning based traffic predictors, our DNN takes the flight environment and detailed UAS mission launch information as the inputs, hence it can be generalized to different air spaces as long as the trajectory of each UAS is routed using the same algorithm. In other words, it will need no "down time" after a change of the no-fly-zone or the launching/landing zone information. The predicted traffic has high correlation to the actual traffic.

However, the complexity of the DNN model increases when more environment dynamics are considered. The traffic predictor is used in the inner loop of many dynamic traffic management applications, such as selecting the landing and launching area, scheduling a mission request, and allocating frequency band resources for UAV communications. Each of such tasks is a combinatorial optimization problem, where the optimal solution cannot be found analytically. To efficiently explore the design space to search for the optimal configuration, the traffic predictor must be able to process many potential traffic scenarios in a short period of time. Therefore, in addition to accuracy, low complexity and low cost are other critical requirements for the traffic predictor.

The goal of this work is to compress the traffic prediction model to a smaller size without sacrificing the prediction accuracy. A network architecture search (NAS) method is developed to find the optimal DNN architecture in the spatial-temporal domain for better prediction performance and robustness. The overall framework is based on an architecture prediction model that predicts the accuracy of a compressed traffic density predictor. With the architecture search framework, we have reduced the size of the original traffic prediction model by 50% without losing accuracy.

2.2 Background and Related Works

In our previous work, a DNN model has been trained to predict the UAV traffic density with a correlation up to 0.945 and to predict traffic hot-spots with the Area Under the Receiver Operating Characteristics (AUROC) score up to 0.95. In order to consider the long, medium and short-term impact of the scheduled UAV missions, the model architecture shown in Figure 2.1, with each term contains two convolutional neural networks (CNNs) in total six was used to extract features from the future mission plans for up to 6 cycles. And each cycle represents 10 simulation cycles which is 10 seconds for real time. Another CNN is needed to process the current traffic density map. Finally, a multi-layer fully connected network is used to fuse the information from different channels to form a vector embedding and a de-convolutional neural network is used to transform the vector into a 2D traffic map. The model has 10.6MB weight parameters and requires 7.85GOP (Giga Operation) computations for each prediction.

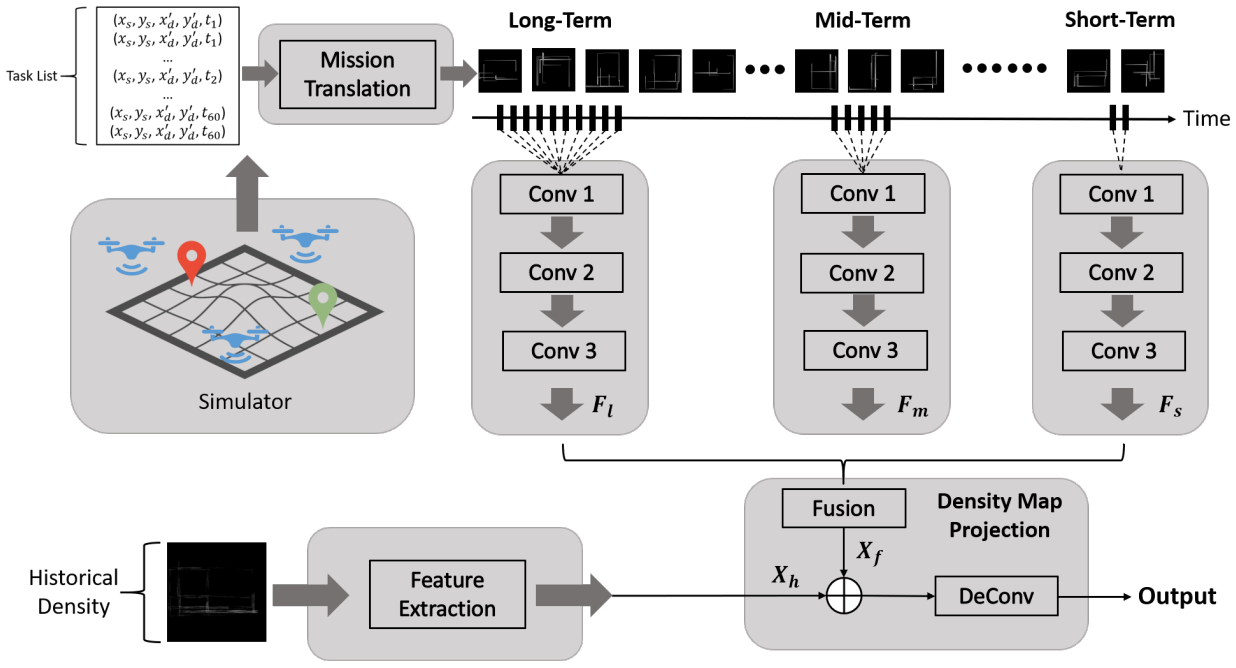


Figure 2.1: Mission-Aware Spatio-Temporal Model Architecture

Recent works have shown that weight pruning techniques [21, 14, 22] can significantly

reduce the size of DNNs without loss of accuracy. [23] proposes a Structured Sparsity Learning (SSL) method to regularize the structures (i.e., filters, channels, filter shapes, and layer depth) of DNNs, and learn a compact structure from a bigger DNN to reduce computation cost. [24, 25] proposed methods to overcome pruning ratio limitations. Other model compression methods, such as connection pruning [14, 22] and low rank approximation [26, 27], have also been proposed.

Most of the previous works aim at removing links and neurons in the given network until the prespecified prune ratio is reached. How much can be pruned without affecting the performance of the model is unknown in advance. Different prune ratios must be tried until the right one is found. As each DNN layer can be pruned with a different ratio, the possible number of combinations increases exponentially with the size of the network. While pruning a network requires iterative training and fine tuning, it obviously is not feasible to try each possible prune configuration.

Our solution in this Chapter 2, is to train an architecture prediction model that predicts the performance of the compressed model based on the combination of its layer-wise compression ratio. The architecture prediction model allows us to quickly estimate the performance (e.g., traffic prediction accuracy) of any compressed model without lengthy pruning, and consequently narrows down the search space. The predictor allows us to find a good combination of prune ratios of different layers that reduces the size of the traffic predictor while maintaining its accuracy.

Table 2.1: Architecture Information of the Original Traffic Predictor

Layer	Filter Size	Number of filters	Stride
1	4×4	64	2
2	2×2	128	1
3	2×2	256	1

2.3 Model Reduction and Architecture Search

Our traffic prediction network extracts mission information using 6 convolutional neural networks (CNNs) as shown in Figure 2.1. Each network processes the set of missions scheduled to launch in a certain time period. All of them have the structure that is given in Table 2.1. The missions scheduled to launch in the near future and far future play different roles in shaping the traffic density, therefore, although initialized with the same architecture, the weight parameters of the 6 CNNs diverge after training.

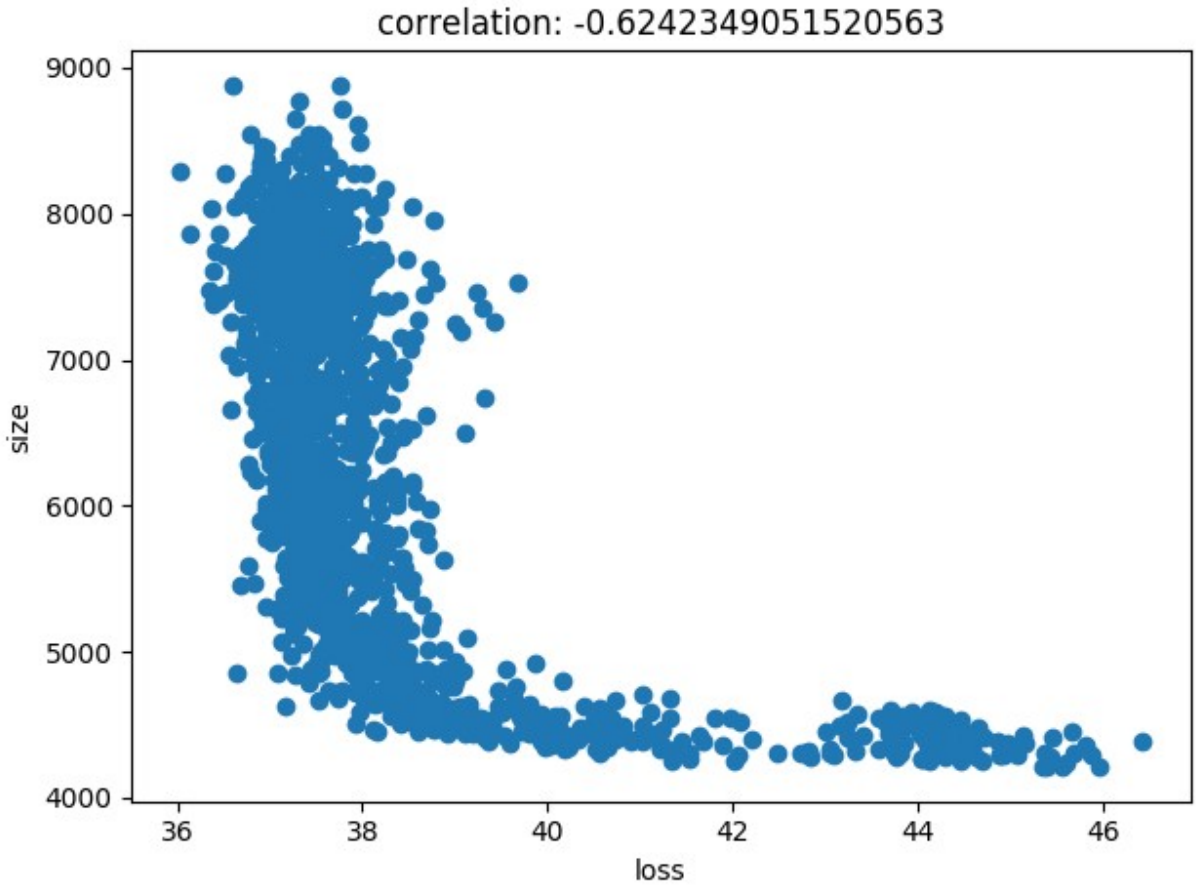


Figure 2.2: Prediction Accuracy versus Compression Ratio

Because their input has different importance to the prediction, it is natural to expect that the 6 CNNs should be compressed in different ways. Moreover, the layers in the CNNs must be pruned differently. For example, some layers play a dominant role in the prediction

process. When the number of filters of these layers is relatively low, the model's final performance will become extremely poor. The overall complexity of the compressed model is determined by the prune ratios of each of the 3 layers in those 6 CNNs. Figure 2.2 shows the relation between the prediction accuracy of the compressed model and its actual model size. Generally, a smaller size indicates a higher compression ratio. As we can see that a smaller model does not always have worse accuracy and vice versa. This is because the same compression ratio may correspond to different neural architectures, depending on how those 18 layers in Figure 2.1 are pruned. To find the best architecture is to find the optimal combination of the 18 prune ratio variables. We also need to point out that, the accuracy of the compressed model is not necessarily lower than the original model. As we will show in the experimental results, the compressed model may slightly outperform the original model, as the highly regulated structure helps it to avoid overfitting.

Exhaustively evaluating each possible compressed architecture to find its accuracy is not realistic as pruning a DNN is extremely time consuming. In this work we develop another DNN that predicts the accuracy of the compressed architecture to accelerate the design space exploration. We refer to it as the architecture predictor to distinguish from the traffic prediction model to be pruned.

2.3.1 Architecture Prediction Model

We expect that the prediction accuracy is a function of the combination of layer-wise prune ratios. A 5-layer DNN is trained to approximate this relationship. The structure of the DNN is shown in Figure 2.3, where N is the batch size. The size of each layer is labeled in the figure. The input of the model is the vector of 18 prune ratios, and the output is the accuracy of the compressed traffic predictor, measured by the mean square error of the traffic prediction.

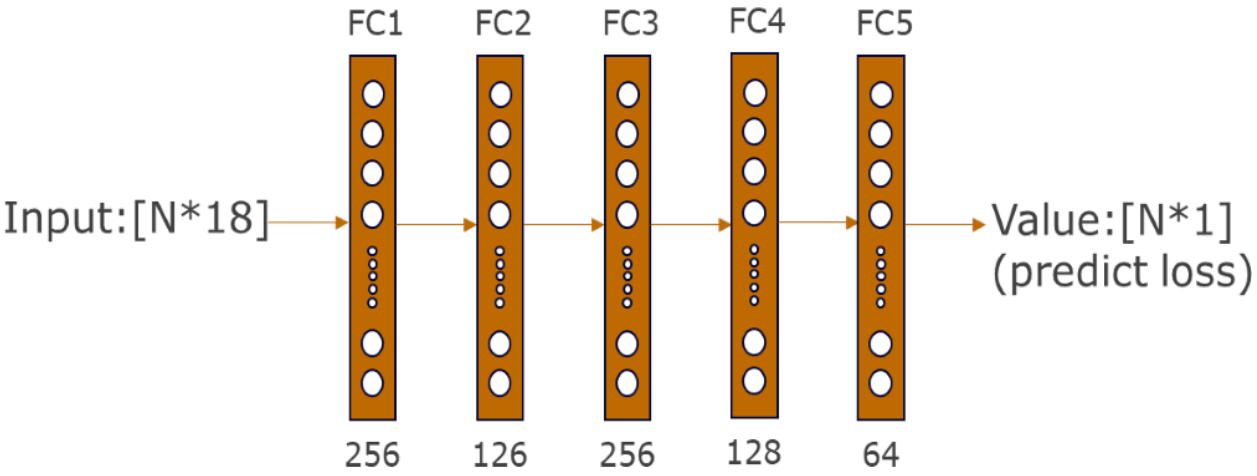


Figure 2.3: Architecture Prediction Model

2.3.2 Training of the Architecture Predictor

The architecture predictor is trained using supervised training. To generate the training data, different prune ratio combinations are sampled and the original traffic predictor is pruned/trained accordingly. The resulting compressed model is tested and its mean square prediction error is used as the target value for the training. As we can see, generating each data point for the training set requires us to train a compressed model, hence is time-consuming. However, as we will show in the experimental results, the DNN can generalize the relation between the architecture and the model performance and extend it to other prune ratio combinations that is not in the training set. While the training set is just a subset of the architecture space, the model can be used to explore the entire architecture space, which is much larger than the training set.

The architecture predictor is slightly different from a regular regression model. The goal of the conventional regression models is to maximize the prediction accuracy; hence they usually use mean square error as the loss function. The architecture predictor is used to compare different compressed architectures (that have similar size) to find the one with the best accuracy. The absolute value of those architectures' accuracy is not important, as long as their relative order is predicted correctly. In other words, it is more important to maintain a high correlation between the predicted accuracy and target accuracy than minimizing the

absolute difference between these two. Therefore, during the training, the following three techniques were adopted to improve the correlation between the predicted and the target accuracy.

2.3.2.1 Adding Correlation to the Loss Function

For each training batch, after the forward pass, the correlation between the model's predicted value $\hat{y}$ and the target value y is calculated using the following equation.

$$corr = \frac{cov(\hat{y}, y)}{var(\hat{y}) * var(y)} \tag{2.1}$$

The loss function of the training is a weighted combination of the mean square error (MSE) loss and the correlation loss as specified in Equation 2.2:

$$loss = \alpha * MSE + \beta * (1 - corr) \tag{2.2}$$

where α and β are hyper-parameters. Smaller α and larger β would cause the model to focus on increasing the overall correlation, while larger α and smaller β would cause the model to focus on reducing the overall loss.

Using the new loss function, the correlation between the prediction and the target value is improved. Although the model may lose some prediction accuracy, this is acceptable in our application.

2.3.2.2 Adjusting Training Data Distribution

We found that adding data with different features as much as possible helps to improve the quality of the model. For instance, in some extreme cases, the randomly sampled compressed architecture has convolutional layers that have only one filter left. The accuracy of such traffic prediction network is obviously very poor. Although these architectures do not have real application significance, including them in the training set can help the model better

understand the role of different convolutional layers in the training process.

2.3.2.3 Training with Mixed Batch Size

In the training process, the architecture predictor is optimized one time in one epoch and the batch size is a crucial factor. We dynamically change the batch size as the epoch increases during training. The batch size increases sequentially. For example, the first epoch's batch size is 4, the batch size of the second epoch is 10, and the batch size of the third epoch is 90.

We use partial correlation to refer to the prediction and target value correlation in the same batch and use overall correlation to refer to the correlation in the entire training set. We found that using a fixed batch size, no matter how large or small, it is hard to balance the overall and partial correlation. When the batch size is small, we get very high partial correlation. However, the model does not give consistent prediction from one batch to another, hence the overall correlation is poor. When the batch size is large, the overall correlation is improved. However, a single outlier does not affect the overall correlation significantly, and this is reflected as the poor partial correlation. Therefore, a mixed batch size is adopted. This method can integrate the advantages of large and small batch sizes and consider the partial correlation while improving the overall correlation. Since this method requires constant batch size changes, the model also needs to train more epochs to achieve the best results.

Using the above three methods, we finally increased the model correlation from 0.71 to 0.92 and even reached 0.96 in the validation set.

2.3.3 Incremental Training to Specific Corner

Since our purpose is to reduce the size of the traffic predictor as much as possible while retaining its accuracy, we are interested more in the data points located at the lower left corner of the size versus accuracy (or loss) design space shown in Figure 2.2. The architectures located in this corner have smaller size and higher accuracy, therefore they are referred

to as effective architectures. And the architectures located outside this corner are referred to as ineffective architectures as they have larger size and/or lower accuracy. During the training process, we try to specialize the model to give better prediction to the data points corresponding to the effective architectures. This is achieved by adding more data points corresponding to the effective architectures to the training set.

However, to distinguish between an effective architecture and an ineffective architecture is not easy. Given a randomly sampled prune vector, while the size of the compressed model can be easily estimated, its accuracy is unknown unless we actually compressed the mode, trained and tested it. Unfortunately, there are more ineffective architectures than effective architectures. Since an ineffective architecture does not help the training process as much as an effective architecture, much of the effort in generating the training data will be wasted. To address this problem, we apply incremental training as described in Figure 2.4. Incremental training accelerates the model's learning speed for the feature that we need by adding the validated model prediction data into the training set. First, we randomly sample 300-400 compressed architectures, train and test them to generate the initial training set. The architecture prediction model is pre-trained on the initial training set. Taking advantage of the pre-trained model, we then predict the accuracy of another set of randomly sampled architectures, and select the architecture whose accuracy is lower than a certain threshold. The corresponding compressed architecture will be generated and tested. These new data will form a new training batch to further refine the model. This procedure is repeated iteratively until the overall correlation achieves a predefined threshold. Totally 2300 training architecture data is generated in this work.

By varying the aforementioned configuration parameters, we generated four air traffic scenarios. The first scenario is the point to point route without the trajectory management. Nonetheless, other three scenarios follow the Manhattan route. Compared with the second scenario, which is the naive Manhattan style route, the third scenario integrates the trajectory management to avoid potential conflicts. The last scenario introduces the geographical

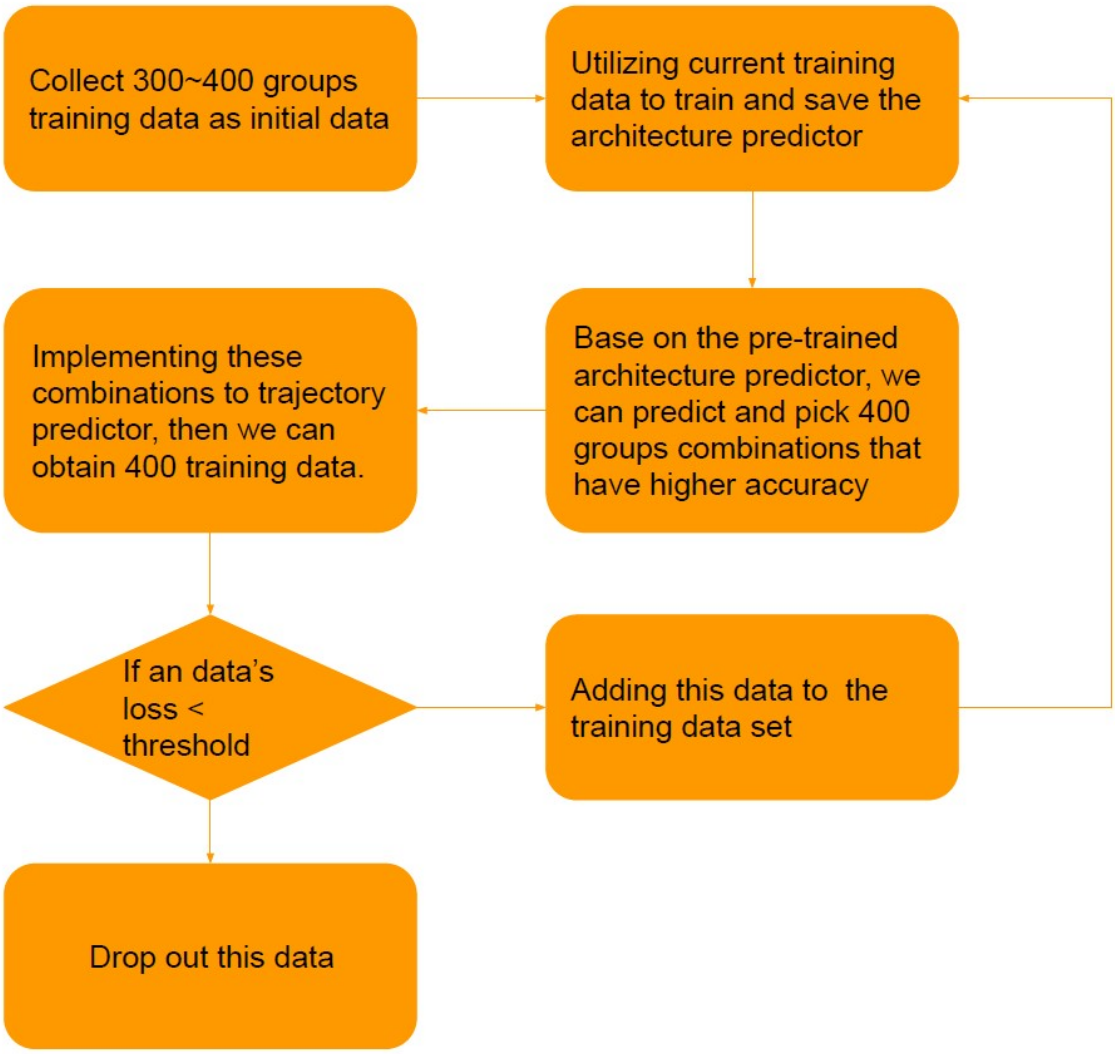


Figure 2.4: Flow of Incremental Training

constraint (the no-fly zone) in the simulation.

2.4 Experimental Results

2.4.1 Original Model vs. Compressed Model

Using the trained architecture predictor, we predicted the accuracy of 30 million random architectures with a size below a given threshold. 30 architectures with the best predicted size vs. accuracy tradeoffs were selected and implemented. The green points in Figure 2.5 show the size vs. accuracy tradeoffs of those architectures. The blue points in the figure give the size vs. accuracy of the 2300 training architectures. As we can see, since the 30 selected architecture have been filtered by the predictor, in average, they have better accuracy than the training architecture. The point located at the lower left corner is the best architecture.

Table 2.2: Original Model vs. Compressed Model

	Original Model	Compressed Model	Comparison
Model size	10,562 KB	4,713 KB	-55.38%
Loss	37.575	37.413	-0.431%
Time (200 Samples)	0.068s	0.025s	-63.24%

Table 2.2 shows the percentage reduction of size, accuracy and inference time of the best selected architecture compared to the original model. As we can see, the compressed model can slightly improve the accuracy (negative accuracy reduction) even though the size is compressed by 55%. The reason that the accuracy is slightly improved is because the compressed architecture has less weight parameters and hence is less likely to do overfitting. The inference speed is also greatly improved due to the reduced complexity. This means that our compressed model consumes fewer resources to achieve the same prediction effect.

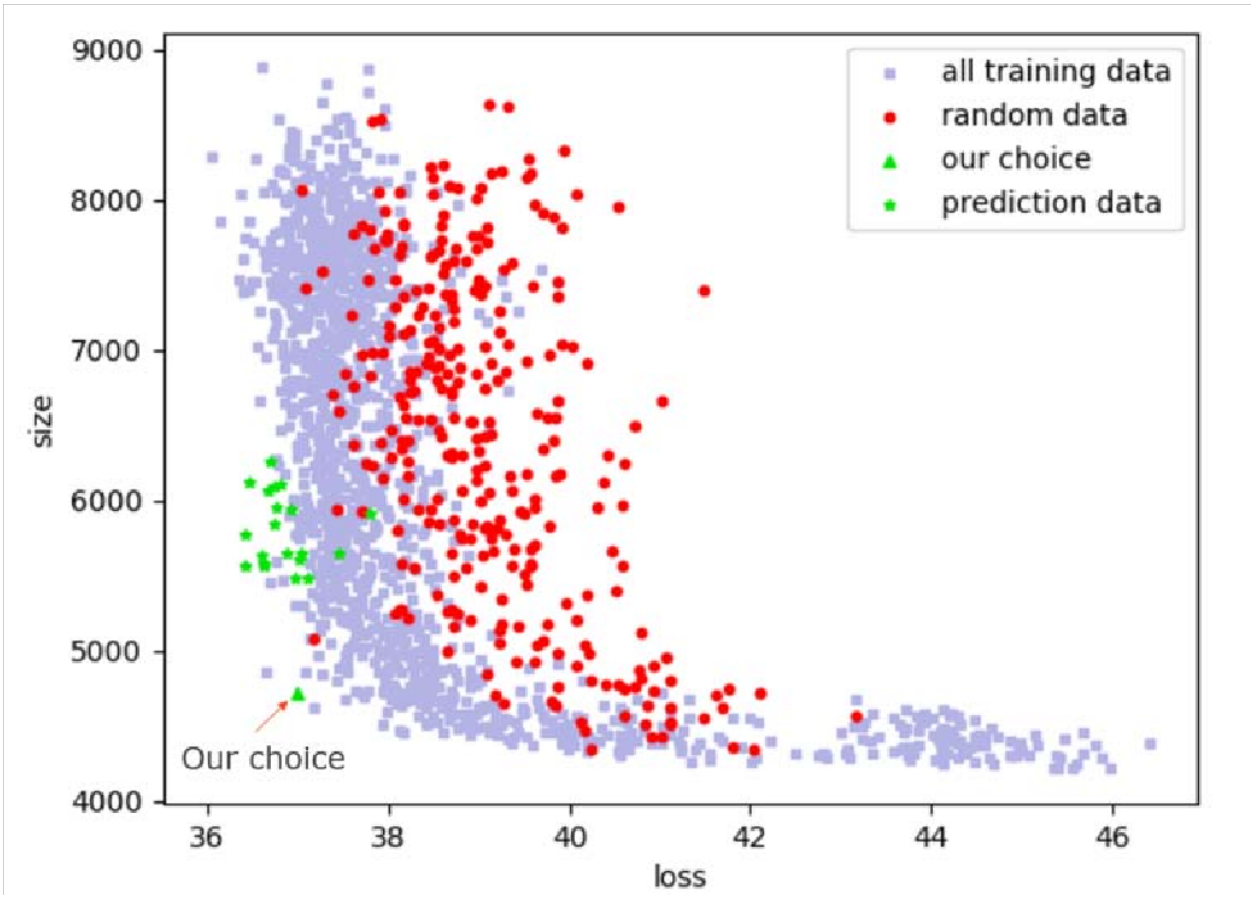


Figure 2.5: Size-Accuracy Tradeoffs of the 30 Selected Architecture

Table 2.3: Original Model vs. Compressed Model (Random Search)

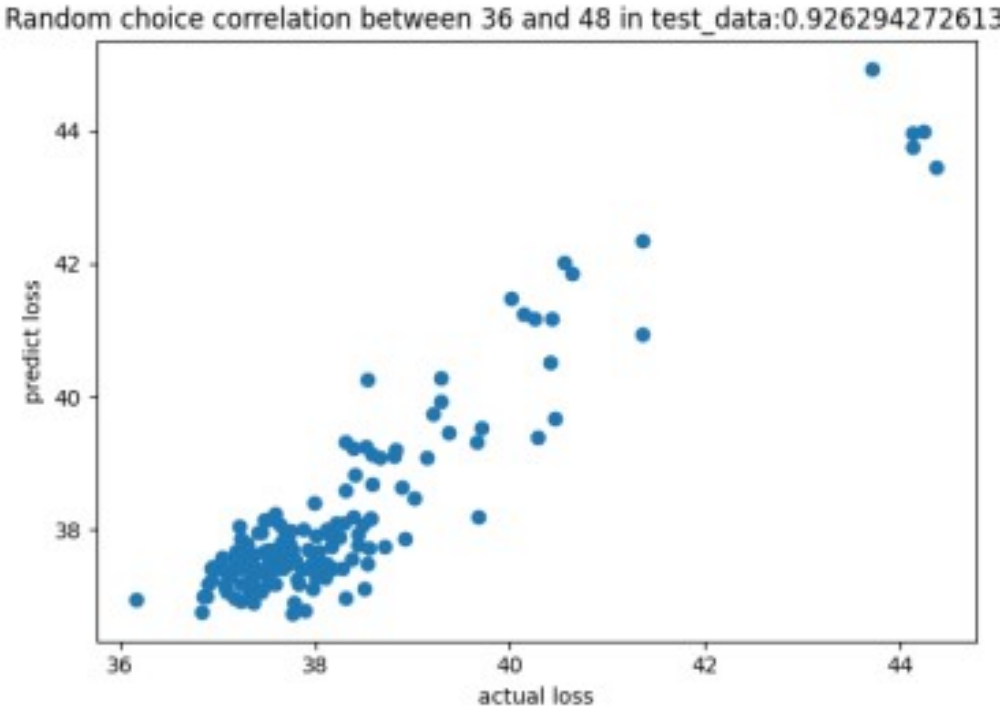
Best Random Result			
	Original Model	Compressed Model	Comparison
Model size	10,562 KB	4,490 KB	-57.49%
Loss	37.575	39.645	+5.221%
Minimum Loss Result			
	Original Model	Compressed Model	Comparison
Model size	10,562 KB	8,286 KB	-21.55%
Loss	37.575	36.038	-4.265%
Minimum Size Result			
	Original Model	Compressed Model	Comparison
Model size	10,562 KB	4,210 KB	-60.14%
Loss	37.575	45.361	+17.16%

2.4.2 Comparison with Randomly Found Architecture

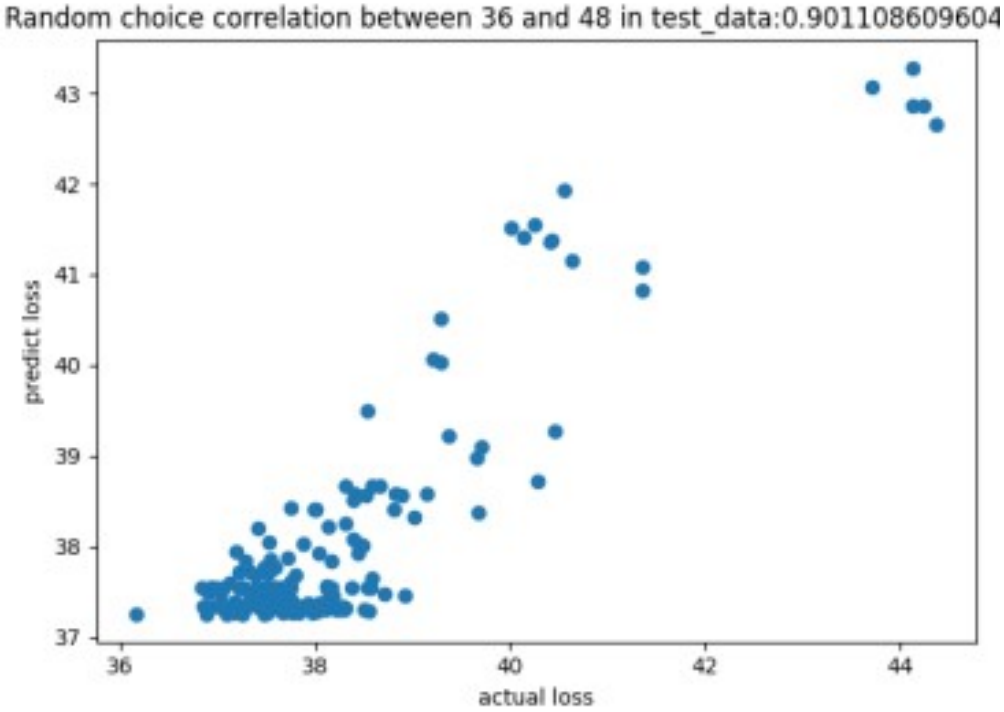
Searching for the optimal architecture by exhaustively evaluating different structures may be feasible for simple and small neural networks, since the possible architectures are limited and the training of a small neural network does not take too much time. However, when the network size gets increased, this is no longer practical. In order to find the optimal size and accuracy tradeoff, a much larger number of compressed architectures must be sampled and tested due to the exponentially increased search space, and each one requires longer training and evaluation times.

The biggest advantage of our approach is that the architecture prediction model can be trained based on a limited number of compressed architectures and it can generalize the learned architecture-accuracy relationship to a wider range. In our work, a total of 2300 compressed architectures were generated to train the model. The average mean square error (MSE) loss between the predicted value and label is only 0.14, and the correlation can reach 0.926. Using trained models, we can evaluate tens of millions of possible compressed architectures within minutes.

We randomly selected 300 architectures and plot them in the size-accuracy space in Figure 2.5 using red color. As we can see, compared to random compressed architecture, the size-accuracy plot of the training data that we generated for the incremental training process (i.e. data points in grey) has already shifted to the lower left side notably. This means these training architectures exhibit better size accuracy tradeoff. And the green data points, which are filtered by the predictor, are further improved compared to the training data. We also selected 3 of those random architectures with the best size accuracy trade off. Their reduction of its size, accuracy and inference time compared to the original model is reported in Table 2.3. It shows that at the similar compress ratio, the randomly selected architecture has much lower accuracy.



(a) W. Correlation



(b) W/O. Correlation

Figure 2.6: The Coverage Map of Good Quality and Poor Quality Links

2.4.3 Impact of Correlation Loss

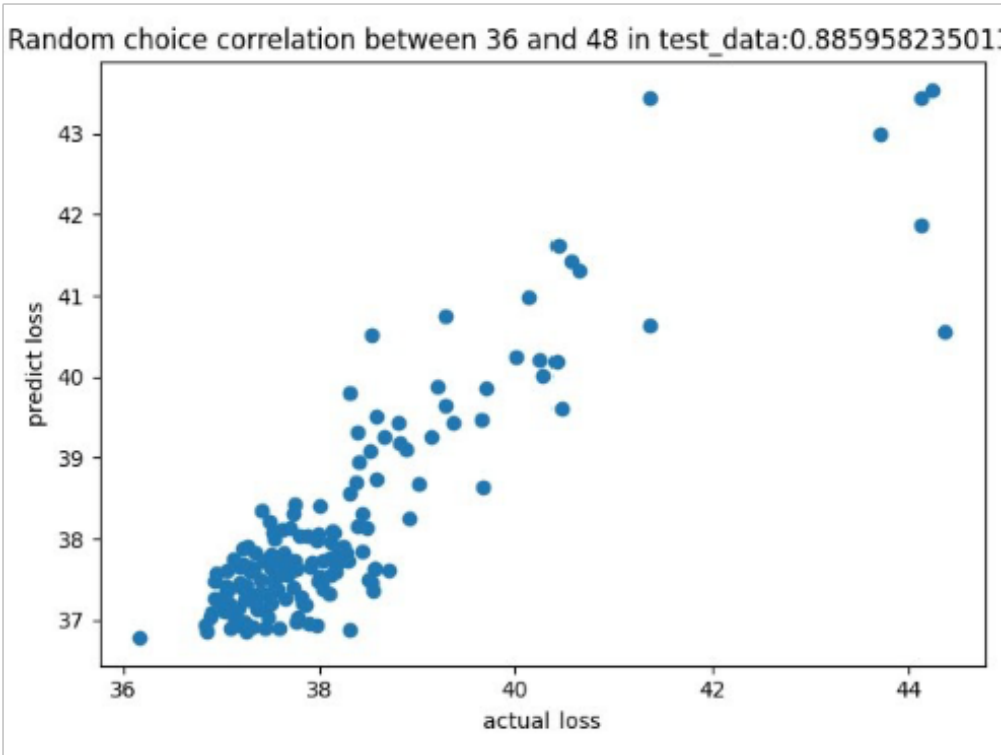
As mentioned above, we modified the loss function of the architecture predictor by adding the correlation loss. The modified loss function can directly boost the correlation between the predicted and actual value. Figure 2.6 gives the scatter plots between the prediction and the target value for the architecture predictor with and without the correlation. As we can see, if only MSE loss is considered in training, when the target accuracy (loss) is high (low), the model cannot distinguish the performance of different architectures and will simply predict the same value. This obviously will not help us select the optimal architecture.

2.4.4 Impact of Different Batch Size

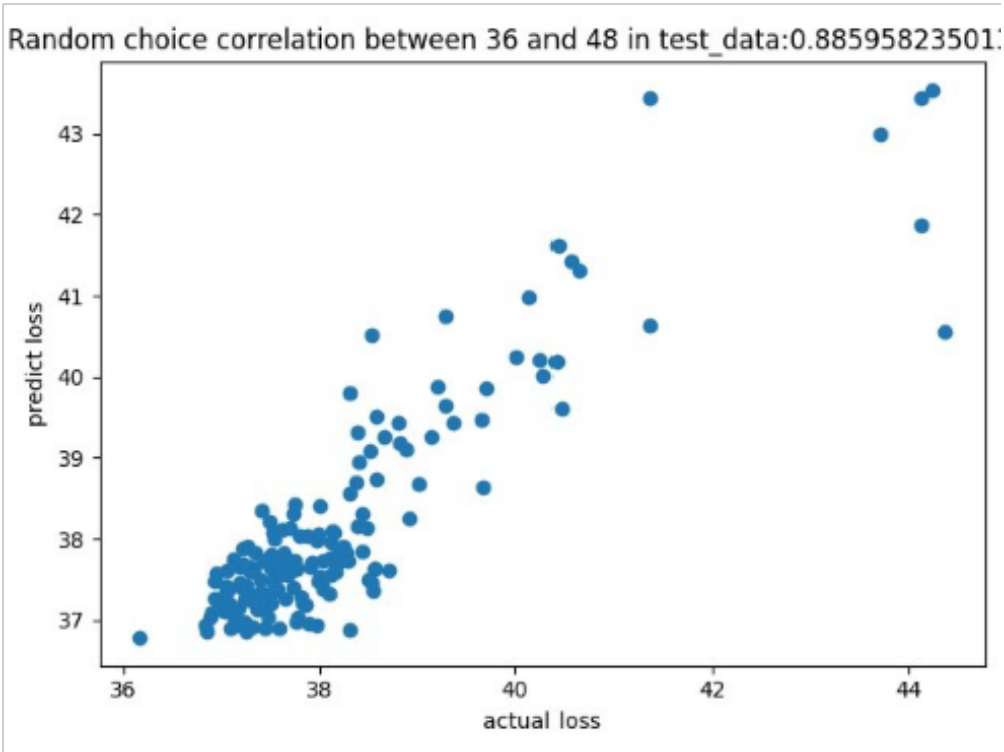
As we discussed before, the batch size also has an impact on the quality of the trained model and it helps to improve data point distribution. Figure 2.7 shows the relation between the predicted value and target value, when trained with a mixed batch and a constant batch with size $N = 90$. As we can see, the model trained with mixed batch size has better correlation. Actually, the model trained mixed batch size has correlation as high as 0.9262. And the models trained fixed batch with size $N = 10$, 50, and 90, have correlations 0.898, 0.879 and 0.885, respectively.

2.5 Conclusions

In this work, we proposed a technique to search for the best prune strategy for the UAV traffic density prediction model to facilitate fast traffic prediction. With the help of the proposed architecture prediction model, we can efficiently evaluate the performance of many different compressed architectures in the design space and select the optimal one We found out that adding correlation into the loss function, dynamically adding new data during the training phase and training with a mixed sized batch can significantly improve the correlation of our architecture prediction model. In our results, we achieved a reduction in the size of



(a) Mixed



(b) N = 90

Figure 2.7: Impact of Batch Size

the original traffic prediction model by 55% while keeping similar accuracy as the original model. At the same time, the model execution time speeds up by 60%.

Chapter 3

Applying Machine Learning in Designing Distributed Auction for Multi-agent Task Allocation with Budget Constraints

3.1 Introduction

Task allocation among multiple agents is a general combinatorial optimization problem underlying many reallife mission planning applications, such as rescue mission, delivery, or transportation scheduling, etc. Centralized and distributed solutions have been proposed based on the network condition and communication capabilities of the agents. Centralized approaches like [28, 29, 30, 31] rely on one agent to retrieve all agents' information and take the responsibility to allocate tasks and generate a proactive plan for each agent to maximize the total rewards of the entire group. They are more likely to suffer from single point failure and have higher demand on the agent's communication range.

These limitations make the decentralized approach more attractive. In the decentralized

approach, each agent selects tasks only for itself and all agents coordinate with each other to maximize overall rewards of the entire group.

Auction based techniques have been used as an alternative way for agents to communicate and coordinate with each other. By broadcasting the biding price of tasks calculated using a carefully chosen score function to its neighbors, an agent conveys the necessary information to coordinate the decision making while hide details of its own mission plan. Based on this idea, a decentralized algorithm, consensus-based bundle algorithm (CBBA) [32] was developed. Agents build their own task bundles locally, exchange bidding prices and rebuild the bundles based on the received bidding information. This bidding and conflict resolution procedure repeats until the consensus among the agents is reached and the results converge. The CBBA algorithm draws significant attention because it has provable convergence and guarantees 50% optimality when the bidding price has the diminishing marginal gain (DMG) property.

The original CBBA algorithm only considers unconstrained optimization. It assumes that each task has a reward (a) and a cost (p), and tries to maximize the score defined as the difference of the rewards and costs (i.e., $a - p$.) However, many real-life applications usually operate on a budget constraint, and their goal is to maximize the reward with the respect of the given budget. For those applications, it is not necessary to minimize the cost if the budget is sufficient. Furthermore, the costs and rewards are usually not measured using the same metric and they are not interchangeable. For example, a team of unmanned aerial vehicles (UAVs) set out to dispatch life supplies in a rescue mission. Each UAV has a limited battery capacity. The reward of the mission is the total number of people being helped while the constraint is the energy dissipation during the trip. A direct addition or subtraction of these two does not make sense. As long as the battery lasts, the goal should be to help as many people as possible. A typical approach to solve such constrained optimization problem is to use Lagrange function to combine the reward and cost. This introduces two problems. First of all, its effectiveness highly depends on the selection of the Lagrange multiplier.

Different Lagrange multiplier must be used for different applications, however its selection is largely empirical. Secondly, the combined score may not be DMG. Therefore, convergence is not guaranteed. In this work, we present a set of heuristic extensions of CBBA based on 7 carefully designed new score functions. Their performance is compared and the DMG property of some of the heuristics is analyzed and proved. The results show that, in average, those heuristic extensions lead to 16% more rewards than the original CBBA algorithm.

Our experimental results show that those heuristic extensions are pareto efficient. This means no extension is dominated by another and each heuristic extension outperforms all other extensions in at least one test case. Using different heuristic extensions for different allocation problems will be better than consistently using any single one of them. However, this brings up a question, how do we know which heuristic works the best for a given task allocation problem? In this work we attempt to answer this question by training a graph convolutional neural network (GCN) to predict the performance of the heuristic extensions and use the predicted performance to guide the selection. Our experimental results show that, based on the prediction, we can select the best heuristic extensions 69% of the time. This stands for a 38% improvement compared to random selection.

3.2 Background and Related Works

3.2.1 Consensus-Based Auction Algorithm (CBBA)

The CBBA algorithm consists of 2 phases [33, 34], bundle construction and conflict resolution. Each agent i carries four vectors of local information: a winning bid list y_i, a winning agent list z_i, a bundle b_i, and a path p_i. Winning bid list y_i records the winner agent id for each task. Winning agent list z_i records the winning bids that the winner agents put for the task. Bundle b_i is the sequence of tasks that agent i selected. Path p_i stored the true path that agent should travel to gain the rewards of the tasks in bundle. In bundle construction phase, each agent will continuously add available tasks from task set Γ into its bundle until

it reaches the limitation of bundle capacity. Marginal score function of task j for agent i is defined as

$$c_{ij}[b_i] = \begin{cases} 0, & \text{if } j \in b_i \\ \max_{n \leq |p_i|} S_i^{p_i \oplus_n \{j\}} - S_j^{p_i}, & \text{otherwise} \end{cases} \tag{3.1}$$

where $S_j^{p_i}$ is the total reward minus the cost of p_i, "$\oplus_l$" denotes the operation that inserts the second list right after the lth element of the first list.

In each iteration, the task which has the maximum marginal score c_{ij} will be added into the bundle. The agent will continue selecting new tasks until it reaches the bundle capacity L_t. The new task will be added to the end of bundle and inserted into the position in the path that gives the maximum marginal score c_{ij}. Algorithm 1 summarizes the process.

Algorithm 1 CBBA Phase 1 for agent i at iteration t:

Build Bundle $(z_i(t-1), y_i(t-1), b_i(t-1)$
$y_i(t) = y_i(t-1)$
$z_i(t) = z_i(t-1)$
$b_i(t) = b_i(t-1)$
$p_i(t) = p_i(t-1)$
while $|b_i| < L_t$ **do**
  $c_{ij} = max_{n \leq |p_i|} S_i^{p_i \oplus_n \{j\}} - S_j^{p_i}, \forall j \in \Gamma \backslash b_i$
  $h_{ij} = I(c_{ij} > y_{ij}), \forall j \in \Gamma$
  $J_i = argmax_j c_{ij} \cdot h_{ij}$
  $n_{i,J_i} = argmax_n S_i^{p_i} \bigoplus_n \{J_i\}$
  $b_i = b_i \oplus_{end} \{J_i\}, P_i = P_i \oplus_{n,J_i} \{J_i\}$
  $y_{i,J_i}(t) = c_{i,J_i}, z_{i,J_i}(t) = i$

In the second (i.e., conflict resolution) phase, all agents exchange their recorded bidding and winner information of the tasks. Only the agent with the highest marginal score could be the winner for the task. For those agents who outbid for the task, they should release this task and all other tasks added in the bundle after the outbid task. Meanwhile, based on their local recorded bidding price and the received bidding price, the agent will either update, reset the bidding/winner list or keep it in the same way.

CBBA process with a synchronized conflict resolution phase over a static communication

network will converge if the scoring function meet the condition of Diminishing Marginal Gain (DMG). DMG requires each task's marginal score $c_{ij}[b_i]$ to remain the same or decrease when other task(s) b is added to the bundle:

$$c_{ij}[b_i] \geq c_{ij}[b_i \oplus_{end} b] \tag{3.2}$$

CBBA has provable convergence and 50% optimality with DMG scoring function for the multi-agent task-allocation problem. When DMG is not available, warping is needed for the CBBA to converge. The score $c_{ij}[b_i]$ will be warped to $minc_{ij}[b], \forall b \in b_i$, to help convergence.

3.2.2 Graph Convolutional Network (GCN)

In the past decade, the deep learning technique has been applied widely for its impressive feature extraction and representation capabilities. As one of the most successful deep neural network models, convolutional neural network [35] is extremely effective in analyzing data with an underlying Euclidean or grid-like structure. However, the conventional CNN is not applicable to the non-Euclidean structured data, such as social networks or information networks, due to the lack of translation invariance. A graph convolutional network (GCN) is a method for processing the data represented in graph domain [36], especially for the Non-Euclidean structured data. It was first proposed for fake news detection in [37, 38]. When combined with the CNNs and graph embedding techniques [39], the GCNs can be used to extract representative features from graph structure.

Given a graph $G = (V, E)$, where V and E represent the set of objects (i.e., the vertices), and their relationship (i.e., the edges), the graph convolution can be carried out in two domains, the spatial domain [40] and spectral domain [38].

The spatial domain [41] approach performs graph convolution by summing up a node's neighborhood information directly. It also applies residual connections and skip connections to memorize information over each layer. Under this model, each vertex is associated with

a neural network. The activation of the kth layer of node $v(h_v^{(k)})$ is calculated as:

$$h_v^{(k)} = f(W^{(k)}x_v + \sum_{i=1}^{k-1} \sum_{u \in N_{(v)}} \theta^{(k)} h_u^{(k-1)}) \tag{3.3}$$

where $W^{(k)}$ and $\theta^{(k)}$ are learnable weight matrices for local and inter-nodes connections. D denotes the activation function.

The spectral domain graph convolution [38] can be defined as a signal $x \in \mathbb{R}^N$ (a scalar for every node) convolves with a filter $g_{\theta'}$ ($g_{\theta'} = diag(\theta)$ parameterized by $\theta \in \mathbb{R}^N$) as the following:

$$g_{\theta'} * x = U g_\theta U^T x \tag{3.4}$$

where U is the matrix of eigenvector of the normalized graph Laplacian $\mathcal{L} = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$, with a diagonal matrix of its eigenvalues Λ. And $U^T x$ is the graph Fourier transform of x. Using the first order approximation of the spectral convolution, the layer-wise propagation rule of GCN can be defined as the following:

$$H^{(l+1)} = f(H^{(l),A}) = \sigma(\check{D}^{-\frac{1}{2}} \tilde{A} \check{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3.5}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix A of graph (V, E) with additional self-loop edges I_N, $\check{D}_{ii} = \sum_j A_{ij}$, and $W^{(l)}$ is a trainable weight matrix. $\sigma()$ denotes the activation function. The spectral based GCN is used in this work.

It is a common practice to model a multi-agent system as a graph. In [42, 43] a graph is used to capture tasks' location and connectivity. In [44, 45] the agent-entity graph and relevance graph are used to represent the relationship among agents and relationship between agents and environment. However, none of these works applied machine learning techniques to analyze those graphs for prediction or classification. In this work, we represent the task location and their distance using a graph, and apply the GCN to extract features that help us to predict the best auction strategy.

3.3 Problem Definition

Without loss of generality, we discuss multi-agent task allocation in the context of UAV-based rescue mission, where UAVs and rescue sites correspond to agents and tasks, respectively. Let N and M represent the number of agents and tasks respectively, and a_i and t_j denote the ith agent and jth task. Associated with each agent and task pair (a_i, t_j), there is a reward r_{ij}. Every agent has a departure site and all agents must return to a common returning site. The rescue sites, and the departure/returning sites together can be considered as a set of vertices $V = \{v_j, 1 \leq j \leq M + N + 1\}$ and the flight routes directly connecting two sites form a set of edges $E = \{e_{j,k}, 1 \leq j, k \leq M + N + 1\}$. The graph (V, E) has an adjacency matrix $\mathbf{A}$, A_{jk} represents the energy needed for UAV to fly directly from v_j to v_k. The first M rows/columns of $\mathbf{A}$ correspond to M tasks, the next N rows/columns correspond to the departure site of the N agents, and the last row/column correspond to the common returning site. Similar to the distance, the energy dissipation is additive. The energy required for a UAV to travel a path is the sum of the energy needed on each edge alone the path. In the rest of the chapter, we simply refer to A_{jk} as the "distance" from v_j to v_k, as if "energy dissipation" is a unit to measure the "distance".

To account for real-life scenarios such as adversarial weather or no flight zone, the distance (i.e., energy) A_{jk} may not be proportional to the Euclidean distance between v_j and v_k. Furthermore, the graph (V, E) does not have to be a complete graph and the adjacency matrix $\mathbf{A}$ does not have to be symmetric either. It is possible that the shortest distance (i.e., least energy) route between two vertices is not the direct connection (e.g., due to the wind speed or other impact). The UAV should not stay in the departure site or any of the rescue site, therefore $A_{ii} = \infty, 1 \leq i \leq M + N$. Once the UAV arrives the returning site, it will stay there indefinitely, hence, $A_{|V||V|} = 0$ and $A_{|V|i} = \infty, 1 \leq i \leq M + N$. Based on $\mathbf{A}$, it is not hard to derive the shortest distance between any two nodes in the graph, and we denote this information using matrix $\mathcal{A}$.

Each agent has a budget $B_i, 1 \leq i \leq N$, which can be interpreted as the maximum

available battery capacity. To make the problem more challenging, we assume that each rescue site has enough power/resource to interact with only one UAV. In other words, each task can only be processed by one agent and its rewards can only be picked once. Hence the coordination among agents becomes more critical. This also means that an agent may pass (without collecting the reward) a task not in its bundle just to reach some tasks in the bundle, if this gives a more energy efficient route.

The multi-agent task allocation problem defined above is a combinatorial optimization, which can be formulated and solved using integer linear programing. To be consistent, we assign an empty reward to the vertices corresponding to the departure and returning site, i.e., $r_{ij} = 0, \forall i, M < j \leq M + N + 1$. We introduce a set of binary variables, $x_{i,n,m,k}, 1 \leq i \leq N, 1 \leq n, m \leq M, 1 \leq k \leq M + 1$. $x_{i,n,m,k} = 1$ if agent i select t_n and t_m as the $(k-1)th$ and kth tasks respectively in its bundle, otherwise it is 0. Then the multi-agent task allocation can be formulated as the following integer linear program:

$$Max : \sum_{1 \leq i \leq N, 1 \leq n,m,k \leq M} r_{i,m} * x_{i,n,m,k} \tag{3.6}$$

With respect that: $x_{i,n,m,k} \in 0, 1$

Subject to:

$$\sum\nolimits_{1 \leq i \leq N, 1 \leq n,k \leq M} x_{i,n,m,k} \leq 1, \forall m, 1 \leq m \leq M \tag{3.7}$$

$$\sum\nolimits_{1 \leq m \leq |V|} x_{i,(M+i),m,1} = 1, \forall i, 1 \leq i \leq N \tag{3.8}$$

$$\sum\nolimits_{1 \leq n \leq |V|} x_{i,n,|V|,M+1} = 1, \forall i, 1 \leq i \leq N \tag{3.9}$$

$$\sum_{1\leq j\leq |V|} x_{i,j,m,k} = \sum_{1\leq j\leq |V|} x_{i,m,j,(k+1)}, \forall i, 1 \leq i \leq N, \forall k, m, 1 \leq k, m \leq M \tag{3.10}$$

$$\sum_{1\leq n,m\leq |V|, 1\leq k\leq M+1} \mathcal{A}_{n,m} x_{i,n,m,k} < B_i, \forall i, 1 \leq i \leq N \tag{3.11}$$

Equation (3.6) specifies that the objective is to maximize the total rewards of all agents over the entire process. Equation (3.7) specifies the constraint that each task can be served at most once. Equations (3.8) and (3.9) specify the constraints that the agents must depart from their corresponding departure sites and come back to the common returning site. Equation (3.10) gives the constraint that agents will not disappear or appear in the middle of the process. Equation (3.11) specifies the budget constraint.

The constrained optimization problem described by Equations (3.6) – (3.11) can be solved by many ILP solvers. However, when the size of the problem increases, the time complexity will soon become prohibitive.

3.4 Constrained Task Allocation Using Extended CBBA

3.4.1 Original CBBA in Constrained Optimization

As discussed in Section 3.2, the CBBA algorithm could handle the general task allocation efficiently in a decentralized way. When the marginal gain is DMG, the algorithm is provable to converge and achieve 50% optimality. However, the algorithm is not designed for task allocation with budget constraints.

Similar to the original CBBA, we use $p[b_i]$ to denote the path that connects all tasks in bundle b_i. (Note that the sequence of tasks along path $p[b_i]$ is not the same as the order of tasks in the bundle b_i.) We use function $D(p[b_i])$ to denote the distance of path $p[b_i]$. To apply CBBA to our problem, each agent will initialize their bundle to include the departure

site and returning site: $b_i = \{v_{M+i}, v_{|V|}\}$. Given current bundle b_i of agent i, using the original CBBA, the marginal score $c_{ij}[b_i]$ of adding a task j to the bundle is its reward minus the marginal cost,

$$c_{ij}[b_i] = \begin{cases} \mathrm{r}_{ij} - \delta_j[b_i], & \text{if } \delta_j[b_i] \leq B_i[b_i] \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

where $B_i[b_i]$ is the remaining budget of agent i after traveling path $p[b_i]$; $\theta_j[b_i]$ is the extra distance that the agent has to travel if task j is inserted into $p[b_i]$, and is referred to as delta distance. It is calculated as $\theta_j[b_i] = min_{l<|p[b_i]|} D(p[b_i] \oplus_l \{j\}) - D(p[b_i])$.

All other steps in the CBBA remain the same except that the condition in Algorithm 1 line 6 will be changed to "while $B_i[b_i] \geq 0$", so that it keeps on adding tasks to the bundle when the budget is not depleted.

Unfortunately, the revised marginal score in Equation (3.12) is not DMG because $\theta_j[b_i]$ may decrease as bundle b_i grows. For example, we will have $\theta_j[b_i \oplus_{end} \{x\}] < \theta_j[b_i]$ if vertex x locates very close to v_j. To use CBBA with non-DMG score function, warping is needed in order to guarantee convergence. As we will show in the experimental results, warping significantly degrades the performance. Another issue with the score function in Equation (3.12) is that it directly operates on the cost and reward. However, these two variables are measured using different metrics. Such combination is not scale invariant. When the map scales linearly, the relative value of the bidding score of different agents will also change, which results in different allocation schemes.

3.4.2 Extended CBBA with New Score Functions

We further propose several heuristic extensions of the CBBA by introducing 6 new marginal score functions. When $\theta_j[b_i] \leq B_i[b_i]$, they define the marginal score as the following:

- Heuristic 1 (H1): $c_{ij}[b_i] = r_{ij}$

- Heuristic 2 (H2): $c_{ij}[b_i] = r_{ij} - (\delta_j[b_i] - avg_{k \notin b_i}(\delta_k[b_i]))$
- Heuristic 3 (H3): $c_{ij}[b_i] = r_{ij}/\delta_j[b_i]$
- Heuristic 4 (H4): $c_{ij}[b_i] = r_{ij} - \delta_j[b_i]/B_i[b_i]$
- Heuristic 5 (H5): $c_{ij}[b_i] = r_{ij} - \frac{\delta_j[b_i] - avg_{k \notin b_i}(\delta_k[b_i])}{B_i[b_i]}$
- Heuristic 6 (H6): $c_{ij}[b_i] = r_{ij} + (\frac{B_i[b_i]}{B_{max}} - \frac{\delta_j[b_i]}{B_{max}})$

When $\delta_j[b_i] > B_i[b_i]$, for all heuristics, $c_{ij} = 0$. We denote the original CBBA score function in Equation (3.12) as H0 to simplify the discussion.

The goal of constrained task allocation is to maximize the reward while the budget is still available. Hence, Heuristic 1 considers only the reward as the score when the budget is available. Heuristic 2 is an improvement of the original CBBA. It is based on the rationale that, as long as there is budget, the agent must serve a task. Hence the marginal cost for task j is not simply $\delta_j[b_i]$ but the extra distance of including task j compared to including any other tasks in the trajectory. Heuristic 3 calculates the marginal score as the ratio of r_{ij} and delta distance. It is another way to prioritize higher reward and lower cost.

The amount of remaining budget decides how critical it is to consider the cost. It is not necessary to select tasks with lower cost if the remaining budget is abundant, and vice versa. Heuristic 4 consider the cost as the ratio between delta distance and the remaining budget. It is scale invariant because the ratio $\delta_j[b_i])/(B_i[b_i]$ is always less than 1, which is less than the minimum unit reward. In other words, this heuristic score function will put reward as the highest priority consideration, when there is a tie, the cost will then be considered. Similar as Heuristic 2 to the original CBBA, Heuristic 5 is an improvement to Heuristic 4, which considers cost as additional distance increase of inserting j compared to inserting any other tasks. Heuristic 6 follows the same idea as Heuristics 4 and 5. However, the difference between the remaining budget and delta distance of j is considered instead of their ratio. The constant B_{max} is a large number to ensure that what inside the parenthesis is less than 1.

Theorem 1: Both heuristics 1 and 6 are DMG, i.e., $c_{ij}[\mathbf{b_i}] \geq c_{ij}[\mathbf{b_i} \oplus_{end} \{x\}]$.

Unfortunately, all other heuristics are not DMG. Warping must be used to ensure the convergence.

We further modify those heuristics such that, for a task j in bundle b_i, the bidding score is r_{ij}. This is achieved by changing the Algorithm 1 line 13 to "$y_{i,J_i}(t) = r_{iJ_i}$". The heuristic score c_{ij} is referred to as the selection score as it is only used for the agent to prioritize tasks and add them to its bundle. When the agent forms the bundle, cost of tasks and remaining budget are considered. However, during the bidding process, the winner will always be the agent with the highest reward. We denote such extension as H[N]', $0 \leq N \leq 6$, and refer to them as the reward-only extensions. The H[N]' are DMG, as the biding score of a task is either 0 or constant. We need to point out that as the selection score and the biding score are different, the relation between the DMG and convergence proved in the original CBBA paper no longer apply here. In other words, the DMG property of the bidding score does not necessarily lead to convergence. However, in our experiments, we do notice that for 99% of the test cases, the H[N]' extension of the CBBA algorithm converges. This is not observed for their H[N] counterparts. The simulation results show that, in average, systems using H[N]' heuristics receives 12% more rewards than systems using H[N] heuristics. All of the heuristic extensions outperform the H0, the original CBBA.

3.4.3 Performance Predictor using GCN

The detailed comparison of the aforementioned heuristic extensions of CBBA can be found in the experimental results section. Although some of them have better performance than others, in general, those heuristics are pareto efficient. None of them completely dominate other heuristics. Even the best one of them, H4, receives the highest rewards only for 46% of the test cases. Every one of them will outperform others or be outperformed by others for some specific testcases. Naturally we want to pick the best heuristic extension to carry out the distributed auction for different task allocation problems. However, to predict which

heuristic will outperform others for a given test scenario is not trivial. It depends on the budget of agents, the reward of tasks, and the location and connectivity of tasks. All of these should be considered jointly.

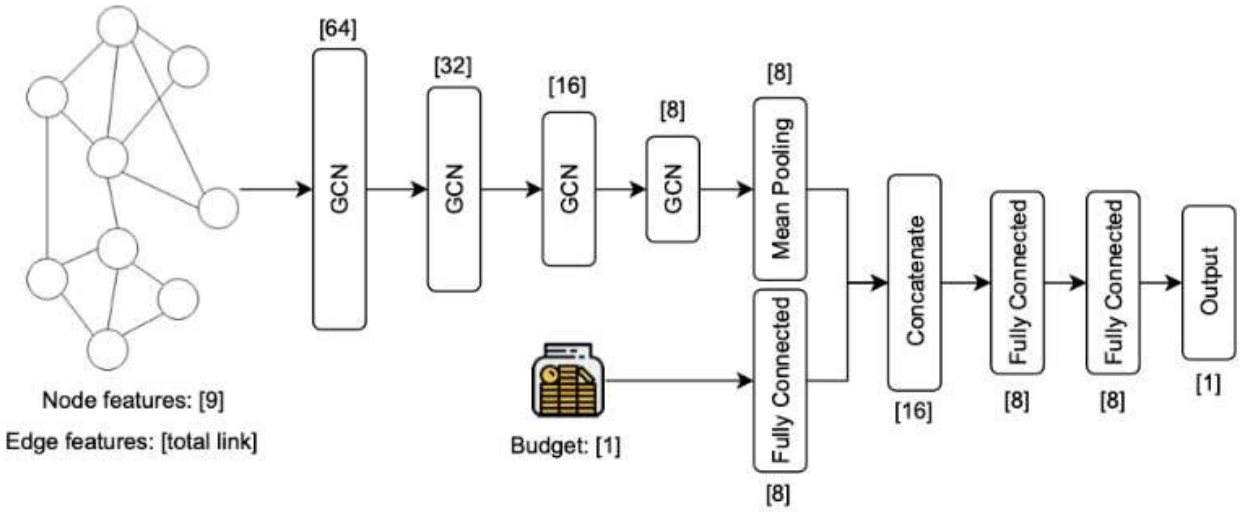


Figure 3.1: Network Structure of the GCN for Performance Prediction

In this work we propose to use neural network to train a machine learning model to predict the performance of different heuristic extensions of the CBBA. The prediction will be used to guide the selection. The neural network used here must be able to extract unique features of the environment, such as the connectivity and distance among tasks and departure/returning sites, and fuse this with UAV related features, such as available budget and rewards. The traditional CNN model does not apply here as the tasks and their connections are not Euclidean. Instead, a GCN model is used here.

The input of the GCN model includes three main features, tasks, edges and budgets. Each task j is represented by a $(2N + 3)$-dimensional feature vector. It includes the task location (x, y), rewards to each agent $r_{ij} \in R, 1 \leq i \leq N$, the distance between the task and departure site $(A_{(M+i),j}, 1 \leq i \leq N)$ or returning site $(A_{j,|V|})$. And the edge feature consists of the connectivity and distance information.

The network architecture of our GCN is shown in Figure 3.1, we use a four-part Graph CNN model with four convolutional layers with 64-, 32-, 16-, 8-dimensional output features

map in each layer to extract the environment information. And then a mean pooling layer is used to sum the learned node representation to create a graph representation. The budget feature is extracted using a fully connected (FC) layer and then fused with the graph representation using three FC layers. The feature sizes of those FC layers are labeled on the figure. The output of the model is a prediction of the global rewards that the agents will receiving by applying the corresponding heuristics extensions. Mean square error loss is used in the training.

3.5 Experimental Results

Extensive simulations have been carried out to evaluate the performance of the 7 extended CBBA algorithms, H0 H6. As we discussed before, all of them uses warping to ensure convergence except H0 and H6, which are DMG. The reward-only counterparts of those heuristics, H0' H6', are also evaluated. Please note that the reward-only extensions use $c_{ij}[b_i]$ as the selection score to build the bundle and use r_{ij} as the bidding score in the bidding process. For H1 heuristic, $c_{ij} = r_{ij}$, therefore, H1 = H1'.

In all the experiments, 20 tasks and 3 agents were used. The tasks are located in a 30×30 2D plane. The distance (energy) between two tasks is set as $A_{jk} = d_{jk} + \theta_{jk}$, where d_{jk} is the Euclidean distance between tasks j and k, and θ_{jk} is either a random variable in the range of [3, 6] or 0. The reward r_{ij} is a random integer variable in the range of [1-5]. We sweep the budget from 40 to 120. Overall, more than 10,000 test cases were generated and simulated. For the dataset, we split those 12,000 cases into train and test sets by 4:1.

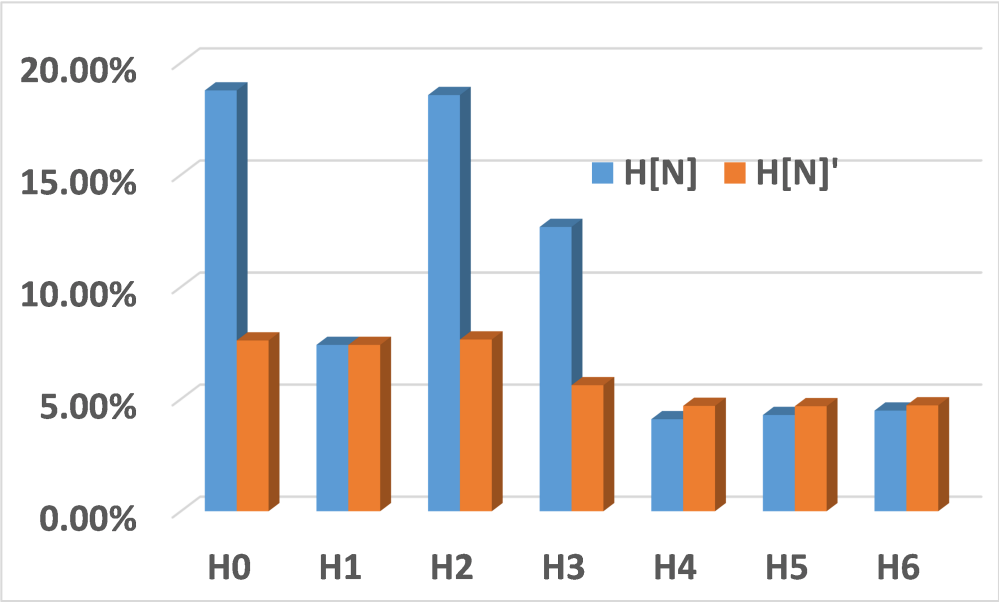
3.5.1 Performance of Heuristic Extensions of the CBBA

In the first experiment, we compare the performance of different heuristics and their non-warping extensions. The CBBA bidding process is simulated and the total rewards received by all agents were recorded. A heuristic covers a test case if it receives the highest rewards.

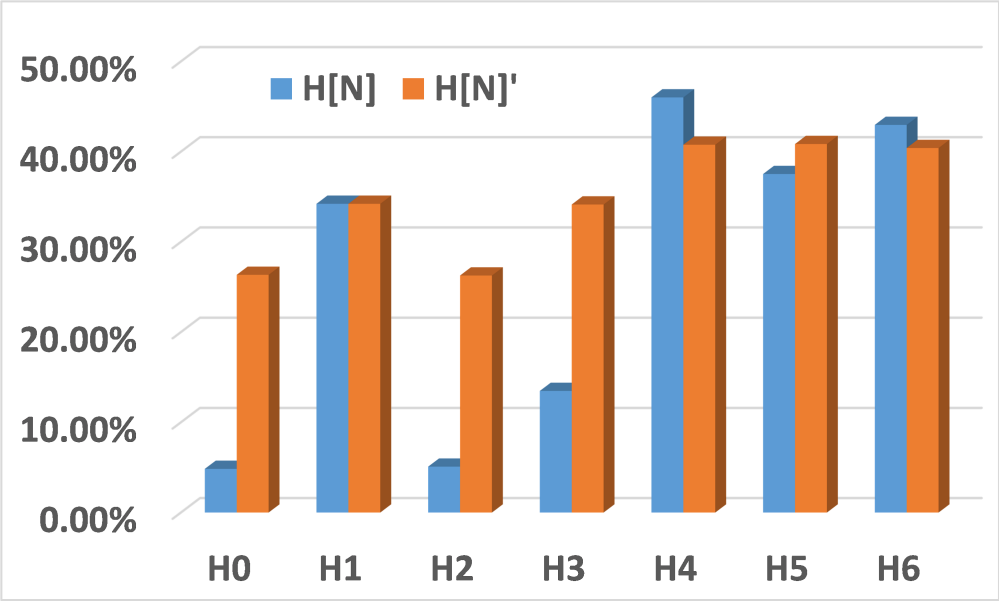
In this case, we also say that this heuristic dominates other heuristics. It is possible for more than one heuristic to cover the same test case if all of them receive the same number of rewards that is higher than other heuristics.

Figure 3.2a shows the the percentage of times a heuristic extension dominates. The higher number indicates better performance. Figure 3.2b shows the average percentage reward reduction of a heuristic compared to the best heuristic. The smaller number indicates the better performance. As we can see, the original CBBA, H0, that naively combines the reward the cost does not perform well. The warping has significant impact on the performance. The reward-only extension significantly improves the reward except for H1, H4 and H6. The average improvement is about 12%. As we mentioned before, H1 and H1' are the same, therefore, there is no difference in their performance. The H6 is DMG, hence, does not trigger the warping function. Therefore, its reward-only counterpart does not provide any improvement. On the contrary, by ignoring the remaining budget and distance in bidding process, it actually degrades the performance. The H4 is a special case. Although it is not DMG, we can prove that $c_{ij}[b'_i] - c_{ij}[b_i] < 1$, for all $b_i \subset b'_i$, and 1 is the minimum unit of the reward. Hence, even if the warping is triggered, the $c_{ij}[b_i]$ is only clamped marginally and so the impact of warping is negligible. The experimental results show that significant warping reduces the performance of the bidding process, because it deprives other agents from receiving the correct reward/cost information.

The experimental results also show that all heuristics have a positive percentage of coverage. Even the worst heuristic H0 covers about 5% of the test cases. On the other hand, none of the heuristics covers 100%. Even the best of them, H8, covers less than 50% of the test cases. Therefore, being able to select the right heuristic (or the reward-only extension) is crucial before the agents start the CBBA process.



(a) Percentage of coverage



(b) Percentage reward reduction

Figure 3.2: Comparison of the Heuristic Extensions of CBBA

3.5.2 Machine Learning Based Heuristic Prediction

Given a task allocation problem, which heuristic extension performs the best depends on the location of tasks, departure and returning sites, their distance, the available budget of each agent and the reward of each agent-task pair. GCN models are trained to analyze these features and predict the performance of different heuristics. The prediction will be used to guide the selection. The test cases are divided into training and testing data based on a 4:1 ratio.

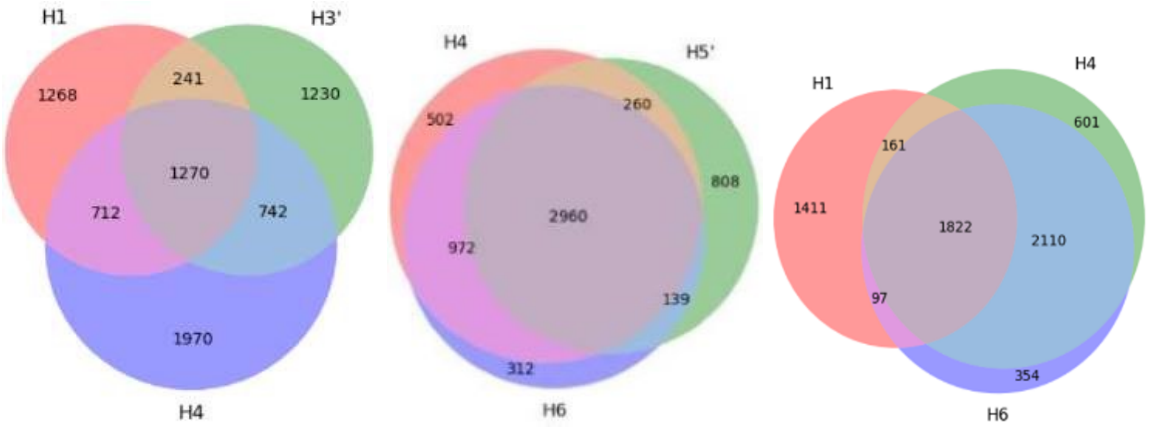
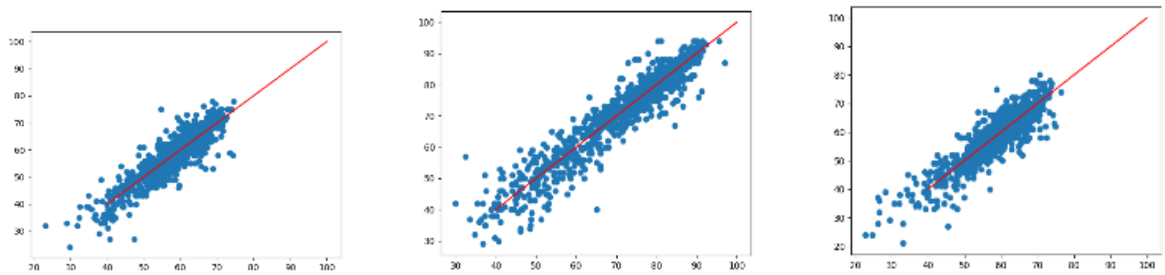


Figure 3.3: Coverage of Different Combinations of Heuristics

Although not exactly the same, some of the 6 heuristics and their reward-only counterparts are very similar, and their coverage are largely overlapped. Figure 3.3 gives the Venn diagram of some of the coverage of some of the heuristics. Among them, H1, H3' and H4 have the largest combined coverage and least overlapping. In the second experiment, we narrow our focus only to these three heuristics. GCN models are trained to predict their performance. In this experiment, a heuristic (among H1, H3' and H4) dominates a test case if it outperforms the other two.

Figure 3.4 plots the reward predicted by the GCN against the actual reward received after completing the bidding for heuristics H1, H3' and H4. All of the predictions are highly correlated to the actual value. The correlations are all above 0.98.

The goal of the performance predictor is to guide the process to select the appropriate



(a) H1 (b) H3' (c) H4

Figure 3.4: Predicted Reward Versus Actual Reward

heuristics for the given task allocation problem. We say that the machine learning model accurately selects the best heuristic if the heuristic with the highest predicted reward is truly dominating. As shown in Table 3.1, machine learning predicted heuristic can cover 69% of the test cases. This means for 69% of the time the machine learning model predicts the dominating heuristic correctly. If we constantly use a specific heuristic such as H1, H3' or H4, then their coverage is only 43.5%, 47.7% and 49.5%. By random selection, we have only 49.3% of chance to select the correct heuristic. Overall, the machine learning model improves the decision accuracy by more than 39%.

Table 3.1: Percentage Coverage of Heuristics and Improvements of the Predicted Approach

Heuristics	**Predicted**	**Random**	**H1**	**H3'**	**H4**
% Coverage	68.9%	49.3%	43.5%	47.7%	49.5%
% Improvement	0	39.7%	52.0%	44.4%	39.1%

3.6 Conclusions

In this work, we consider multi-agent task allocation with limited budget constraints. We compare the original consensus-based auction algorithm under those constraints, and extend it with several heuristic scoring functions. To ensure convergence, reward-only counterparts of those heuristics are also introduced, which separately consider selecting score and bidding score during CBBA. The performance of those heuristics is evaluated and compared. We show that those heuristics are Pareto efficient. Finally, we proposed a graph convolutional networks (GCNs) model to predict the total rewards that agent can receive for a specific task allocation problem when using different heuristic extensions, and use the predicted reward to guide the selection of appropriate scoring functions. The results show that we can correctly select the dominating heuristic for 69% of the testcases, which outperforms a random selection by 38%.

Chapter 4

Multi-agent Cooperative Games Using Belief Map Assisted Training

4.1 Introduction

A multi-agent cooperative game involves multiple autonomous systems collaborating with each other to achieve a common goal and maximize the overall utility of the system. These games can be used to model various applications, such as rescue missions where multiple robots are deployed to search for missing persons, military operations where multiple UAVs survey a large area, and scientific expeditions where rovers explore unknown terrain together. However, as the number of agents increases, centralized monitoring, controlling, and optimization becomes infeasible due to the exponential growth in complexity [46][47]. It will also increase the vulnerability of the system to single-point failures [48][49]. To overcome these issues, distributed control and optimization are introduced, where each agent makes its own decisions based on local information. However, this approach also has limitation, as agents only have partial observations of their immediate surroundings, and may not be able to make globally optimal decisions.

Message exchanges among agents can provide global information and help the agents

move out of local optima. However, excessive communication can consume communication energy, bandwidth, and processing power. Sending redundant messages in consecutive cycles, or by agents close to each other, is likely to waste resources. Additionally, frequently communicating every piece of observed information can be wasteful and also undermine the receiver's decision-making ability. Furthermore, to save communication energy and improve security, the high-dimensional observation should be encoded into a low-dimensional message that can only be decoded by the agents. Therefore, when to communicate, what to communicate and how to encode/decode the message are variables that need to be optimized. Reinforcement learning (RL), such as the actor-critic model, is commonly used to optimize multi-agent games. Manually design message passing system usually does not work well with the RL due to the lack of prior knowledge of the features that are needed by the policy network. A typical approach [50][51] is to train the message passing network together with the policy network so that they can evolve simultaneously.

Training a deep neural network using reinforcement learning is time consuming because the only feedback for the training is delayed, sparse, and indirect in the form of rewards. Training a multi-agent reinforcement learning (MARL) model [52][53] is even more challenging due to the fact that agents' decisions are not visible to one another. This lack of visibility reduces the predictability of the environment and makes it non-stationary. When a trainable message passing network is used to connect agents, things become even more complicated. The additional trainable variables in the message network significantly increase the model's complexity, prolong the training time, and escalate the chance of overfitting.

In this work, we accelerate the MARL by introducing another feedback channel that helps to learn a more efficient message passing network and a more effective representation of the environment. This consequently leads to better policy and faster convergence. In our belief-map assisted multi-agent system (BAMS)[1], each agent is supplemented with a map decoder, which transforms its hidden state into a belief map, a neuro-symbolic representation

[1]The code is available at Github

of the agent's knowledge of the global environment. This symbolic representation is simple, making it easy to obtain its corresponding ground-truth value. By comparing the belief map with the ground-truth map, the system receives an additional feedback that supervises the training process. During execution, the belief map provides a way to interpret the agent's hidden state, which can further be used to explain the agent's behavior.

To improve coordination among the agents and increase the efficiency of message retrieval, our message passing system incorporates gating and attention mechanisms. The attention model enables agents to differentiate important and irrelevant messages, while the gating removes the redundancy and saves communication power and bandwidth.

We assessed the performance of the BAMS model using a multi-agent predator-prey game with and without obstacles. Centralized training and distributed execution are adopted in the experiments. The experimental results indicate that BAMS outperforms existing models, proving to be more fitting for large-scale environments with complex landscapes and providing more robust performance.

The key contributions of this chapter are summarized as follows:

- We proposed a belief-map assisted training mechanism that complements reinforcement learning with supervised information to accelerate training convergence.
- We proposed a belief-map decoder to reconstruct a neuro-symbolic map from the environment embedding to provide additional feedback during the training. The map transforms the hidden state of agents into a human-readable format, which significantly improves the interpretability of the agent's decision-making process.
- Agents trained using BAMS model communicate more effectively, catching the prey faster and being less susceptible to noises from redundant messages as the number of agents increases.
- Simulation results show that agents with these enhancements can be trained effectively for operation in large and complex environments, reducing training time by an average

of 66% and improving overall performance by 34.62%.

The rest of the chapter is organized as follows. Section **??** introduces previous works related to communication in a multi-agent reinforcement learning system. Section **??** gives the details of our proposed method including the believe map decoder and attention model. The experimental results are given in Section **??** followed by the conclusions in Section **??**.

4.2 Background and Related Works

Previous research models a multi-agent communication system as a message passing graph neural network [54][55], where each node in the graph represents an agent and each edge models a communication pathway equipped with message encoding and decoding. Different graph topologies have been studied [56], and recent works focus on improving the efficiency and reducing the cost of the communication using gated message passing [57], attention [58], schedule communication [59] and event/memory driven processing [60][61] [62].

The first study on learnable communication, known as RIAL and DIAL [63], developed a message passing network that generates message generation based on the agent's local observation, action, and received messages. The message encoder is a multi-layer perceptron trained together with the policy network using reinforcement learning. CommNet [42] includes a centralized communication channel into the network, which enhances [63] by maintaining a local hidden state in each agent using a Recurrent Neural Network (RNN). The hidden state is determined by the sequence of local observations and received messages and is sent as the communication message to other agents. When multiple messages are received, the agent consolidates them using their average.

Message gating [64][51] has been proposed as a binary action to dynamically block or unblock message transmission, thereby improving communication efficiency and conserving power and bandwidth. IC3Net [51], an extension of CommNet [42], utilizes Long Short-Term Memory (LSTM) [65] to generate hidden states. Gated-ACML [57] performs message

pruning before transmission. For both approaches, communication gating is optimized by the policy network using reinforcement learning.

Other studies [64][66][67] have employed attention model to prioritize received messages so that agents can select useful features. ATOC [64] applies attention to determine which agent to communicate with, and dynamically changes network structure accordingly by generated a directed graph. G2ANet [66] combines a hard attention and a soft attention as two stage attention model to process different incoming message from different agents. MAGIC [67] uses a multi-layout graph attention network among agents. However, it performs centralized communication and message processing. All messages are sent to a communication hub where they are consolidated using the attention model and then broadcasted to all agents.

TarMAC [50] utilizes both gating and attention to enhance the communication efficiency. However, upon careful examination of its code, we found that an implementation error in the SoftMax function leads to unintended message leakage. If all agents gate their transmissions, the receiver may still receive this message. In other words, an agent i can only gate its message to another agent j if at least one other agent k, $1 \leq k \leq N$, $k \neq i$ or j, decided to send its message to j in the same cycle. As a result, agents must synchronize with each other regarding gating decisions during each cycle to determine whether to transmit messages.

All the works mentioned above train the message passing network with the policy network using the game rewards as the feedback. This approach tends to have a slow convergence and the agents do not understand each other well. In this chapter, we proposed a belief-map assisted training method (BAMS) that significantly improves the training speed and quality for large and complex games. The agents trained using BAMS communicates more efficiently with fewer messages and better attentions.

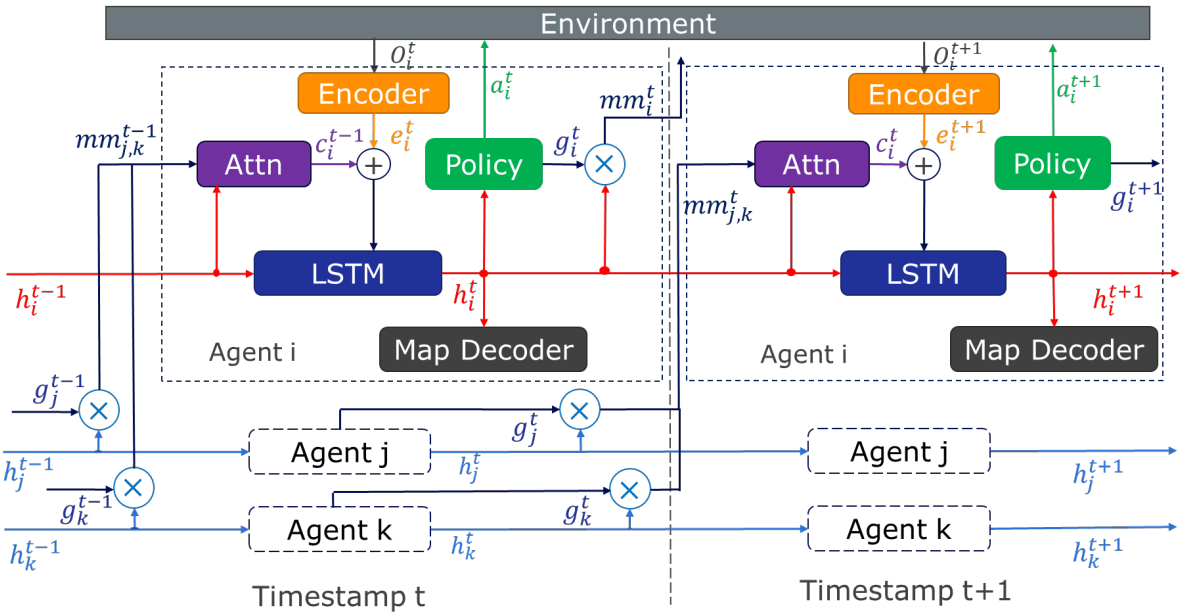


Figure 4.1: Architecture of BAMS Model

4.3 Proposed Method

In this section, we present the structure and training of belief-map assisted multi-agent system (BAMS). Details of the BAMS are illustrated in Figure 4.1. For each agent i, the model comprises five major components:

- Observation Encoder $E_i()$: The observation encoder extracts key features from the agent's local observation, which will later be combined with received messages and be used to update the hidden states.
- Message Attention Module $A_i()$: The attention module assigns weights to different messages to select relevant information.
- Hidden State Generator $lstm_i()$: The hidden state generator is a Long Short-Term Memory (LSTM) that fuses the local observation and received messages into a feature vector h_i.
- Policy Network $p_i()$: The policy network is an actor-critic model that selects the best action for the local agent to maximize the overall system utility. In BAMS, the action

consists of two parts, a discrete movement action a_i, which decides how agent moves to complete the game; and a binary communication action g_i, which decides whether the agent should broadcast its hidden state. The outgoing message mm_i is the product of g_i and h_i as shown in Figure **??**.

- Map Decoder $D_i()$: The decoder reconstructs a neuro-symbolic belief map of the environment based on the hidden state of the local agent. The belief map represents agent's knowledge of the global environment. It will be compared with the ground truth to provide additional feedback to assist the training.

4.3.1 Hidden State Generation and Policy Network

At each time step, every BAMS agent collects observations from its local sensor. The local observation for agent i at time t is denoted as o_i^t. Typically, the representation of o_i^t is designed manually and tailored to the specific application. The agent then update its hidden state, which is maintained by an LSTM, using both local observations and the received messages as the following:

$$h_i^{t+1}, s_i^{t+1} = lstm_i(E_i(o_i^t), c_i^t, h_i^t, s_i^t), \tag{4.1}$$

where h_i^t and s_i^t are hidden state and cell state at time t of agent i, and c_i^t is the aggregated feature extracted from the received messages using the attention model. $E_i(o_i^t)$ is the encoded observation.

Based on the hidden state, the agent chooses actions using a policy network $p_i()$. The policy network follows the actor-critic model and comprises an actor network $\theta_i(h_i^t)$ and a critic network $V_i(h_i^t)$. The $\theta_i(h_i^t)$ is a one-layer fully connected network with an input of h_i^t. Its output has two components a_i^t and g_i^t,

$$a_i^t, g_i^t = \theta_i(h_i^t). \tag{4.2}$$

The vector a_i^t represents the probabilities of the game actions available to the agent, i.e., the movement that the agent can make to complete the game. The variable g_i^t, which is either 1 or 0, represents the probability of the binary communication action, i.e., blocking or passing. At each step, the action was sampled according to the probability distribution.

4.3.2 Message Passing Model

Agents communicate their connected neighbors by sending messages. Following the approach used in TarMAC and IC3Net, we employ the hidden state as the communication message. The hidden state contains all the information that an agent requires to make local decisions. However, not all the information is useful to the agent's neighbors. Furthermore, some of the information may overlap with previous messages from the same agent or messages sent by a nearby agent. To improve the efficiency of the communication network, the senders must reduce the number of redundant messages they send and the receivers must be able to extract useful information relevant to their own decision making.

We implement the message gating at the sender side. The outgoing message mm_j^t of agent j is calculated as the product of h_j^t and the binary gate action g_j^t.

$$mm_j^t = h_j^t \times g_j^t. \tag{4.3}$$

After receiving messages $mm_j^t (j \neq i)$ from neighbor j, agent i aggregates the messages using an attention model, which is trained to maximize the reward from the game and minimize the loss of the belief-map construction. Considering the communication delay, agent i uses gated message mm_j^{t-1} send by agent j in previous time step as the input of the key and value networks to generate k_j^t and v_j^t for time t. The query q_i^t of the attention model is generated based on the agent's local hidden state at current time step (h_i^t).

$$k_j^t = key(mm_j^{t-1}) \tag{4.4}$$

$$v_j^t = value(mm_j^{t-1}) \tag{4.5}$$

$$q_i^t = query(h_i^t) \tag{4.6}$$

$$\alpha_i^t = softmax\left[\frac{(q_i^{t^T} k_1^t)}{\sqrt{(d_k)}} \cdots \frac{(q_i^{t^T} k_j^t)}{\sqrt{(d_k)}} \cdots \frac{(q_i^{t^T} k_1^t)}{\sqrt{(d_k)}}\right] \tag{4.7}$$

$$c_i^t = \sum\nolimits_{j=1}^{N} \alpha_i^t v_j^t \tag{4.8}$$

where $key()$, $value()$ and $query()$ are networks with one fully connected linear layer, d_k is the dimensions of hidden state. c_i^t is the aggregated feature vector that will be used to update the hidden state in Equation 4.1.

4.3.3 Map Decoder

Instead of relying solely on the reward from the environment, additional channels of feedback information could be added to expedite the training process. In this work we assist the RL by using a decoded belief map. As the aggregation of past observations and incoming messages, an agent's hidden state represents its knowledge of the environment. The more accurate this knowledge is, the better decision an agent can make. However, an agent's hidden state is a feature vector that is not interpretable. The basic idea of BAMS is to decode the hidden state into a neuro-symbolic map that is human interpretable, allowing for the construction of the ground truth version of the map. By comparing the decoded map with ground truth map, we provide additional feedback to assist the training of the entire system.

The map decoder $D_i(h_i^t)$ can be viewed as the inverse process of the observation encoder $E_i(o_i^t)$. The encoder $E_i(o_i^t)$ uses a Convolutional Neural Network (CNN) to extract the information. Therefore, we selected transposed CNN to decode the map. Both the observations and decoded maps are $m \times m$ gridded planes, where m is the size of the environment. The status of each grid location is coded as a size M vector, where M represents the number of possible states of the grid. For example, in the predator-prey game, a grid can have 4 possible states that indicate whether it has been observed, is currently occupied by a predator,

occupied by a prey, or occupied by an obstacle. These 4 states are not necessarily exclusive; hence each grid is encoded as a multi-hot vector with a size of M. Overall, both maps have dimension $M \times m \times m$. The observation map only contains information from the local agent, while the belief map should incorporate the information from all agents.

4.3.4 Loss Function

In this work, we apply centralized training and distributed execution. All components in the BAMS are trained together.

The training loss for each agent comprises two components, the L_{map} and the L_{RL}, $Loss = \alpha L_{map} + \beta L_{RL}$, where α and β are two hyper parameters. The map loss comes from comparing the decoded belief map (b_i^t) and the ground truth map $\mathbb{G}_i^t$. Mean Squared Error (MSE) is used to calculate the loss, $L_{map} = \sum_t MSE(\mathbb{G}_i^t - b_i^t)$. During training, the central controller tracks the movement and status of all agents to generate ground truth map. The map loss is obtained in every time step t. Minimizing the map loss can help all agents converge to an effective communication protocol and efficient message processing.

The RL loss is the error of the critic network,

$$L_{RL} = \sum\nolimits_t \|(r(h_i^t, a_i^t) + \gamma \hat{V}(h_i^{t+1}) - \hat{V}(h_i^t)\|^2 \tag{4.9}$$

where $r(h_i^t, a_i^t)$ is the reward of the entire system, and $\hat{V}()$ is the value estimation of the critic model. The actor network is updated using policy gradient:

$$\nabla_\theta J(\theta) = \sum\nolimits_t \nabla_\theta \log(p_\theta(a_i^t|h_i^t)[r(h_i^t, a_i^t) + \gamma \hat{V}(h_i^{t+1}) - \hat{V}(h_i^t)] \tag{4.10}$$

where $p_\theta()$ is the prediction of the actor network. The BAMS model is updated using the average gradient of all agents.

4.4 Experiments

For our experiments and evaluations, we utilized a classic grid-based predator-prey game [55]. It involves N predators (agents) with limited vision (v) to explore an environment of size $m \times m$ to capture either a static prey or a moving prey. The value of N ranges from 3 to 10, and m ranges from 7 to 20, representing games with varying complexity. The environment is further divided into 2 categories, with obstacles and without obstacles.

4.4.1 Experiment Setting

We trained our network using RMSprop [68] with a learning rate of 0.001 and smoothing constant 0.97. The entropy regularization is used with coefficient 0.01. The hidden state size for LSTM is 64. For the attention model, the key (k_j^t) and query (q_j^t) have a dimension of 16 and the value (v_j^t) has a dimension of 64.

The agents have limited observation capabilities. Specifically, each agent is only able to observe objects within a 3×3 or 5×5 area centered around itself. At each time step, an agent can choose from 5 possible actions: up, down, left, right, and stay. Additionally, all agents (predators) have a maximum step limitation, which varies according to the size of the environment. Prior to reaching the prey, an agent will receive a penalty $r_{searching}$ = -0.05 during each time step. Once an agent reaches the prey, it will remain there and receives no further penalty. The game is considered as complete when all agents reach the prey within the maximum number of steps. The number of steps taken to complete the game serves as the performance metric.

We conducted a comparison of BAMS with 2 baseline models: TarMAC and IC3Net. I3CNet employs message gating while TarMAC employs both message gating and attention. To the best of our knowledge, these models have state-of-the-art performance while employing decentralized communication and decision-making. For TarMAC, regardless of the gating action, the message will be sent, together with the gating action. So we also

implemented a variation of BAMS that removes the belief map decoder and conducts the training without the use of additional feedback. This reduced version of BAMS is referred to as BAMS-R.

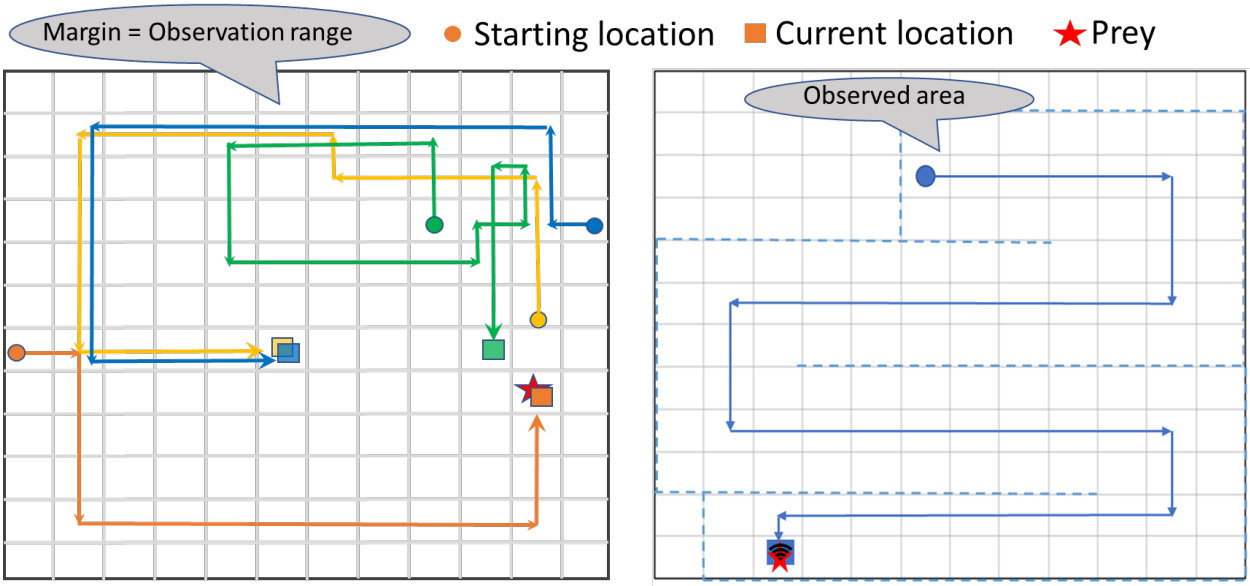


Figure 4.2: **Left** figure shows the example of 4 trajectories exhibiting keep the border within their observation range. **Right** figure shown the heuristic trajectory.

In addition to the aforementioned models, we implemented a heuristic rule-based algorithm. The algorithm directs the agents to explore the map from left to right, and top to bottom. after finishes exploring a row/column, an agent will move to the next row/column beyond its previous observation range. When it reaches the map's edge, it will turn around and explore in the opposite direction. Once an agent has sighted the prey, it will transmit the prey's location to all other agents, who will then take the shortest path to capture the prey. An example of the heuristic trajectory is shown as the right figure of Figure 4.2.

4.4.2 Experimental Results for Simple Environment

The first experiment is carried out on simple environment without any obstacles. We discovered that agents developed significant levels of intelligence and mutual understanding, allowing them to complete the game with minimum communications. For example, all agents

learned to explore the map by moving in a counterclockwise circle. Instead of exploring the entire map, agents circle a local region based on their initial position. Additionally, the agents tend to keep the border within their observation range while also staying as far from it as possible. These behaviors allow the agents to observe the maximum area while traveling the minimum distance. Left figure of Figure 4.2 presents an example of 4 trajectories exhibiting such behavior.

Table 4.1 compares the BAMS with four reference algorithms for games with different sizes when agents have 3×3 vision. The column "comm rate" shows the average percentage of times an agent transmits its hidden state. The results indicate that BAMS takes fewer steps on average to complete the game than the other algorithms. Specifically, compared to IC3Net and the heuristic algorithm, BAMS completes the game with approximately 30% fewer steps on average. Compared to TarMAC, BAMS completes the game with 6% fewer steps. However, it should be noted that agents using TarMAC transmit their hidden state much more frequently. Moreover, as mentioned in Section **??**, agents in TarMAC must synchronize with each other about their gating decision at every time step, which incurs significant overhead. The comparison between BAMS and BAMS-R demonstrates that the use of belief-map assisted training leads to a 27% reduction in the number of steps required to complete the game. As the map size increases, the communication rate of the BAMS agents reduces as the possibilities of encountering new events, such as observing another agent, the map edge, or the prey, decreases. In other words, the agents spend most of the time moving straight ahead.

Table 4.1: Avg Steps & Comm Rate for Simple Environments.

	N=3, m=7, Max Steps = 20		N=5 m=12, Max Steps = 40		N=10, m=20, Max Steps = 80	
	Avg steps	Comm rate	Avg steps	Comm rate	Avg steps	Comm rate
Heuristic	14.56	-	33.24	-	68.90	-
IC3Net	12.48	0.60	32.90	0.39	73.82	0.60
TarMAC	8.79	0.99	22.59	0.91	60.72	0.76
BAMS-R	12.39	0.32	29.80	0.04	71.76	0.35
BAMS	**8.17**	1.00	**21.64**	0.27	**56.46**	0.05

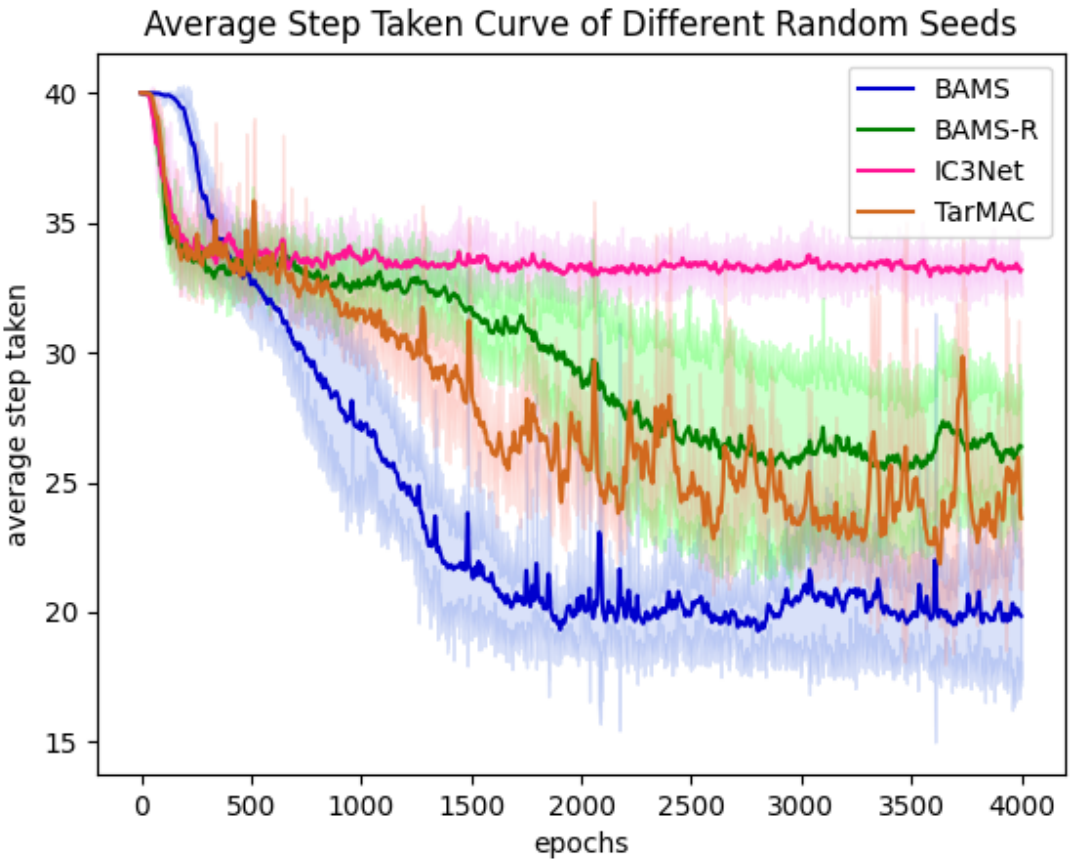
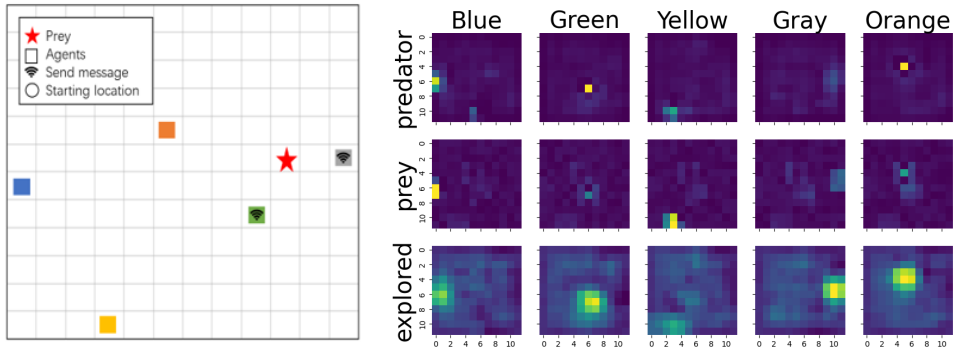


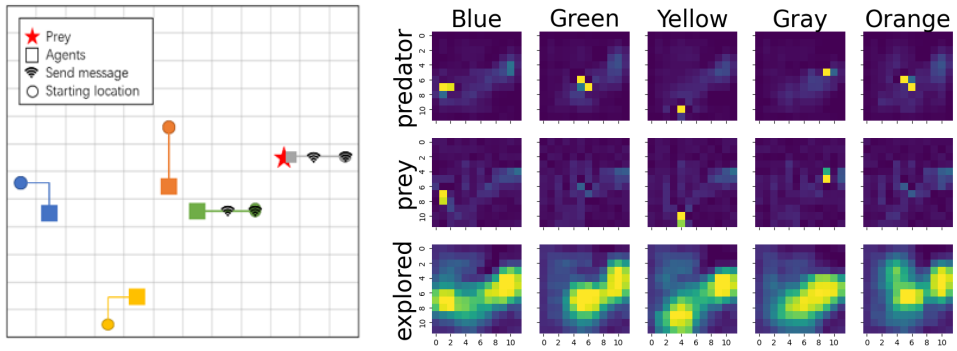
Figure 4.3: Average Step Taken Curve of Different Random Seeds

Figure 4.3 compares the convergence speed of BAMS with BAMS-R, Tarmac and IC3Net average step taken in random seeds under simple 12×12 environment. The results indicate that, IC3Net has the fastest convergence due to its relatively simpler architecture that does not employ an attention mechanism in message processing. However, for the same reason, it also has the worst performance. On average, BAMS improves the convergence by 66% compared BAMS-R. This improvement can be attributed to the additional feedback from the belief map, which provides a more consistent relationship among hidden state, action, and reward, resulting in faster learning with fewer iterations. Even TarMAC sends out messages every time step, our BAMS still beat the convergence of Tarmac. It should be noted that this feedback is only available during the training as no ground truth map is available during the execution. Nevertheless, the decoded belief map can provide a visualization of the agent's

hidden state and hence can be used to interpretate the agent's decision-making process.



(a) Step 1 Ground Truth Map (b) Step 1 Decoded Map



(c) Step 3 Ground Truth Map (d) Step 3 Decoded Map

Figure 4.4: **Left** grid figures (a) (c) is ground truth map shows the trajectory of agents. Square represents agent and star represents prey. Circle represents the starting location of agent, and the Wi-Fi icon represents that agent sent out a message on that step. **Right** heatmap figures (b) (d) give the visualized belief map of agents. Brighter grids indicate higher possibility that the grids are taken by agents, prey, or explored.

Table 4.2: Scalability Analysis of Model with Varying Numbers of Agents

	2		5		7		10		15	
	Avg steps	Comm rate	Avg steps	Comm rate	Avg steps	Comm rate	Avg steps	Comm rate	Avg steps	Comm rate
Heuristic	33.36	-	27.40	-	25.94	-	20.47	-	**15.56**	-
IC3Net	29.34	0.43	32.90	0.39	33.13	0.42	34.91	0.42	35.74	0.39
TarMAC	23.03	0.96	22.59	0.91	23.67	0.81	24.55	0.75	24.79	0.63
BAMS-R	28.53	0.04	29.80	0.04	32.55	0.04	33.77	0.05	34.82	0.06
BAMS	**23.04**	0.31	**21.64**	0.27	**19.32**	0.28	**18.76**	0.29	18.88	0.30

Figure 4.4b and Figure 4.4d depict an example of the decoded believe map for five agents at the beginning of the game and at time step 3 of the game, respectively. The gridded map shows the location of the agents and the prey, and the location where the agent sends out

a message. The 3 channels of the decoded map indicate the belief of the agents' location, the prey's location and the explored area. At the beginning of the game, all agents only have access of their local information. Interestingly, we found that the agents learned to be optimistic, as each agent believed that the prey was located nearby. The Gray agent reached the prey in step 2 and both Gray and Green agents sent out messages in steps 1 and 2. Therefore, at step 3, all agents updated their belief map to reflect the messages they received. In their prey location map, the areas around location $(5, 9)$ is highlighted, which reflects the correct prey location that they learned from the Gray agent. In their explored area map, the right side and center area of the map are highlighted, indicating the area that has been explored by the Gray and Green agents. The Green and Orange agents observed each other in step 3, resulting in the highlighting of each other's location in their location map. Interestingly, even though the Blue and Yellow agents did not send out any messages, the other agents still slightly highlighted the left and bottom sides of their explored area maps, as if they anticipated someone exploring this area. This suggests a type of mutual understanding without direct communication.

To test the robustness of the policies, we train the BAMS, BAMS-R and IC3Net model in an environment with 5 agents and test them in different environments with agent numbers varying from 2 to 15. The results are reported in Table 4.2, where we also listed the performance of the heuristic algorithm as a reference. As we expected, for BAMS, the average number of steps needed to complete the game reduces as the number of agents increases. However, for IC3Net and BAMS-R, the trend goes in the opposite direction. As the number of agents increases, due to the increased number of messages, the agents have difficulty extracting useful information, resulting in an increased number of steps to complete the game. This experiment demonstrates that BAMS helps to train an effective message passing framework, allowing agents to perform better in the game.

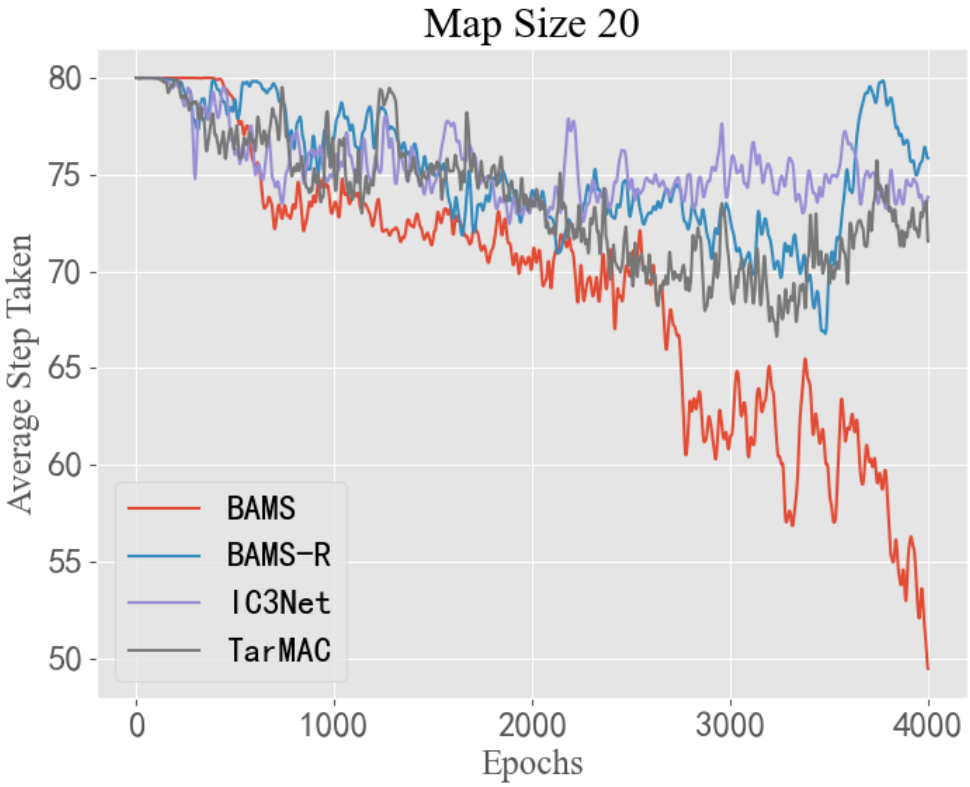
4.4.3 Experimental Results for Complex Environment

In the second experiment, we tested our approach under a complex environment with obstacles. Each grid in the environment is encoded as multi-hot vector of size 4, which represents whether the grid is occupied by a predator, a prey, or an obstacle, and whether it has been observed. We fixed the environment size to be 12×12. In each randomly generated training environment, there are 20 randomly placed obstacles

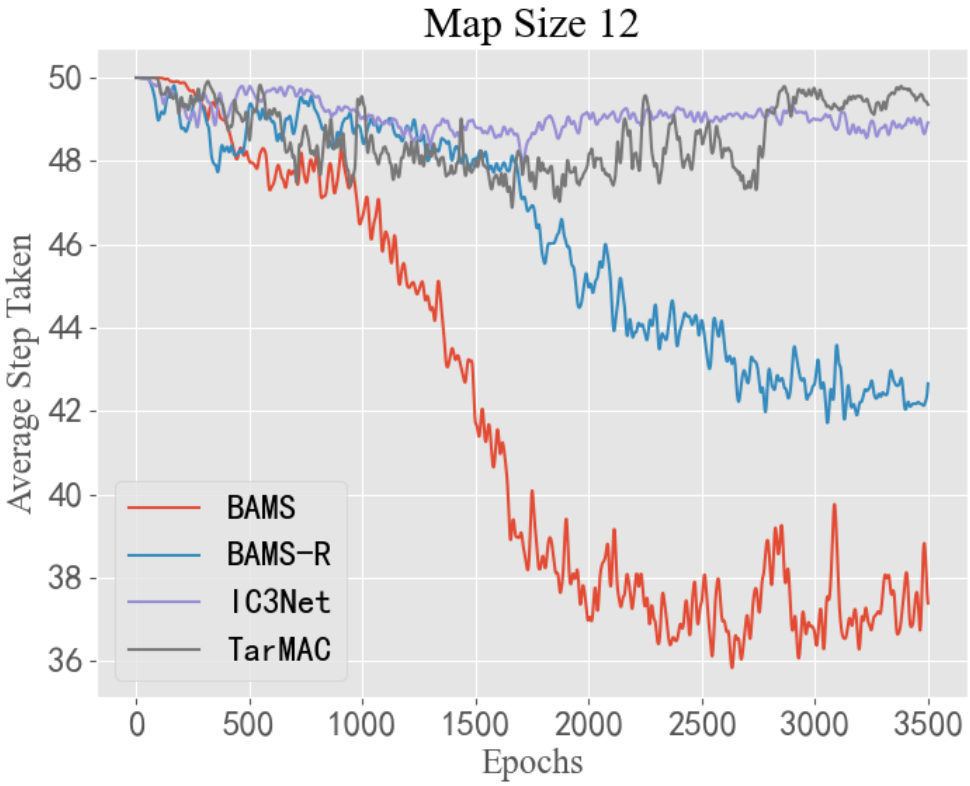
Figure 4.5b compares the convergence speed of BAMS, Tarmac, IC3Net and BAMS-R. We can see that BAMS once again has the fastest convergence speed compared to the other models, completing the game with 3 fewer steps than the other two models in average. In comparison to Figure 4.5a, the performance of IC3Net, which does not employ attention to the received messages, deteriorates much faster than Tarmac, BAMS-R and BAMS. This means effective message passing network becomes increasingly important in a complex environment.

We also observed that as the environment becomes more complex, the performance of those models oscillates more significantly. This can be seen in Figure 4.5a and Figure 4.5b when the environment size is 20 or when obstacles are included. The reason for this is that in randomly generated large and complex environments, the difficulty level of the game can vary significantly. Factors such as the initial location of the agents and distribution of obstacles can affect the number of steps needed to complete the game.

We further tested the model using testing environments with 10, 20 and 30 obstacles. We found that even though the network is trained with 20 obstacles, it was able to handle different environments. The performance of the three deep learning models in a complex environment is shown in Table 4.3. In average BAMS reduces the number of steps by 23.6% and 16.5% compared to IC3Net and BAMS-R, respectively.



(a) Simple 20×20



(b) Complex 12×12

Figure 4.5: Average Step Taken Comparison

Table 4.3: Avg Steps & Comm Rate for Complex Environments

No. of obstacles	10		20		30	
	Avg steps	Comm rate	Avg steps	Comm rate	Avg steps	Comm rate
IC3Net	45.39	0.53	48.56	0.54	49.37	0.57
BAMS-R	39.43	0.062	44.78	0.073	46.92	0.076
BAMS	**31.80**	0.056	**36.51**	0.065	**41.42**	0.054

4.5 Conclusion

This chapter proposes a novel training approach called belief map assisted training to improve the convergence and efficiency of multi-agent cooperative games with distributed decision-making. To overcome the issue of partial observation, attention-based inter-agent communication is adopted. The agents are trained to learn when to gate the message to save bandwidth and avoid interference with irrelevant information. We compared our approach with IC3Net and TarMAC in both simple and complex predator-prey environments. The experimental results show that our attention-based belief map can help the agents learn a better representation of the environment's hidden state and process messages effectively, leading to wiser decisions. Additionally, the belief map assisted training improves convergence speed and reduces the average number of steps needed to complete the game.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we study an unmanned aerial system traffic density prediction model and multi-agent task allocation with budget constraint problem. The contribution is summarized as follows:

- In Chapter 2, starting from our previous work of density prediction model, we proposed a technique to search for the best prune strategy for the UAV traffic density prediction model to facilitate fast traffic prediction. Our pruning model achieved a reduction in the size of the original traffic prediction model by 55% while keeping similar accuracy as the original model. At the same time, the model execution time speeds up by 60%.

- In Chapter 3, we present several heuristics method to improve the original CBBA algorithm under considering the budget constraint. Then we proposed a graph convolutional networks (GCNs) model to predict the total rewards that agents can receive for a specific task allocation problem when using different heuristic extensions, and use the predicted reward to guide the selection of appropriate scoring functions. The results show that we can correctly select the dominating heuristic for 69% of the testcases, which outperforms a random selection by 38%.

- In Chapter 4, we proposes a novel training approach called belief map assisted training (BAMS) to improve the convergence and efficiency of multi-agent cooperative games with distributed decision-making. The results show that our attention-based belief map can help the agents learn a better representation of the environment's hidden state and process messages effectively, leading to wiser decisions. We evaluate BAMS's performance in a cooperative predator and prey game with varying levels of map complexity and compare it to previous multi-agent message passing models. The simulation results showed that BAMS reduced training epochs by 66%, and agents who apply the BAMS model completed the game with 34.62% fewer steps on average.

5.2 Future Research Directions

As in the real-world environment, unlike certain games that afford complete observation, even complex scenarios such as StarCraft2, DOTA2 and Google Football Research impose limitations on an agent's observational capacity. The actions and state of agents heavily rely on what they observe and how they communicate. But fully communication requires a huge communication cost, which makes fully communication impossible. our future steps involve a continued exploration of partial observation and limited communication strategies for agents, particularly in the context of dynamic and complex environments. We aim to refine and extend our BAMS model to further enhance the cooperative and communicative capabilities of agents. First, we plan to delve deeper into the integration of advanced techniques, such as reinforcement learning and additional attention mechanisms, to augment the decision-making processes of agents. Second, we will explore ways to dynamically adjusting the degree of relationships among agents using Graph Convolutional Networks (GCN). Furthermore, we will consider the incorporation of multi-modal learning, enabling agents to leverage various sources of information beyond visual observations. This can include incorporating data from sensors, textual inputs, or other relevant modalities, enriching the agent's understanding of

the environment.

Bibliography

[1] M. Odelga, P. Stegagno, and H. H. Bülthoff, "Obstacle detection, tracking and avoidance for a teleoperated uav," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 2984–2990.

[2] H. L. N. N. Thanh and S. K. Hong, "Completion of collision avoidance control algorithm for multicopters based on geometrical constraints," *IEEE Access*, vol. 6, pp. 27 111–27 126, 2018.

[3] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[4] S. A. Bortoff, "Path planning for uavs," in *Proceedings of the 2000 american control conference. ACC (IEEE Cat. No. 00CH36334)*, vol. 1, no. 6. IEEE, 2000, pp. 364–368.

[5] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous uav path planning and estimation," *IEEE Robotics & Automation Magazine*, vol. 16, no. 2, pp. 35–42, 2009.

[6] R. W. Beard and T. W. McLain, "Multiple uav cooperative search under collision avoidance and limited range communication constraints," in *42nd IEEE international conference on decision and control (IEEE Cat. No. 03CH37475)*, vol. 1. IEEE, 2003, pp. 25–30.

[7] B. D. Song, J. Kim, and J. R. Morrison, "Rolling horizon path planning of an autonomous system of uavs for persistent cooperative service: Milp formulation and efficient heuristics," *Journal of Intelligent & Robotic Systems*, vol. 84, pp. 241–258, 2016.

[8] Z. Zhao, Z. Jin, C. Luo, H. Fang, F. Basti, M. C. Gursoy, C. Caicedo, and Q. Qiu, "Temporal and spatial routing for large scale safe and connected uas traffic management in urban areas," in *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2019, pp. 1–6.

[9] Z. Jin, Z. Zhao, C. Luo, F. Basti, A. Solomon, M. C. Gursoy, C. Caicedo, and Q. Qiu, "Simulation of real-time routing for uas traffic management with communication and airspace safety considerations," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, 2019, pp. 1–10.

[10] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[11] E. Russo, M. Palesi, S. Monteleone, D. Patti, A. Mineo, G. Ascia, and V. Catania, "Dnn model compression for iot domain-specific hardware accelerators," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6650–6662, 2021.

[12] Z. Li, Z. Wang, L. Xu, Q. Dong, B. Liu, C.-I. Su, W.-T. Chu, G. Tsou, Y.-D. Chih, T.-Y. J. Chang *et al.*, "Rram-dnn: An rram and model-compression empowered all-weights-on-chip dnn accelerator," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1105–1115, 2020.

[13] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[14] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[15] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[16] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017.

[18] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[19] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[20] Z. Zhao, C. Luo, A. Solomon, F. Basti, C. Caicedo, M. C. Gursoy, and Q. Qiu, "Machine learning-based traffic management model for uas instantaneous density prediction in an urban area," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–10.

[21] X. Dai, H. Yin, and N. K. Jha, "Nest: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1487–1497, 2019.

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[23] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[24] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns," *arXiv preprint arXiv:1807.11091*, vol. 2, no. 3, 2018.

[25] T. Zhang, S. Ye, K. Zhang, X. Ma, N. Liu, L. Zhang, J. Tang, K. Ma, X. Lin, M. Fardad *et al.*, "Structadmm: A systematic, high-efficiency framework of structured weight pruning for dnns," *arXiv preprint arXiv:1807.11091*, 2018.

[26] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," *Advances in neural information processing systems*, vol. 26, 2013.

[27] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *Advances in neural information processing systems*, vol. 27, 2014.

[28] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, "Multi-task allocation and path planning for cooperating uavs," *Cooperative control: models, applications and algorithms*, pp. 23–41, 2003.

[29] C. Schumacher, P. R. Chandler, and S. R. Rasmussen, "Task allocation for wide area search munitions," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 3. IEEE, 2002, pp. 1917–1922.

[30] L. Xu and U. Ozguner, "Battle management for unmanned aerial vehicles," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 4. IEEE, 2003, pp. 3585–3590.

[31] D. Turra, L. Pollini, and M. Innocenti, "Fast unmanned vehicles task allocation with moving targets," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 4. IEEE, 2004, pp. 4280–4285.

[32] L. Brunet, H.-L. Choi, and J. How, "Consensus-based auction approaches for decentralized task assignment," in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 6839.

[33] L. B. Johnson, H.-L. Choi, S. S. Ponda, and J. P. How, "Decentralized task allocation using local information consistency assumptions," *Journal of Aerospace Information Systems*, vol. 14, no. 2, pp. 103–122, 2017.

[34] L. Johnson, H.-L. Choi, S. Ponda, and J. P. How, "Allowing non-submodular score functions in distributed task allocation," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 4702–4708.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[37] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," *arXiv preprint arXiv:1902.06673*, 2019.

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[39] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[40] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*. PMLR, 2016, pp. 2014–2023.

[41] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[42] S. Sukhbaatar, R. Fergus *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, 2016.

[43] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," *Advances in neural information processing systems*, vol. 30, 2017.

[44] A. Agarwal, S. Kumar, and K. Sycara, "Learning transferable cooperative behavior in multi-agent teams," *arXiv preprint arXiv:1906.01202*, 2019.

[45] A. Malysheva, T. T. Sung, C.-B. Sohn, D. Kudenko, and A. Shpilman, "Deep multi-agent reinforcement learning with relevance graphs," *arXiv preprint arXiv:1811.12557*, 2018.

[46] R. Huang, X. Chu, J. Zhang, and Y. H. Hu, "Energy-efficient monitoring in software defined wireless sensor networks using reinforcement learning: A prototype," *International Journal of Distributed Sensor Networks*, vol. 2015, 2015.

[47] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[48] G. S. Lynch, *Single point of failure: The 10 essential laws of supply chain risk management*. John Wiley and Sons, 2009.

[49] M. Moradi, "A centralized reinforcement learning method for multi-agent job scheduling in grid," in *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2016, pp. 171–176.

[50] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, "Tarmac: Targeted multi-agent communication," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1538–1546.

[51] A. Singh, T. Jain, and S. Sukhbaatar, "Learning when to communicate at scale in multiagent cooperative and competitive tasks," *arXiv preprint arXiv:1812.09755*, 2018.

[52] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[53] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[54] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11 785–11 792.

[55] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, "Multi-agent game abstraction via graph attention neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7211–7218.

[56] J. Sheng, X. Wang, B. Jin, J. Yan, W. Li, T.-H. Chang, J. Wang, and H. Zha, "Learning structured communication for multi-agent reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 2, p. 50, 2022.

[57] H. Mao, Z. Zhang, Z. Xiao, Z. Gong, and Y. Ni, "Learning agent communication under limited bandwidth by message pruning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5142–5149.

[58] M. Geng, K. Xu, X. Zhou, B. Ding, H. Wang, and L. Zhang, "Learning to cooperate via an attention-based communication neural network in decentralized multi-robot exploration," *Entropy*, vol. 21, no. 3, p. 294, 2019.

[59] D. Kim, S. Moon, D. Hostallero, W. J. Kang, T. Lee, K. Son, and Y. Yi, "Learning to schedule communication in multi-agent reinforcement learning," *arXiv preprint arXiv:1902.01554*, 2019.

[60] G. Hu, Y. Zhu, D. Zhao, M. Zhao, and J. Hao, "Event-triggered communication network with limited-bandwidth constraint for multi-agent reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[61] E. Pesce and G. Montana, "Improving coordination in small-scale multi-agent deep reinforcement learning through memory-driven communication," *Machine Learning*, vol. 109, no. 9, pp. 1727–1747, 2020.

[62] D. Simões, N. Lau, and L. P. Reis, "Multi agent deep learning with cooperative communication," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 10, 2020.

[63] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 29, 2016.

[64] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," *Advances in neural information processing systems*, vol. 31, 2018.

[65] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[66] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, "Multi-agent game abstraction via graph attention neural network," 2019.

[67] Y. Niu, R. R. Paleja, and M. C. Gombolay, "Multi-agent graph-attention communication and teaming." in *AAMAS*, 2021, pp. 964–973.

[68] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

Vita

Chen Luo received his B.S. degree in Automation in 2015 from Hunan University, Hunan, China and M.S. degree in Computer Engineering in 2016 from Syracuse University, NY, US. He has completed the Ph.D. degree in the Department of Electrical Engineering and Computer Science, Syracuse University, in 2023. His research interests are in the fields of computer vision, multi-agent system, reinforcement learning and robotics.