Learning and Control in Multi-Agent Systems with Applications on Cyber-Physical Systems

by

Jianyu Su

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Systems Engineering
at the
UNIVERSITY OF VIRGINIA

May 2021

Signature of the Author _____

Certified by _____
PhD Program Director Date

Ph.D. IN SYSTEMS ENGINEERING

UNIVERSITY OF VIRGINIA

CHARLOTTESVILLE, VIRGINIA

CERTIFICATE OF APPROVAL

Ph.D. DEGREE DISSERTATION

The Ph.D. degree dissertation of Jianyu Su
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Systems Engineering

_____

Peter A. Beling, Dissertation Advisor

_____

Arsalan Heydarian, Chair

_____

Hongning Wang

_____

Qing Chang

_____

Stephen Adams

_____

Date

Acknowledgement

I want to thank my adviser Prof. Dr. Peter A. Beling for accepting me as his PhD student and for his longstanding support and excellent supervision since my joining his lab. I am deeply grateful to Dr. Stephen Adams for his invaluable discussions, as well as his patience.

I am wholeheartedly appreciative of Dr. Arsalan Heydarian's great support and helpful advice throughout the years. The experience of writing grant proposal with Arsalan refines my research vision and horns my writing skills, laying the cornerstone for my phd research.

This thesis emerged from my research at Toyota Infotech Labs, Mountain View, CA. I am thankful to my project supervisor Dr. Kyuangtae Han for giving me the opportunity to join their research team.

I wish to thank my friends and Link Lab colleagues Yawen Shen, Alan Wang, Jingyun (Brian) Ning, and Wenpeng (Nelson) Wang for their friendship, support, mental therapy, and academic discussion over hotpot. I also want to thank Jing Huang, Professor Qing Chang for helping me implement my algorithm on manufacturing systems. I want thank Hongning Wang for offering the information retrieval class, which motivates me to learn many useful methods and concepts from the field of natural language processing.

Last but not the least, I want to thank my family for supporting my decision to earn my degree in US.

This dissertation is dedicated to my love, Ziyue (Zoey) Li, my parents, Zonghai Su and Wenqiong Tan, and people that influence me the most Shenwu Tian, Wenfeng Tan, Yanming Tian, and Zuobing Yuan.

Abstract

Recent years have seen the application of machine learning (ML) to various domains. Coupled with deep neural networks, machine learning methods are cornerstones of adaptive learning and decision-making frameworks. However, Conveniently adopting ML techniques without carefully examining the systems' structures and ML techniques' assumptions often leads to systems' sub-optimal performance. For instance, reinforcement learning (RL), which is a machine learning technique that estimates optimal policies through interactions between an agent and the environment, is designed for single-agent systems. In practice, a centralized RL framework is often utilized to coordinate agents in multi-agent settings. As a result, the action space of a such centralized framework grows exponentially with the number of agents included in the system, which might lead to the system's poor performance. In this dissertation, we present a multi-agent RL framework that is applicable to various cooperative multi-agent systems, as well as proposing new methods to bridge the gaps in the current literature for tasks such vehicle trajectory prediction and decision making in large-scale manufacturing systems. Our contribution is three-fold:

1. We propose a graph-based vehicle acceleration framework *Traffic Graph Framework* (TGF), which captures hierarchical and chains of interactions that might affect the predicted vehicle's future state. The proposed framework utilizes new variants of graph convolution that are specifically adopted for modeling the traffic. Combined with the flexibility of graph data structure, TGF treats the traffic as a multi-agent system and can be employed for various traffic configurations with high prediction quality.

2. We propose a novel MARL algorithm, value-decomposition multi-agent actor-critic (VDAC). VDAC is an on-policy actor-critic that is compatible with the parallel training paradigm, A2C. As a result, VDAC offers a reasonable trade-off between training efficiency and algorithm performance. In our competitive evaluation, VDAC reports higher win rates than other multi-agent actor-critics on complex multi-agent coordination tasks, StarCraft II micromanagement games.

3. We propose an adaptive preventive maintenance (PM) scheduling framework based on VDAC for large-scale manufacturing systems. In the simulation study, the proposed framework demonstrates its effectiveness by leading other baselines, including RL-based methods and traditional maintenance models, on a comprehensive set of metrics. Our analysis further demonstrates that our MARL-based method learns effective PM policies without any knowledge about the environment and maintenance strategies.

Contents

List of Symbols

The next list describes some of the main symbols that will be later used in the Chapter 1. Other symbols will be introduced throughout the body of this document.

Graph and Graph Convolution

G A directed or an undirected graph

MARL

A A set of agents in a multi-agent system

a An agent index that follows $a \in A$

U A set of actions that are available for every agent

u^a An action taken by an agent indexed by a

$\mathbf{u}$ A joint action

S A set of true states

π^a A policy taken by an agent indexed by a

π A joint policy

r A reward

o^a An observation received by an agent indexed by a

τ^a An observation-action history of an agent indexed by a

$\mathbf{u}^{-a}$ A joint action that excludes the action of an agent a

π^{-a} A joint policy that excludes the policy of an agent a

A^a An advantage function with respect to agent a

Q	An action-value function
V	An state-value function
V_{tot}	An state-value function that quantifies global state values
V^a	An state-value function that quantifies agent a's local observation values
J	The objective of a MARL problem
γ	The discount factor
θ	Learnable weights in policy networks

Chapter 1

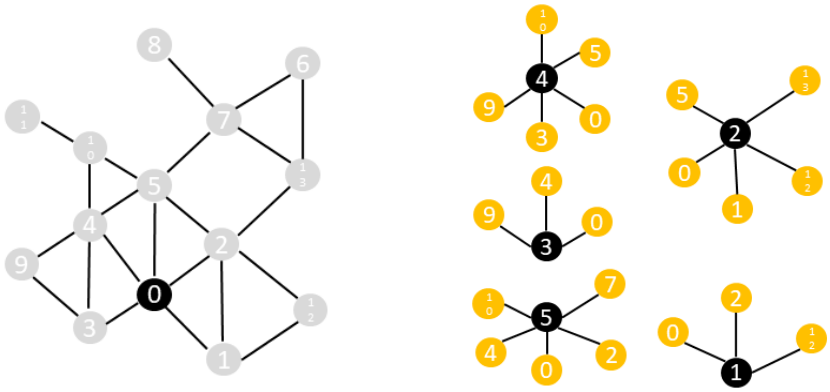
Introduction

Recent years have seen rapid developments in the field of deep machine learning (ML). Benefiting from deep ML's ability to extract features, deep ML, including deep reinforcement learning, has been applied across different domains.

In the field of intelligent driving assistance, car-following models have moved away from traditional approaches, which make assumptions about driver's state such as reaction time, to deep learning models. While those deep learning attempts differ from each other by the number of interactions among the ego-vehicle and its surrounding vehicles, they share a commonality that only inputs of the fixed spatial organization are allowed. This impeds those methods from generalizing into practice as traffic scenarios in reality can be very dynamic.

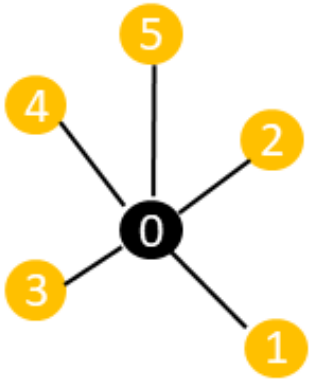
Graph is a flexible data structure. In a graph, a node can be connected to an arbitrary number of nodes, which allows graph to be applied to various domains such as social science, biology, etc. Graph convolution, which directly operates on graphs, aggregates neighbor information for the central node, which represents the process of communication between the central node and its neighbors. Furthermore, a 2-layer graph convolution enables nodes to communicate with nodes that are 2 hops away as shown in Figure 1.1.

Our traffic graph algorithm, combining the flexibility of graph data structure and the representation ability of graph convolutions, facilitates high-quality ego-vehicle trajectory prediction under numerous traffic scenarios. This algorithm allows the ego-vehicle to communicate with a dynamic number of neighbors to learn its surrouding traffic. Although this supervised algorithm achieves state-of-the-art performance on generating human-like driving trajectories, it cannot be applied to multi-agent control problems as the error accumulates over the time horizon. In contrast, RL, which maximizes a long-term objective, is adaptive to environmental dynamics and has shown human-level performance on many control tasks without prior knowledge. Therefore, we turn to multi-agent RL for multi-agent system



(a) A graph

(b) The first graph convolution operation



(c) The second graph convolution operation

Figure 1.1: Illustration for 2-layer graph convolution operation

control.

RL is a family of machine learning techniques that learns optimal policies through the interactions between the agent and the environment. It does not necessarily assume an intricate understanding of the environment (environment dynamics). Instead, the RL agent gathers information about the environment by exploring it. And the interaction proceeds in the following sequence: RL agent receives an environment state, on which it conditions its action. The environment proceeds to

the next state once the RL agent takes its action. At the same time, the RL agent obtains a reward from the environment. Multi-agent RL is a branch of RL, where more than one agents work independently but collectively for a common goal of the environment. More specifically, agents condition their actions on local partial-observables, instead of true states of the environment. The individual actions taken simultaneously constitute a joint action. The environment proceeds to the next true state and returns a reward for the joint action. It is possible to directly apply single-agent RL to multi-agent tasks. However, such practice only neglects the interactions among agents but also suffers from the magnitude of action space arised from the number of agents in the system. In general, the lack of tailored rewards and the access to true states pose difficulties on multi-agent coordination problems.

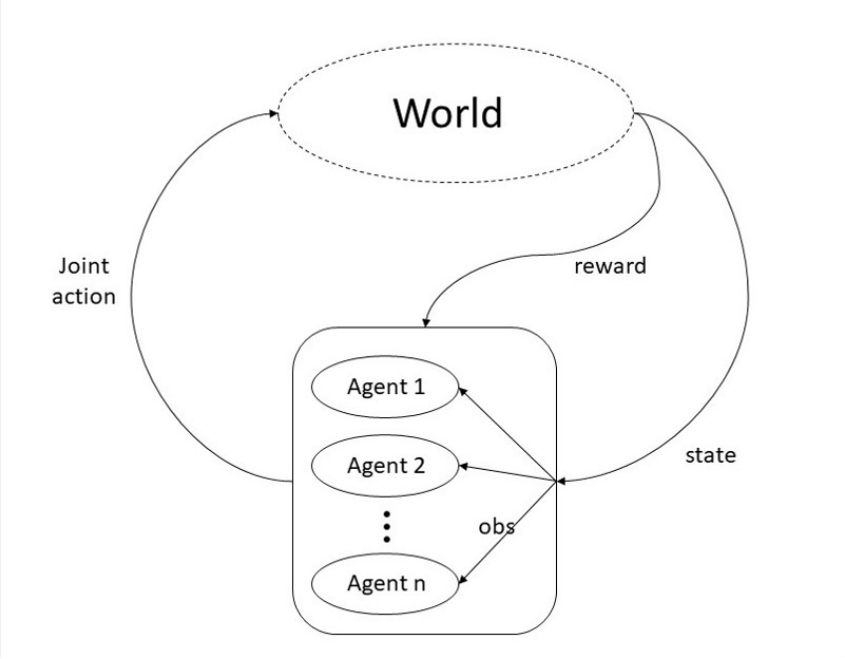


Figure 1.2: Multi-agent RL setup

The Last few years have seen the advance of multi-agent RL, especially value-based methods. For instance, *Value-decomposition networks* (VDN) represent joint action-value as a summation of local action-value conditioned on individual agents' local observation history [82]. QMIX [73], a more general case of VDN, uses a mixing network that approximates a broader class of monotonic functions to represent joint action-values. In [79], a more complex factorization framework three modules, called QTRAN, is introduced and shown to have good performance on a range of cooperative tasks. While these value-based methods report a good performance on most of the StarCraft micromanagement games [76], policy-based

methods such as COMA [29] struggles with some of the easiest tasks in the StarCraft micromanagement testbed.

Multi-agent policy gradients methods are facing a number of issues :

- High variance gradient estimates exacerbated in the multi-agent setting [60]
- The discrepancy between the local action-value and global action-value (*the credit assignment issue*
- The proof of the learned policy converging to a locally optimal policy

Difference rewards, $D^a = r(s, \mathbf{u}) - r(s, (\mathbf{u}^{-a}, c^a))$, presents a principled framework for distributing rewards among agents based on their performance. The shaped reward D^a, defined by a reward change incurred by replacing the original action u^a with a default action c^a, precisely reflects the contribution of agent a to the system by taking action u^a. *Difference rewards* requires $|A| \times |U|$ number of parallel simulation to be rolled out at every time step to calculate D^a for every agent, which is impractical for complex tasks with numerous agents. While *difference rewards* might be demanding to implement, it possesses two properties that is useful for MARL:

- Monotonic relations holds between shaped rewards D^a and the global reward $r(s, \mathbf{u})$
- The baseline, $r(s, (\mathbf{u}^{-a}, c^a))$, does not depend on u^a

Therefore, any advantage function takes the form $A^a = Q(s, \mathbf{u}) - b$, where b does not depend on u^a, suffice the relation implied by *difference rewards*.

COMA is a *credit-assignment* actor-critic whose convergence to a locally optimal policy is established. COMA devices tailored gradients for agents based on their contribution to the overall system. COMA gradients can be written as $\nabla_\theta J = \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) A^a(s, \mathbf{u}) \right]$, whose expectation, however, is proven to be exactly the expectation of a vanilla policy gradients. COMA's ability to assign credits is questionable as it predicts counterfactual action-values for unseen actions. Furthermore, COMA predicts counterfactual action-value for each action of each agent, having issues generalizing to large-scale multi-agent problems.

This thesis presents a principled and effective actor-critic framework, value-decomposition multi-agent actor-critic (VDAC), by enforcing the the monotonic relation between local state-values V^a and the global state-value V_{tot}, $\frac{\partial V_{tot}}{\partial V^a} \geq 0, \quad \forall a \in \{1, \ldots, n\}$, and a rather simple TD-advantage policy gradients, $r_t + \gamma V(s_{t+1}) - V(s_t)$, that can scale to large multi-agent applications.

The successive step is to apply VDAC to a real-world application to demonstrate the effectiveness and efficiency of VDAC. In manufacturing systems, machines suffer from random failures as the result of degradation of parts over time [89]. These unexpected failures abruptly interrupt normal production operations, which can not only cause significant production losses but also require considerable resources for prompt maintenance actions. The maintenance procedure in response to random failures is referred to as corrective maintenance (CM). To reduce random failures, a common industrial practice is to proactively shut down machines for preventive maintenance (PM) according to predefined policies. However, deriving PM policies that ensure smooth and efficient production has always been a nontrivial task since PM actions also impact the system in ways that can incur production losses and resource costs. The costs of excessive PM might outweigh the benefits, but conversely, inadequate PM can be ineffective in preventing random failures. It is challenging to balance the delicate decision trade-offs that arise in PM for manufacturing systems, which are distinguished by complicated and non-linear system dynamics due to interactions among machines/buffers and interruptions from production-related activities [39]. Recent model-free reinforcement learning (RL) methods shed light on how to cope with the nonlinearity and stochasticity in such complex systems. However, the action space explosion impedes RL-based PM policies to be generalized to real applications. In order to obtain cost-efficient PM policies for a serial production line that has multiple levels of PM actions, a novel multi-agent modeling is adopted to support adaptive learning by modeling each machine as a cooperative agent.

The principal contribution of this thesis is threefold:

- We propose a graph-based vehicle acceleration framework *Traffic Graph Framework* (TGF), which captures hierarchical and chains of interactions that might affect the predicted vehicle's future state. TGF builds graphs based on the defined relationship among vehicles. TGF's key element is a 2-layer graph convolution that can directly operate on graphs, which enables the predicted vehicle to "communicate" with neighbor vehicles that are 2 hops away. Unlike other deep learning methods, the proposed framework does not require inputs of a fixed spatial organization, thus can be employed in applications with various traffic configurations.

- We present a general-purpose multi-agent RL algorithm, value-decomposition multi-agent actor critic (VDAC) for various multi-agent control tasks. VDAC is a credit-assignment actor-critic inspired by *difference rewards*. It learns a viable approach to distributing rewards to agents. The proposed VDAC outperforms other policy-based algorithms in complex multi-agent coordination tasks, StarCraft II micromanagement games.

- Designing preventive maintenance (PM) policies that ensure smooth and efficient production for large-scale manufacturing systems is non-trivial. Recent model-free reinforcement learning (RL) methods shed light on how to cope with the nonlinearity and stochasticity in such complex systems. However, the action space explosion impedes RL-based PM policies to be generalized to real applications. To address the limitation of RL-based PM policies, We apply VDAC to manufacturing systems to obtain PM policies effectively and efficiently. In simulation studies, the proposed framework demonstrates its effectiveness by leading other baselines on a comprehensive set of metrics whereas the centralized RL-based method struggles to converge to stable policies. Our analysis further demonstrates that our multi-agent reinforcement learning based method learns effective PM policies without any knowledge about the environment and maintenance strategies.

The rest of this dissertation is organized as follows:

Chapter Graph Convolution for Traffic. We introduce the necessary background on graph and graph convolution. We also provide details on converting traffic states to traffic graphs and the adaptation we made on the original GCN. Finally, benchmark tests are presented to demonstrate the effectiveness of the proposed framework.

Chapter Value-Decomposition Multi-Agent Reinforcement Learning. We present VDAC, a general-purpose multi-agent actor-critic. Additionally, preliminaries and necessary background on multi-agent RL are introduced. We further report VDAC's performance on complex multi-agent tasks, StarCraft II micromanagement games, demonstrating VDAC's effectiveness and efficiency.

Chapter Preventive Maintenance with VDAC. We first identify the gaps in the current literature of preventive maintenance (PM) policies for manufacturing systems. We demonstrate how to adopt VDAC to PM scheduling tasks for a production line. At last, we present the benchmark comparison between the VDAC-based PM policies and other baselines, including traditional methods based on heuristics and a RL-based method. Two simulation studies are conducted to show both the effectiveness and the scalability of the proposed framework.

Chapter 2

Graph Convolution for Traffic

2.1 Background

Autonomous pilots or intelligent driving assistants predict the future state of traffic in order to warn human drivers about collision risks. The autonomous system in the ego-vehicle should consider not only the ego-vehicle's interactions with its immediate neighbors, but also hierarchical and chains of interactions that might affect the ego-vehicle's future state.

Many approaches have been proposed to predict the behavior of vehicles, with most methods falling into the broad categories of regression formulations or classification formulations. While formulating the problem of predicting vehicle behaviors as a classification problem makes it easier to train the model and compare its performance, this classification approach fails to provide detailed future traffic information for planning the future trajectory. Regression methods, however, are able to infer the future state of traffic, such as vehicle positions, velocities and accelerations. In the literature, many of the methods for the regression formulation of traffic prediction employ Recurrent Neural Networks (RNNs). RNNs are widely used to study time-series data. In particular, researchers have been successfully applying Long-Short Term Memory (LSTM) network to various applications such as speech generation, machine translation, and speech recognition [37]. In this work, we also use an RNN structure as part of our proposed framework.

A principal weakness of existing driving behavior prediction methods is that they use models that require inputs of fixed size and fixed spatial organization, making it difficult to generalize from training sets into practice. In [67], for instance, the proposed method uses a leader-follower model that focuses only on the interactions between the ego-vehicle and its leading vehicle. More recently, neighbor models that capture more interactions between ego-vehicle and its surrounding vehicles have been proposed [4, 57]. Though these neighbor methods show some success in

predicting the ego-vehicle's future acceleration, they only consider a fixed number of neighbor vehicles. In addition, they need to deal with information padding if one of the pre-defined neighbors is absent.

2.2 Executive Summary

In this study, we propose a flexible driving behavior prediction framework that we call the *Traffic Graph Framework*. Combining Graph Convolution Networks (GCNs) and LSTMs, our proposed method is able to capture not only spatial features of various sizes but also temporal features. This framework consists of undirected graphs that represent the interactions between vehicles, a multi-layer graph convolution neural network used to directly encode the graph structure, and a fully-connected or LSTM mixture density network used to predict future acceleration distributions.

In series of empirical tests, we investigate the the performance of our proposed models relative to baselines, including GAT and other GCN variants. The test environment for our methods is a simulation designed to mimic real-world traffic. The simulation is built using the NGSIM I-80 dataset, which contains vehicle trajectories of more than 2000 individual drivers [20]. In the simulation, ego-vehicles' traffic states are propagated based on models' predictions. Models are evaluated by comprehensive metrics to measure the discrepancy between the generated trajectories and the ground truth. Furthermore, ablation studies were performed to analyze the effectiveness of the proposed GCNs and RNN architectures. Results show that including the proposed GCNs and RNN structure improves model's prediction quality.

2.3 Graph Convolution

In this study, we consider an undirected graph $G = (E, V)$ with N nodes $v_i \in V$, edges $(v_i, v_j) \in E$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$, a degree matrix with $D_{ii} = \sum_j A_{ij}$, and a nodes feature information matrix $X \in \mathbb{R}^{N \times F}$. This graph G is utilized to describe the relationship of interests (i.e. whether two vehicles are considered as neighbors) among vehicles on highway.

Graph neural networks (GNNs) are a type of neural network designed for the analysis of graphs [98]. Recently, GNNs have been drawing increasing attention from both academia and industry for the flexibility that the graph data structure provides and for their convincing performance on various tasks in different domains, such as social science [34, 51], neural science [30], and knowledge graphs [33]. For instance, motivated by a first-order approximation of spectral convolution on a graph, Graph Convolution Networks (GCNs) are a computationally efficient variant

of GNNs that have shown success in achieving fast and scalable classification of nodes in a graph [51]. Another class of GNNs is the Graph Attention Network (GAT), which utilizes *self-attention* [7] to allow for inductive reasoning among nodes, thereby providing additional interpretability while matching other GNNs on benchmark evaluation.

GCN: GCN takes input as a graph G and output nodes encodings. We consider the propagation rule originally introduced in [51] as our base model:

$$H^{l+1} = \sigma\big(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{l}W^{l}\big), \tag{2.1}$$

where $\hat{A} = A + I_N$ is the summation of the undirected graph G's adjacency matrix with binary entries A and self-connection $l_N \in \mathbb{R}^N$, $l_N \in \mathbb{R}^N$ is a identity matrix, D is a degree matrix with $D_{ii} = \sum_j A$, $W^l \in \mathbb{R}^{N \times C^l}$ is a matrix of trainable weights at depth l, σ is an activation function, and H^l is the encoding of all nodes in the graph at depth l ($H^0 = X$).

This layer-wise propagation rule can be rewritten in the following vector form:

$$h_{v_i}^{l+1} = \sigma\big(\sum_j \frac{h_{v_j}^{l}}{c_{ij}}W^{l} + \frac{h_{v_i}^{l}}{c_{ii}}W^{l}\big). \tag{2.2}$$

Here, j indexes neighboring nodes of v_i, normalization factor $\frac{1}{c_{ij}}$ is an entry located at the ith row, jth column of $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$.

The propagation rule represented by Equation 2.1 is a first-order approximation of spectral convolution on a graph. It provides two advantages when used to analyze graphs: first, it enables to aggregate l^{th} order neighborhood of a central node during the encoding process; second, it prevents us from prohibitively expensive eigendecomposition of the graph Laplacian compared with spectral convolution models [51]. Those properties offer us a computational efficient approach to learn the interactions between vehicles that are not directly connected in the graph.

GCN utilizes a pre-defined adjacency matrix A whose entries are binary to obtain normalization factors $c_{i,j} = \frac{1}{\alpha_{i,j}}$. In contrast, GAT, which applies *self-attention*, learns to generate the normalization factors for neighbouring nodes rather than resorting to weights in adjacency matrix A:

$$\alpha_{ij} = \frac{\exp\Big(\text{LeakyReLU}\big(BW_a h_i + WW_a h_j)\big)\Big)}{\sum_{k \in N_i}\exp\Big(\text{LeakyReLU}\big(BW_a h_i + WW_a h_k)\big)\Big)}, \tag{2.3}$$

where i indexes the central node, j indexes the surrounding nodes, W_a, B and W are weights in *self-attention* with B applied to central nodes and W applied to

surrounding nodes, LeakyReLU is an activation function, and α_{ij} is equivalent to normalization factor $\frac{1}{c_{ij}}$ mentioned in previous equations. Following the practice in [23], we utilized different layer weights, B and W, to attend to central and neighbouring nodes respectively. *Self-attention* weights W_a, B, and W are updated such that GAT learns how to distribute α_{ij}.

2.4 Micro-level Highway Traffic Graph

In the car-following literature, models are proposed to capture the interactions among the ego vehicle and its neighbors despite the definition of neighbors differs in the spectrum of research. In this study, we use a flexible data structure, graph, to model the relationship between the ego vehicle and its neighbors, which enable us to include a flexible number of neighbors depending on the traffic. As shown in Figure 2.1, for a vehicle pair (v_i, v_j) where $v_i \in V$ and $v_j \in V$, the edge (v_i, v_j) is connected if and only if:

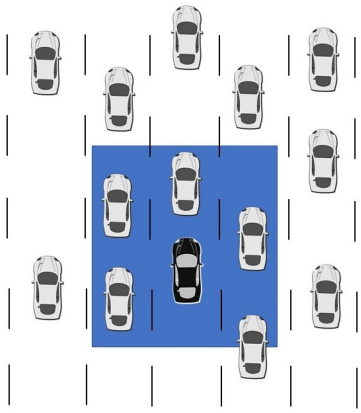
- vehicle v_j and v_i appear at the same frame; and
- vehicle v_j is less than one lane away from vehicle v_i at the current frame (vehicle v_j should be on the same lane with vehicle v_i or on vehicle i's left, right lanes); and
- the absolute value difference of vehicle v_j's y-coordinate and vehicle v_i's y-coordinate is less than the designated value τ at the current frame.

Note that there is no fixed limit on the number of neighbors; all vehicles within an ego-vehicle's designated distance τ are its neighbor vehicles. In NGSIM I-80 dataset, the traffic of the study area changes frequently. The traffic, hence, is updated at the same frequency as data was collected in the original dataset. Figure 2.1(c), and Figure 2.1(d) present statistics regarding graphs.

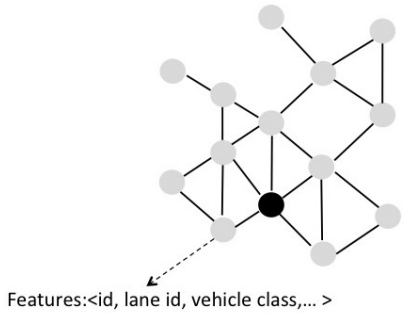
In this work, we adopt the features used in [57]. For a vehicle node in the graph at frame t, its feature vector includes the following elements: vehicle lane id l_t, vehicle class id c, vehicle velocity v_t, vehicle acceleration a_t, relative distance from 3 nearest front neighbor vehicles $\{d_{f_1}, d_{f_2}, d_{f_3}\}$ (pad τ if the number of front neighbors is smaller than 3), and negative relative distance from 3 nearest rear neighbor vehicles $\{-d_{r_1}, -d_{r_2}, -d_{r_3}\}$ (pad $-\tau$ if the number of rear neighbors is less than 3).

2.5 GCNs for Traffic

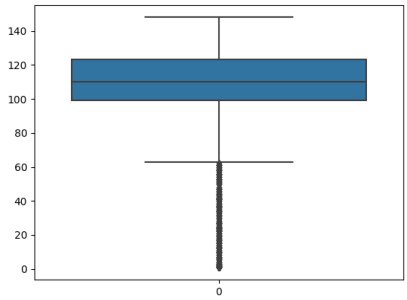
The propagation rule represented by Equation 2.1 is a first-order approximation of spectral convolution on a graph. It provides two advantages when used to analyze



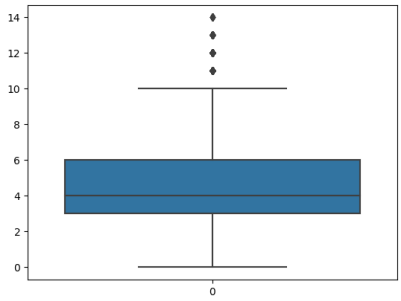
(a) An ego-vehicle considers vehicles only within 1 lanes away as potential neighbor vehicles. A potential neighbor vehicle will be deemed as ego-vehicle's neighbor if and only if the absolute value of their headway distance is smaller than τ



(b) A graph is constructed by connecting every vehicle with their neighbor vehicles. Graph nodes share the same feature fields



(c) A box plot of the number of nodes in graphs. This depicts the size of traffic graphs



(d) A box plot of the number of neighbours possessed by every vehicle node in graphs. This indicates the number of edges per each node possessed in traffic graphs

Figure 2.1: Mapping from real world traffic to traffic graph

graphs: first, it enables to aggregate l^{th} order neighborhood of a central node during the encoding process; second, it prevents us from prohibitively expensive eigendecomposition of the graph Laplacian compared with spectral convolution

models [51]. Those properties offer us a computational efficient approach to learn the interactions between vehicles that are not directly connected in the graph.

2.5.1 Ego-discriminated GCN

(EGCN): During the implementation of the base model, we find that self-connection affects the performance of the system, an observation that leads to our adaptation of the base model. Self-connection was used to alleviate the problem of vanishing/exploding gradients in GCNs [51]. However, this method applies the same weight W^l to both the central node and its surrounding nodes. In our experiments, we find it is beneficial to remove the self-connection and apply different layer weights to discriminate the central node from its surrounding node. This leaves us with the ego-discriminated propagation rule, which can be represented as follows:

$$H^{l+1} = \sigma\Big(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^lW^l + I_NH^lB^l\Big), \tag{2.4}$$

where $l_N \in \mathbb{R}^N$ is an identity matrix, $B^l \in \mathbb{R}^{N\times C^l}$ are trainable weights at depth l for central nodes. The corresponding vector form is given in the following expression:

$$h_{v_i}^{l+1} = \sigma\Big(\sum_j \frac{h_{v_j}^l}{c_{ij}}W^l + \frac{h_{v_i}^l}{c_{ii}}B^l\Big). \tag{2.5}$$

2.5.2 Distance-Aware Graph Convolution Network

: For the models mentioned in the previous section, their adjacency matrices $\hat{A}$ and A only denote whether a pair of vehicles is close or not, but they do not describe the degree of closeness. Based on our empirical driving experience–the closer our neighbor vehicle is, the more attention we will pay to it–we use absolute inverse relative distances as entries for our adjacency matrix $\tilde{A}$ to differentiate the degree of closeness between vehicles. Therefore, we introduce the following distance-aware layer-wise propagation rule in our multi-layer GCN (DGCN):

$$H^{l+1} = \sigma\Big(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^lW^l + I_NH^lB^l\Big). \tag{2.6}$$

Here, $\tilde{A}$ is an adjacency matrix with $\tilde{A}_{ij} = \frac{1}{|y_{v_i} - y_{v_j}|}$ where y_i represents vehicle v_i's y-coordinate. $\tilde{D}$ is a degree matrix with $\tilde{D}_{ii} = \sum_j A_{ij}$. In this propagation rule, $\tilde{A}$'s entries denote the degree of closeness between vehicles. To stablize gradients during training, we discretize the degree of closeness into three levels: $1, 2$, and 3,

which represent far away, medium close and very close, respectively. Equation 2.6 can also be rewritten in the following vector form:

$$h_{v_i}^{l+1} = \sigma\Big(\sum_j \frac{h_{v_j}^l}{\tilde{c}_{ij}} W^l + h_{v_i}^l B^l\Big), \tag{2.7}$$

where $\tilde{c}_{ij}$ is an entry located at ith row and jth column of $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$.

2.5.3 Long-short Term Memory

The Long-short Term Memory (LSTM), which is a type of RNN that can remember the past state, is used to incorporate the temporal information in predicting the ego-vehicle's future acceleration. The LSTM unit that is used in this study can be written as follows:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.8}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.9}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.10}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.11}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{2.12}$$
$$h_t = o_t \circ \sigma_c(c_t) \tag{2.13}$$

. Here, W, and U denote learnable weights and b represents bias, α_g is a sigmoid activation, α_c is a hyperbolic tangent activation, x_t is the input for the currrent timestep t, h_{t-1} and h_t denote the hidden state for the previous and the current timestep respectively, f_t is the output of the forget gate, i_t is the output of the input gate, o_t is the output of the output gate, $\tilde{c}_t$ is used to calculate the cell state c_t.

2.5.4 Gaussian Mixture Model

In this work, we aim to predict human driver's acceleration distribution given the current traffic state. Hence the output of our network model is Gaussian mixture model (GMM) parameters that characterize the future acceleration distribution. This *mixture density network* (MDN) is first proposed by Bishop [9] and been successfully applied in speech recognition and other fields [75]. For a K-component GMM, the probability of the predicted acceleration follows this equation:

$$p(a) = \sum_{i=1}^{K} w_i \mathcal{N}(a|\mu_i, \sigma_i^2), \tag{2.14}$$

where w_i, μ_i, and σ_i are the weight, mean, standard deviation of the ith mixture component respectively.

2.6 Dataset and Training

Dataset: The NGSIM I-80 dataset contains detailed vehicle trajectory data collected using synchronized digital video cameras on eastbound I-80 in Emeryville, CA. This dataset provides precise positions, velocities and other vehicle information over three 15-minute periods at 10 Hz. The study area covers approximately 500 meters in length and consists of six freeway lanes, including a high-occupancy lane and an on-ramp lane. We use the NGSIM I-80 reconstructed dataset, which contains vehicles position, velocity, acceleration from 4:00 p.m. to 4:15 p.m., because it corrects errors such as extreme acceleration, and inconsistent vehicle IDs [65] [66]. We split the data into training sets and testing sets by a ratio of 4 to 1.

Data Preparation: Both training set and testing set are divided into 12-second segments (120 frames). The first 2-second segments (20 frames) are used to initialize the internal state of LSTM networks. Since the aim of the research is to predict driving acceleration using GCNs, we need to prepare traffic graphs from the raw data.
Training: All models are trained to output predicted parameters for distributions

Table 2.1: Model Configuration

Model	layer 1	layer 2	layer 3	LSTM	clip norm	adjacency type
Fully-connected	128	256	128	no	5	/
GCN base	128	256	128	no	5	binary
GAT	128	256	128	no	5	binary
EGCN	128	256	128	no	5	binary
DGCN	128	256	128	no	5	inverse distance
LSTM	128	256	128	yes	5	/
GCN with LSTM	128	256	128	yes	5	binary
GAT with LSTM	128	256	128	yes	5	binary
EGCN with LSTM	128	256	128	yes	5	binary
DGCN with LSTM	128	256	128	yes	5	inverse distance

over future acceleration values. Note that every model in this work shares the same hyperparameters because we aim to compare the effectiveness of GNN and LSTM on improving model performance in the task of driver behavior prediction. We set $\tau = 20$ feet, empirically.

Model structures are shown in Table 2.1. Each model consists of 3 hidden layers and a 30-component MDN layer. Layer 1 applies Relu activation while other layers do not use any activation. Layer 1 and layer 2 are followed by batch normalization. Batch normalization is a mechanism to address the problem of *internal covariate shift*. It has been reported that adding batch normalization to

state-of-the-art image classification networks yields higher classification accuracy compared with the original networks [42]. The performance of our models is also found to improve when batch normalization is applied. Layer 3 is either an FC layer or an LSTM layer. The final 30-component MDN layer follows layer 3 and has an output size of 90, which corresponds to a 30-component GMM's parameters.

All models are trained for 5 epochs. During training, the models are optimized by the Adam optimizer with a learning rate of 1×10^{-3} [50]. A dropout of 10 percent is applied to help prevent overfitting. Gradient norm clipping is also used to deal with gradient vanishing and gradient explosion [70]. All networks are implemented in TensorFlow [2] based on Kpif's *GCN* package [51] and Veličković's *GAT* package [88].

2.7 Evaluation

Once trained, each model is used to generate simulated trajectories. For every trajectory in the test set, the first 2-second segments (20 frames) of true data are used to initialize LSTM's internal state. In the following 10 seconds, ego-vehicle's velocity and position can be updated by assuming the following equations:

$$\begin{aligned} v(t+\delta t) &= v(t) + a(t+\delta t) \times \delta t \\ y(t+\delta t) &= y(t) + v(t+\delta t) \times \delta t, \end{aligned} \tag{2.15}$$

where v is ego-vehicle's velocity, y is ego-vehicle's Y-coordinate and a is vehicle's acceleration. The graph and node features are updated by propagating other vehicles' true trajectory data and ego-vehicle's simulated trajectory. Following the practice in [67], we evaluate the quality of simulated trajectories by the following metrics:

- *Root Mean Squared Error (RMSE):* We use root mean squared error to evaluate the discrepancy of speed values between simulated trajectories and true trajectories at designated horizons for a given ego-vehicle:

$$RMSE_{velocity} = \sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(v_H^i - \hat{v}_H^{i,j})^2}, \tag{2.16}$$

  where m is the number of true trajectories, $n = 20$ is the number of simulated trajectories per true trajectory, v_H^i is the velocity of ith true trajectory at horizon H, $\hat{v}_H^{i,j}$ is the value in jth simulated trajectory at time horizon H. Similarly, we also use root mean squared error to evaluate the displacement in Y-coordinate at 10 second horizon between simulated trajectories and true

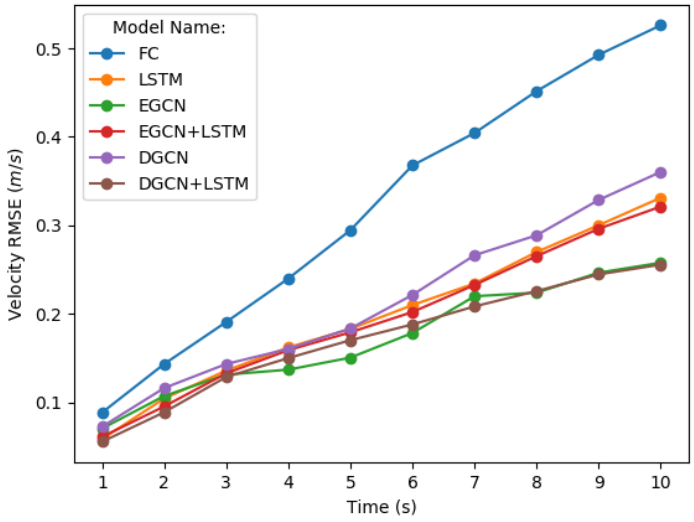


Figure 2.2: RMSE results for all models

trajectories:

$$RMSE_Y = \sqrt{\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{10}^{i} - \hat{y}_{10}^{i,j})^2}, \tag{2.17}$$

where y_{10}^{i} is the Y-coordinate of ith true trajectory at 10 second, $\hat{y}_{10}^{i,j}$ is the simulated Y-coordinate value for sample j in the ith trajectory at 10 second horizon.

Figure 2.2 shows the velocity RMSE for the top 6 models over prediction horizons between 1 and 10 seconds. Models with original GCN [51] and GAT [88] are not included because of their bad performance in generating predicted trajectories. In general, the velocity RMSE accumulates over the time horizon. Our adapted GCN models outperform non-GCN models. For non-GCN models, LSTM outperforms the fully-connected model because LSTM is able to access past information. For GCN models, EGCN model and DGCN with LSTM outperform other GCN models.

The Y-coordinate RMSE column in Table 2.2 denotes the displacement in Y-coordinate between simulated trajectories and their corresponding true trajectories. EGCN model outperforms other models. Velocity RMSE at 10 second horizon reveals the discrepancy of speed between simulated trajectories and the ground truth. DGCN with LSTM outperforms other models in

this metric.

Table 2.2: RMSE Analysis

Model	Y RMSE @ 10 s (m)	Velocity RMSE @ 10 s (m/s)
Fully-connected (FC)	2.89	0.526
GCN base	3.52	0.622
GAT	4.13	0.688
EGCN	**1.40**	**0.258**
DGCN	1.91	0.360
LSTM	**1.61**	0.331
GCN with LSTM	3.40	0.653
GAT with LSTM	4.09	0.728
EGCN with LSTM	1.86	0.321
DGCN with LSTM	1.63	**0.256**

- *Negative Headway Distance Occurrence:* This metric is used to evaluate models' robustness. It records the occurrences of unrealistic states led by models' poor decision making. Two types of negative headway distances are considered: (1) ego-vehicle's negative headway distance representing collisions with the front vehicle; and (2) following vehicle's negative headway distance denoting collisions between the ego-vehicle and its following vehicle. A robust model will have minimal negative headway distance occurrence.

  Table 2.3 shows the number of negative headway occurrences over number of simulated trajectories for all models. Consistent with RMSE analysis, the results from Table 2.3 demonstrates that original GCN models often produce poor acceleration predictions which lead to unrealistic states. EGCN model and DGCN with LSTM model are robust because there are no unrealistic states occurring in their simulated trajectories.

- *Jerk Sign Inversions:* We use the number of jerk sign inversions per trajectory to evaluate the similarity between the smoothness of the true and simulated trajectories. This metric is used to quantify oscillations in model's acceleration predictions.

  Simulated trajectories of most of models have slightly higher jerk sign inversions than the true trajectories while the LSTM baseline model is not able to generate smooth trajectories. In addition, jerk sign inversions, combined with previous metrics, indicate that the trajectories generated by GAT with LSTM model fail to react against the changes of the ego-vehicle's surrounding traffic.

Table 2.3: Jerk Sign Inversions Per Trajectory

Model	Jerk Sign Inversions	Negative Headway Occurrence Rate
Fully-connected (FC)	7.5	0.08
GCN base	7.5	0.17
GAT	**5.9**	0.27
EGCN	7.5	**0**
DGCN	7.3	0.03
LSTM	13.7	0.02
GCN with LSTM	**6.7**	0.17
GAT with LSTM	0.0	0.27
EGCN with LSTM	9.5	0.01
DGCN with LSTM	7.3	**0**
True trajectory	6.3	/

Figure 2.3 shows the sample simulated trajectories by models, including adapted GCN models and non-GCN models. It can be seen that non-GCN models predict poorly if the ground truth trajectory includes a long period of acceleration values that are very close to zero while GCN models is able to generate smooth trajectories close to the ground truth. In addition, non-GCN models are prone to predict extreme acceleration values, which is compensated by oscillation of acceleration values.

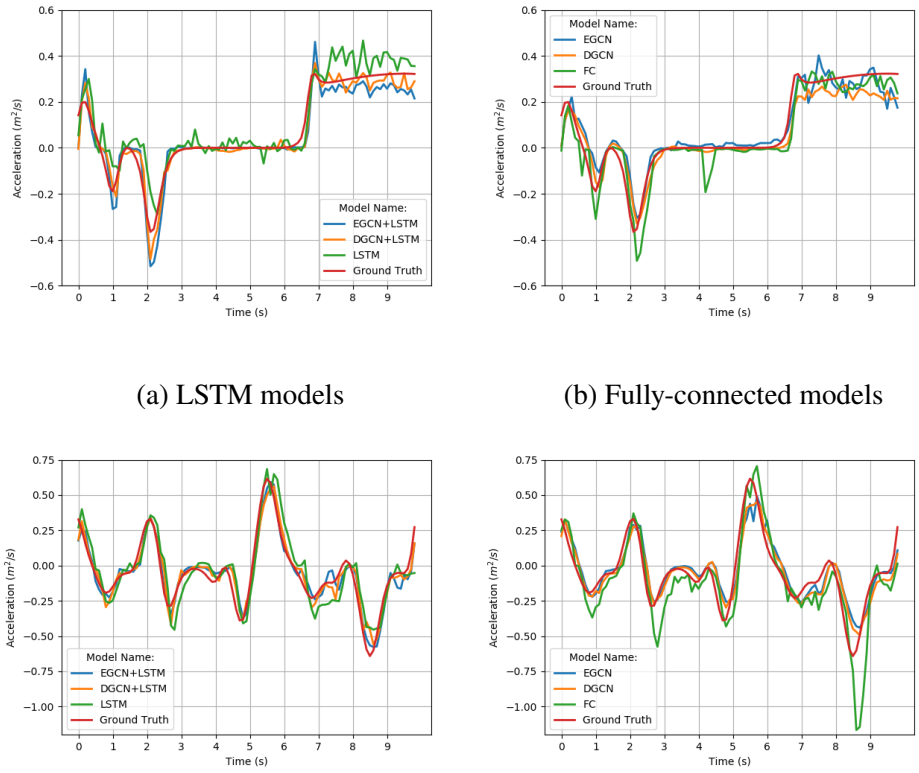
2.8 Discussion

Our experiments are designed to answer the following research questions:

- Does GCN improve model performance and are our adaptations to GCN beneficial?
- Does including LSTM increase prediction quality?
- Why do GAT models fail to generate realistic trajectories?

First, we discover that we improve GCN's performance when we delete self-connections and apply different weights to the central nodes and their surrounding nodes. For GCN base model, we reduced velocity RMSE by 58.5% at 10 seconds horizon and negative headway occurrence by 17% during simulation. For GCN with LSTM model, we reduced its 10 seconds horizon velocity RMSE by 50.8% and negative headway occurrence by 15%.

Second, our experiments demonstrated that GCNs improve model performance. GCN models are able to generate smooth and robust trajectories close to the ground



(a) LSTM models

(b) Fully-connected models

(c) LSTM models

(d) Fully-connected models

Figure 2.3: Simulated Trajectories For All Models (Orignial GCN and GAT models are excluded for their bad performance)

truth. For both LSTM and fully-connected models, the non-GCN baseline model is outperformed by its GCN couterparts, in general. Note that, in the experiments, our GCN models and non-GCN models share the same number of hidden layers and the same number of neurons in each hidden layer. Compared with non-GCN fully-connected model, our EGCN model reduced the negative headway occurrence rate from 0.08 to 0, 10 seconds horizon velocity RMSE by 59.6%. Compared with non-GCN LSTM, our DGCN with LSTM reduced the negative headway occurrence rate from 0.02 to 0, jerk sign inversions from 13.7 to 7.3 and 10 seconds horizon velocity RMSE by 22.7%. This trend can also be observed in Figure 2.3. The multi-layer GCN's ability to capture multitudes of interactions between vehicles hierarchically improves model's prediction quality in terms of our evaluation metrics.

In general, we find that adding LSTM structure improves model prediction

quality. Among all models, the best model is DGCN with LSTM. During simulation, this model is able to generate robust and smooth driving trajectories with 0 negative headway, 7.3 jerk sign inversions and 0.256 for 10-second horizon velocity RMSE.

GATs utilize *self-attention* to assign attentional weights α_{ij} for neighbouring node j. The attentional weights α_{ij} indicate the relationship between the central node and its surrounding nodes. Following [7], we investigated α_{ij} to understand why GAT models fail in our experiment. The investigation shows that the relational kernel in the baseline GAT models fails to learn the relationships between central nodes and their surrounding nodes. From the sample in the vehicle 829 at the frame 2373, the relational kernel in the second GAT layer of the GAT model assigns the same weights to every node: vehicle 829 initially has two neighbours, 818 and 835. The attentional weights for each node, including the central node 829, is 0.333. Later in the trajectory, another vehicle 795 approaches the ego-vehicle 829 and the attentional weights assigned to all 4 nodes are 0.25.

2.9 Conclusion

In this study, we propose the use of graphs defined by the spatial relationships between vehicles, to model traffic. We further build GCN models, operating on graphs, to predict future acceleration distributions. We propose two GCN models adapted from the state-of-art GCN and studied the effectiveness of LSTM architectures in our prediction models. Our resulting frameworks outperform others on the task of acceleration prediction.

While our proposed methods have been shown to improve prediction performance, much work remains to be done. This work can be extended to prediction in two dimensions, which is an important problem in autonomous driving. At the same time, it will be interesting to evaluate different graph construction strategies, such as strategies that include multiple layers of relationships.

2.10 Related Work

The task of modeling driving behavior consists of modeling car-following behaviors and lane-changing behaviors. In our work, we focus on augmenting the car-following model.

Car-following models capture the interaction between the ego-vehicle and the vehicles directly adjacent on the microscopic level of the traffic. Based on the number of interactions captured, models can be categorized as being either a single-lane or multiple-lanes.

A single-lane model focuses on the interactions between vehicles in a single lane. This model considers up to two kinds of interactions: namely, the ego-vehicle with its leading vehicle, and the ego-vehicle with its following vehicle. Many traditional fixed-form models fall into this category, including the Gazis-Herman Rothery model [15], the collision avoidance model [52], linear models [36], psycho-physical models [62], and fuzzy logic-based models [47].

Some recent general driving models have moved away from making assumptions about drivers. Lefèvre et al. compare the performance of feed-forward mixture density network against traditional baselines [56]. Their empirical tests suggest that the proposed method is able to achieve performance comparable to the baselines. Morton et al. study the effectiveness of LSTM in predicting driving behavior on highways. They reveal that the LSTM's ability to remember historic states of the ego-vehicle appears to be the key to achieving the state-of-art performance [67].

More recently, multiple-lane models that consider more interactions, coupled with neural networks, have been introduced in the literature. Kim et al. propose a framework based on LSTM to predict vehicle's future position over the occupancy grid [48]. Altche et al. use LSTM that predicts traffic using as input state information on the ego-vehicle states and up to 9 of its neighbors. The model is trained and evaluated on the NGSIM 101 dataset which has trajectories from more than 6000 individual drivers [4].

Diehl et al. [23] used GNNs for vehicle coordinates prediction and demonstrated that GAT models outperform other baselines. Note that our method differs from that work in three main ways. First, we are interested in generating vehicle trajectories. Hence, our models are structured to predict 0.1-second future acceleration, which can be propagated to vehicle trajectories of any length with velocity, acceleration, and coordinates information. Diehl et al. aim to predict the 5-second-later coordinates of a vehicle, which contains limited information for the construction of realistic vehicle trajectories. Second, our framework allows for including an arbitrary number of neighbors. The models of Diehl et al., by contrast, consider only up to 8 neighbors, which might result in ignoring important information about the state of traffic around ego-vehicles. Third, we believe that information of ego-vehicle's past states affects future actions. Diehl et al. does not consider RNN structure, whereas we include this structure because it acts to memorize the past states of a vehicle. Furthermore, we analyze the performance of GNNs with and without RNNs.

Chapter 3

Value-Decomposition Multi-Agent Reinforcement Learning

3.1 Background

Breakthroughs in Q-learning have been made using joint action-value factorization techniques. *Value-decomposition networks* (VDN) represent joint action-value as a summation of local action-value conditioned on individual agents' local observation history [82]. In [73], a more general case of VDN is proposed using a mixing network that approximates a broader class of monotonic functions to represent joint action-values called QMIX. In [79], a more complex factorization framework three modules, called QTRAN, is introduced and shown to have good performance on a range of cooperative tasks. While QMIX reports the best performance on the StarCraft micromanagement testbed [76], we find that QMIX, in some StarCraft II compositions, has issues learning good policies that can consistently defeat enemies when using the A2C training paradigm [63], which was originally introduced to enable algorithms to be executed efficiently.

On the other hand, on-policy actor-critic methods, such as *counterfactual multi-agent* (COMA) [29], can leverage the A2C framework to improve training efficiency at the cost of performance. [76] point out that there is a performance gap between the state-of-the-art actor-critic method, COMA, and QMIX on the StarCraft II micromanagement testbed.

To bridge the gap between multi-agent Q-learning and multi-agent actor-critic methods, as well as offer a reasonable trade-off between training efficiency and algorithm performance, we propose a novel actor-critic framework called value-decomposition actor-critic (VDAC). Let $V^a, \forall a \in \{1, \ldots, n\}$ denote the local state value that is conditioned on agent a's local observation, and let V_{tot} denote the global state-value that is conditioned on the true state of the environment. VDAC

takes an actor-critic approach but adds local critics, which share the same network with the actors and estimate the local state values V^a. The central critic learns the global state value V_{tot}. The policy is trained by following a gradient dependent on the central critic. Further, we examines two approaches for calculating V_{tot}.

3.2 Executive Summary

VDAC is based on three main ideas. First, unlike QMIX, VDAC is compatible with a A2C training framework that enables game experience to be sampled efficiently. This is due to the fact that multiple games are rolled out independently during training. Second, similar to QMIX, VDAC enforces the following relationship between local state-values V^a and the global state-value V_{tot}:

$$\frac{\partial V_{tot}}{\partial V^a} \geq 0, \quad \forall a \in \{1, \ldots, n\}. \tag{3.1}$$

This idea is related to *difference rewards* [95], in which each agent learns from a shaped reward that compares the global reward to the reward received when that agent's action is replaced with a default action. *Difference rewards* require that any action that improves an agent's local reward also improves the global reward, which implies the monotonic relationship between shaped local rewards and the global reward. While COMA (also inspired by *difference rewards*) focuses on customizing shaped rewards r^a from the global reward r_{tot} in a pairwise fashion $r^a = f(r_{tot})$, VDAC represents the global reward by all agents' shaped rewards $r_{tot} = f(r^1, ..., r^n)$. Third, VDAC is trained by following a rather simple policy gradient that is calculated from a temporal-difference (TD) advantage. We theoretically demonstrate that the proposed method is able to converge to a local optimum by following this policy gradient. Despite the fact that TD advantage policy gradients and COMA gradients are both unbiased estimates of a vanilla multi-agent policy gradients, our empirical study favors TD advantage policy gradients over COMA policy gradients.

This study strives to answer the following research questions:

- **Research question 1**: Is the TD advantage gradient sufficient to optimize multi-agent actor-critics when compared to a COMA gradient?
- **Research question 2**: Does applying state-value factorization improve the performance of actor-critics?
- **Research question 3**: Does VDAC provide a reasonable trade-off between training efficiency and algorithm performance when compared to QMIX?

- **Research question 4**: What are the factors that contribute to the performance of the proposed VDAC?

3.3 Preliminaries and Technical Background

Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs): Consider a fully cooperative multi-agent task with n agents. Each agent identified by $a \in A \equiv \{1, \ldots, n\}$ takes an action $u^a \in U$ simultaneously at every timestep, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^a, \forall a \in \{1, \ldots, n\}$. The environment has a true state $s \in S$, a transition probability function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to S$, and a global reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$. In the partial observation setting, each agent draws an observations $z \in Z$ from the observation function $O(S, A) : S \times A \to Z$. Each agent conditions a stochastic policy $\pi(u^a|\tau^a) : T \times U \to [0, 1]$ on its observation-action history $\tau^a \in T \equiv Z \times U$. Throughout this paper, quantities in bold represent joint quantities over agents, and bold quantities with the superscript $-a$ denote joint quantities over agents other than a given agent a. MARL agents aim to maximize the discounted return $R_t = \sum_{l=1}^{\infty} \gamma^l r_{t+l}$. The joint value function $V^\pi(s_t) = \mathbb{E}[R_t|s_t = s]$ is the expected return for following the joint policy π from state s. The value-action function $Q^\pi(s, \mathbf{u}) = \mathbb{E}[R_t|s_t = s, \mathbf{u}]$ defines the expected return for selecting joint action $\mathbf{u}$ in state s and following the joint policy π.

Single-Agent Policy Gradient Algorithms: Policy gradient methods adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim p^\pi, u\sim\pi}[R(s, u)]$ by taking steps in the direction of $\nabla J(\theta)$. The gradient with respect to the policy parameters is

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s, u)] \tag{3.2}$$

, where p^π is the state transition by following policy π, and $Q_\pi(s, u)$ is an action-value.

To reduce variations in gradient estimates, a baseline b is introduced. In actor-critic approaches [53], an actor is trained by following gradients that are dependent on the critic. This yields the advantage function $A(s_t, u_t) = Q(s_t, u_t) - b(s_t)$, where $b(s_t)$ is the baseline ($V(s_t)$ or another constant is commonly used as the baseline). TD error $r_t + \gamma V(s_{t+1}) - V(s_t)$, which is an unbiased estimate of $Q(s_t, u_t)$, is a common choice for advantage functions. In practice, a TD error that utilizes an n-step return $\sum_{i=0}^{k-1} \gamma^i r_t + \gamma^k V(s_{t+k}) - V(s_t)$ yields good performance [63].

Multi-Agent Policy Gradient (MAPG) Algorithms: Multi-agent policy gradient methods are extensions of policy gradient algorithms with a policy $\pi_{\theta_a}(u^a|\tau^{`a}), a \in$

$\{1, \cdots, n\}$. Perhaps the simplest multi-agent gradient can be written as:

$$\nabla_\theta J = \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi_\theta(u^a|\tau^a) Q_\pi(s, \mathbf{u}) \right]. \tag{3.3}$$

Taking into the fact agent's policies are independent, the joint policy π can be written as a product of independent policies:

$$\pi(\mathbf{u}|s) = \prod_a \pi^a(u^a|\tau^a) \tag{3.4}$$

And the vanilla multi-agent gradient can be represented in a single-agent RL fashion:

$$\begin{aligned} \nabla_\theta J &= \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi_\theta(u^a|\tau^a) Q_\pi(s, \mathbf{u}) \right] \\ &= \mathbb{E}_\pi \left[\nabla_\theta \log \prod_a \pi_\theta(u^a|\tau^a) Q_\pi(s, \mathbf{u}) \right] \\ &= \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(\mathbf{u}|s) Q_\pi(s, \mathbf{u}) \right] \end{aligned} \tag{3.5}$$

Compared with policy gradient methods, MAPG faces the issues of high variance gradient estimates [60] and credit assignment [29]. Multi-agent policy gradients in the current literature often take advantage of CTDE by using a central critic to obtain extra state information s, and avoid using the vanilla multi-agent policy gradients (Equation 3.3) due to high variance. For instance, [60] utilize a central critic to estimate $Q(s, (a_1, \ldots, a_n))$ and optimize parameters in actors by following a multi-agent DDPG gradient, which is derived from Equation 3.3:

$$\nabla_{\theta_a} J = \mathbb{E}_\pi [\nabla_{\theta_a} \pi(u^a|o^a) \nabla_{u^a} Q_{u^a}(s, \mathbf{u})|_{u^a = \pi(o^a)}]. \tag{3.6}$$

Unlike most actor-critic frameworks, [29] claim to solve the credit assignment issue by applying the following counterfactual policy gradients:

$$\nabla_\theta J = \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) A^a(s, \mathbf{u}) \right], \tag{3.7}$$

where $A^a(s, \mathbf{u}) = Q_\pi(s, \mathbf{u}) - \sum_{u^a} \pi_\theta(u^a|\tau^a) Q_\pi^a(s, (\mathbf{u}^{-a}, u^a))$ is the counterfactual advantage for agent a. Note that [29] argue that the COMA gradients provide agents with tailored gradients, thus achieving credit assignment. At the same time, they also prove that COMA is a variance reduction technique.

Table 3.1: Actor-Critics studied.

Algorithm	Central Critic	Value Decomposition	Policy Gradients
IAC [29]	No	-	TD advantage
VDAC-sum	Yes	Linear	TD advantage
VDAC-mix	Yes	Non-linear	TD advantage
Naive Critic	Yes	-	TD advantage
COMA [29]	Yes	-	COMA advantage

3.4 Methods

In addition to the previously outlined research questions, our goal in this work is to derive RL algorithms under the following constraints: (1) the learned policies are conditioned on agents' local action-observation histories (the environment is modeled as Dec-POMDP), (2) a model of the environment dynamics is unknown (i.e. the proposed framework is task-free and model-free), (3) communication is not allowed between agents (i.e. we do not assume a differentiable communication channel such as [21]), and (4) the framework should enable parameter sharing among agents (namely, we do not train different models for each agent as is done in [85]). A method that met the above criteria would constitute a general-purpose multi-agent learning algorithm that could be applied to a range of cooperative environments, with or without communication between agents. Hence, the following methods are proposed.

3.4.1 Naive Central Critic Method

A naive central critic (naive critic) is proposed to answer the first research question: is a simple policy gradient sufficient to optimize multi-agent actor-critic methods. As shown in Fig 3.1, naive critic's central critic shares a similar structure with COMA's critic. It takes (s_t, u_{t-1}) as the input and outputs $V(s)$. Actors follow a rather simple policy gradient, a TD advantage policy gradient that is common in the RL literature given by:

$$\nabla_\theta J = \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) \big(Q(s, \mathbf{u}) - V(s) \big) \right], \tag{3.8}$$

where $Q(s, \mathbf{u}) = r + \gamma V(s')$. In the next section, we will demonstrate that policy gradients taking the form of Equation 4.9 under our proposed actor-critic frameworks

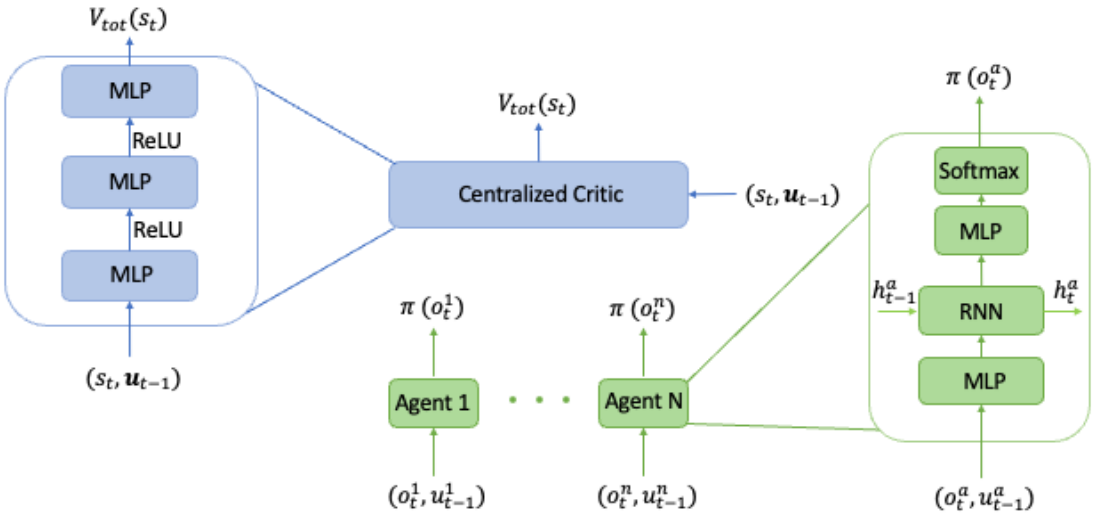


Figure 3.1: Naive Critic

are also unbiased estimates of the naive multi-agent policy gradients. Algorithm 1 describes the naive critic.

3.4.2 Value Decomposition Actor-Critic

Difference rewards enable agents to learn from a shaped reward $D^a = r(s, \mathbf{u}) - r(s, (\mathbf{u}^{-a}, c^a))$ that is defined by a reward change incurred by replacing the original action u^a with a default action c^a. Any action taken by agent a that improves D^a also improves the global reward $r(s, \mathbf{u})$ since the second term in the difference reward equation does not depend on u^a. Therefore, the global reward $r(s, \mathbf{u})$ is monotonically increasing with D^a. Inspired by *difference rewards*, we propose to decompose state value $V_{tot}(s)$ into local states $V^a(o^a)$ such that the following relationship holds:

$$\frac{\partial V_{tot}}{\partial V^a} \geq 0, \quad \forall a \in \{1, \ldots, n\}. \tag{3.9}$$

With Equation 3.9 enforced, given that the other agents stay at the same local states by taking $\mathbf{u}^{-a}$, any action u^a that leads agent a to a local state o^a with a higher value will also improve the global state value V_{tot}.

Two variants of value-decomposition that satisfy Equation 3.9, VDAC-sum and VDAC-mix, are studied.

Algorithm 1 Naive Central Critic

Initialize critic θ^c, target critic $\hat{\theta}^c$, and actor θ
for each training episode e **do**
  Empty buffer
  for $e_c = 1$ to $\frac{\text{BatchSize}}{n}$ **do**
    $t = 0, h_o^a$ for each agent a
    while game not terminated **and** $t < T$ **do**
      $t = t + 1$
      for each agent a **do**
        $h_t^a, \pi_t^a = \text{Actor}(o_t^a, h_{t-1}^a, u_{t-1}^a, a; \theta)$
        Sample action u_t^a from π_t^a
      end for
      Get reward r_t and next state s_{t+1}
    end while
    add experience to buffer
  end for
  Collate episodes in buffer into single batch
  for $t = 1$ to T **do**
    Batch unroll RNN using states, actions and reward
    Calculate y_t and A_t using $\hat{\theta}^c$
  end for
  for $t = T$ down to 1 **do**
    Calculate gradient wrt $\theta^c : \Delta\theta^c \leftarrow \nabla_{\theta^c}\big(y_t - V(s_t, \mathbf{u}_{t-1}; \theta^c)\big)^2$
    Update critic $\theta^c \leftarrow \theta^c - \alpha\Delta\theta^c$
    Every C steps update target critic $\hat{\theta}^c \leftarrow \theta^c$
  end for
  for $t = 1$ down to T **do**
    Accumulate gradient wrt $\theta : \Delta\theta \leftarrow \Delta\theta + \nabla_\theta \log \pi(u_t^a | o_t^a) A_t$
  end for
  Update actor weights $\theta = \theta + \alpha\Delta\theta$
end for

VDAC-sum

In VDAC-sum, the total state value $V_{tot}(s)$ is a summation of local state values $V^a(o^a)$: $V_{tot}(s) = \sum_a V^a(o^a)$. This linear representation is sufficient to satisfy Equation 3.9. VDAC-sum's structure is shown in Figure 3.2. Note that the actor outputs both $\pi_\theta(o^a)$ and $V_{\theta_v}(o^a)$. This is done by sharing non-output layers between distributed critics and actors. In this paper, θ_v denotes the distributed critics'

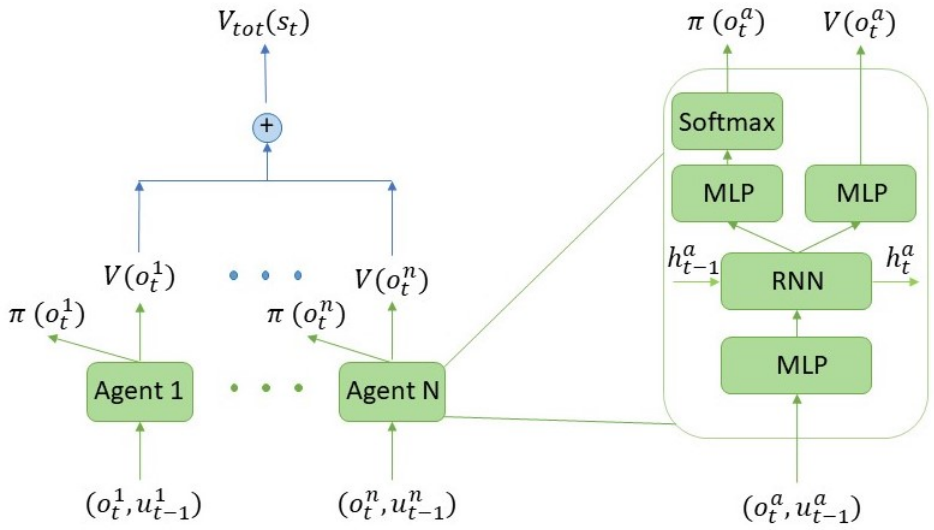


Figure 3.2: VDAC-sum

parameters and θ denotes the actors' parameters for generality. The distributed critic is optimized by minibatch gradient descent to minimize the following loss:

$$\begin{aligned} L_t(\theta_v) &= \left(y_t - V_{tot}(s_t)\right)^2 \\ &= \left(y_t - \sum_a V_{\theta_v}(o_t^a)\right)^2, \end{aligned} \tag{3.10}$$

where $y_t = \sum_{i=t}^{k-t-1} \gamma^i r_i + \gamma^{(k-t)} V_{tot}(s_k)$ is bootstrapped from the last state s_k, and k is upper-bounded by T.

The policy network is trained using the following policy gradient $g = \mathbb{E}_\pi[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) A(s, \mathbf{u})]$, where $A(s, \mathbf{u}) = r + \gamma V(s') - V(s)$ is a simple TD advantage. Similar to independent actor-critic (IAC), VDAC-sum does not make full use of CTDE in that it does not incorporate state information during training. Furthermore, it can only represent a limited class of centralized state-value functions.

VDAC-mix

To generalize the representation to a larger class of monotonic functions, we utilize a feed-forward neural network that takes input as local state values $V_\theta(o^a), \forall a \in \{1, \ldots, n\}$ and outputs the global state value V_{tot}. To enforce Equation 3.9, the

Algorithm 2 Value Decomposition Actor-Critic (VDAC-sum)

Initialize actor network θ
for each training episode e **do**
  Empty buffer
  for $e_c = 1$ to $\frac{\text{BatchSize}}{n}$ **do**
    $t = 0, h_o^a$ for each agent a
    while game not terminated **and** $t < T$ **do**
      $t = t + 1$
      for each agent a **do**
        $h_t^a, \pi_t^a, V_t^a = \text{Actor}(o_t^a, h_{t-1}^a, u_{t-1}^a, a; \theta)$
        Sample action u_t^a from π_t^a
      end for
      Get reward r_t and next state s_{t+1}
    end while
    add experience to buffer
  end for
  Collate episodes in buffer into single batch
  for $t = 1$ to T **do**
    Batch unroll RNN using states, actions and reward
    Calculate y_t and A_t using θ
    Accumulate gradient wrt $\theta : \Delta\theta_v \leftarrow \Delta\theta_v + \nabla_\theta \big(y_t - \sum_a V_t^a\big)^2$
  end for
  for $t = 1$ to T **do**
    Accumulate gradient wrt $\theta : \Delta\theta_\pi \leftarrow \Delta\theta_\pi + \nabla_\theta \log \pi(u_t^a | o_t^a) A_t$
  end for
  Update actor weights $\theta = \theta + \alpha_\pi \Delta\theta_\pi - \alpha_v \Delta\theta_v$
end for

weights (not including bias) of the network are restricted to be non-negative. This allows the network to approximate any monotonic function arbitrarily well [24].

The weights of the mixing network are produced by separate hypernetworks [32]. Following the practice in QMIX [73], each hypernetwork takes the state s as an input and generates the weights of one layer of the mixing network. Each hypernetwork consists of a single linear layer. An absolute activation function is utilized in the hypernetwork to ensure that the outputted weights are non-negative. The biases are not restricted to being non-negative. Hence, the hypernetworks that produce the biases do not apply an absolute non-negative function. The final bias is produced by a 2-layer hypernetwork with a ReLU activation function following the first layer. Finally, the hypernetwork outputs are reshaped into a matrix of appropriate size.

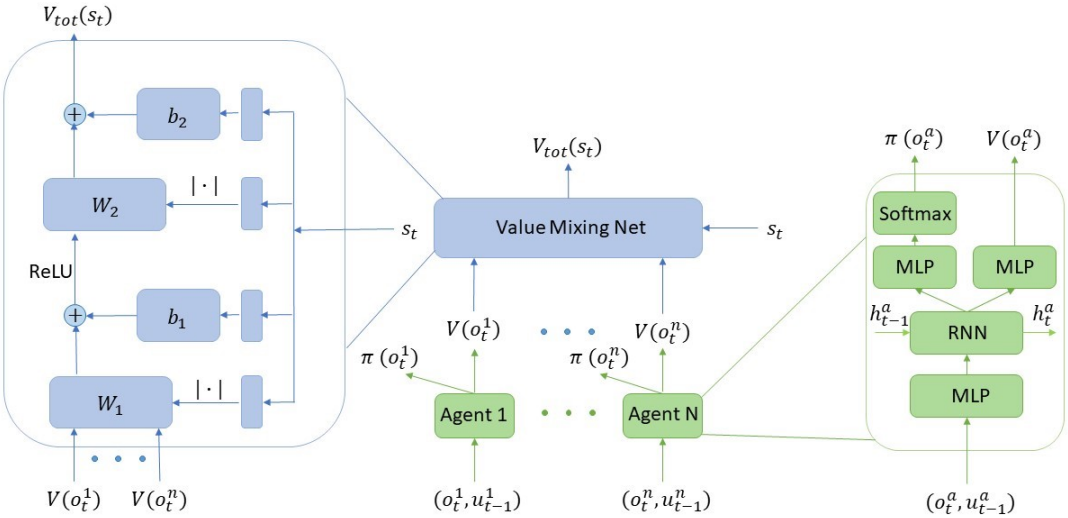


Figure 3.3: VDAC-vmix

Figure 3.3 illustrates the mixing network and the hypernetworks.

The whole mixing network structure (including hypernetworks) can be seen as a central critic. Unlike critics in [29], this critic takes local state values $V^a(o^a), \forall a \in \{1, \ldots, n\}$ as additional inputs besides global state s. Similar to VDAC-sum, the distributed critics are optimized by minimizing the following loss:

$$\begin{aligned} L_t(\theta_v) &= \Big(y_t - V_{tot}(s_t)\Big)^2 \\ &= \Big(y_t - f_{mix}(V_{\theta_v}(o_t^1), \ldots, V_{\theta_v}(o_t^n))\Big)^2, \end{aligned} \tag{3.11}$$

where f_{mix} denotes the mixing network. Let θ^c denote parameters in the hypernetworks. The central critic is optimized by minimizing the same loss $L_t(\theta^c) = (y_t - V_{tot}(s_t))$. The policy network is updated by following the same policy gradient in Equation 4.9. The pseudo code is provided in Algorithm 3.

3.5 Convergence of VDAC Frameworks

[29] establish the convergence of COMA based on the convergence proof of single-agent actor-critic algorithms [53, 83]. In the same manner, we utilize the following lemma to substantiate the convergence of VDACs to a locally optimal policy.

Algorithm 3 Value Decomposition Actor-Critic (VDAC-mix)

Initialize hypernetwork θ^c, and actor network θ
for each training episode e **do**
  Empty buffer
  for $e_c = 1$ to $\frac{\text{BatchSize}}{n}$ **do**
    $t = 0, h_o^a$ for each agent a
    while game not terminated **and** $t < T$ **do**
      $t = t + 1$
      for each agent a **do**
        $h_t^a, \pi_t^a, V_t^a = \text{Actor}(o_t^a, h_{t-1}^a, u_{t-1}^a, a; \theta)$
        Sample action u_t^a from π_t^a
      end for
      Get reward r_t and next state s_{t+1}
    end while
    add experience to buffer
  end for
  Collate episodes in buffer into single batch
  for $t = 1$ to T **do**
    Batch unroll RNN using states, actions and reward
    Calculate y_t and A_t using θ^c
    Accumulate gradient wrt θ^c : $\Delta\theta^c \leftarrow \Delta\theta^c + \nabla_{\theta^c}\big(y_t - V_{tot}(V_t^1, \ldots, V_t^n)\big)^2$
    Accumulate gradient wrt θ : $\Delta\theta_v \leftarrow \Delta\theta_v + \nabla_\theta\big(y_t - V_{tot}(V_t^1, \ldots, V_t^n)\big)^2$
  end for
  for $t = 1$ to T **do**
    Accumulate gradient wrt θ : $\Delta\theta_\pi \leftarrow \Delta\theta_\pi + \nabla_\theta \log \pi(u_t^a|o_t^a)A_t$
  end for
  Update actor weights $\theta = \theta + \alpha_\pi \Delta\theta_\pi - \alpha_v \Delta\theta_v$
  Update hypernet weights $\theta^c = \theta^c - \alpha\Delta\theta^c$
end for

Lemma 1: For a VDAC algorithm with a compatible TD(1) critic following a policy gradient

$$g_k = \mathbb{E}_\pi\left[\sum_a \nabla_{\theta_k} \log \pi(u^a|\tau^a)A(s, \mathbf{u}))\right],$$

at each iteraction k, $\liminf_k ||\nabla J|| = 0 \quad w.p.1.$

Proof The VDAC gradient is given by:

$$g = \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) A(s, \mathbf{u}) \right], \tag{3.12}$$

$A(s, \mathbf{u}) = Q(s, \mathbf{u}) - V_{tot}(s)$. We first consider the expected distribution of the baseline V_{tot}:

$$\begin{aligned} g_b &= -\mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) V_{tot}(s) \right] \\ &= -\mathbb{E}_\pi \left[\nabla_\theta \log \prod_a \pi(u^a|\tau^a) V_{tot}(s) \right], \end{aligned} \tag{3.13}$$

where the distribution $\mathbb{E}_\pi$ is with respect to the state-action distribution induced by the joint policy π. Writing the joint policy as a product of independent actors $\pi(\mathbf{u}|s) = \prod_a \pi(u^a|\tau^a)$. The total value does not depend on agent actions and is given by $V_{tot}(s) = f(V_1(o^1), \ldots, V_n(o^n))$ where f is a non-negative function. This yields a single-agent actor-critic baseline: $g_b = -\mathbb{E}_\pi[\nabla_\theta \log \pi(\mathbf{u}|s) V_{tot}(s)]$.

Now let $d^{\pi(s)}$ be the discounted ergodic state distribution as defined by [83]:

$$\begin{aligned} g_b &= -\sum_s d^{\pi(s)} \sum_{\mathbf{u}} \nabla_\theta \log \pi(\mathbf{u}|s) V_{tot}(s) \\ &= -\sum_s d^{\pi(s)} V_{tot}(s) \nabla_\theta \sum_{\mathbf{u}} \log \pi(\mathbf{u}|s) \\ &= -\sum_s d^{\pi(s)} V_{tot}(s) \nabla_\theta 1 \\ &= 0 \end{aligned} \tag{3.14}$$

The remainder of the gradient is given by:

$$\begin{aligned} g &= \mathbb{E}_\pi \left[\sum_a \nabla_\theta \log \pi(u^a|\tau^a) Q(s, \mathbf{u}) \right] \\ &= \mathbb{E}_\pi \left[\nabla_\theta \log \prod_a \pi(u^a|\tau^a) Q(s, \mathbf{u}) \right], \end{aligned} \tag{3.15}$$

which yields a standard single-agent actor-critic policy gradient $g = \mathbb{E}_\pi[\nabla_\theta \log \pi(\mathbf{u}|s) Q(s, \mathbf{u})]$. [53] establish that an actor-critic that follows this gradient converges to a local maximum of the expected return J^π, subject to assumptions included in their paper.

In the naive critic framework, $V_{tot}(s)$ is evaluated by the central critic and does not depend on agent actions. Hence, by following the same proof in Equation 3.14, we can show that the expectation of naive critic baseline is also 0, thus proves naive critic also converges to a locally optimal policy.

3.6 Experiments

In this section, we benchmark VDACs against the baseline algorithms listed in Table 3.1 on a standardized decentralised StarCraft II micromanagement environment, SMAC [76]. SMAC consists of a set of StarCraft II micromanagement games that aim to evaluate how well independent agents are able to cooperate to solve complex tasks. In each scenario, algorithm-controlled ally units fight against enemy units controlled by the built-in game AI. An episode terminates when all units of either army have died or when the episode reached the pre-defined time limit. A game is counted as a win only if enemy units are eliminated. The goal is to maximize the win rate.

We consider the following maps in our experiments: 2s_vs_1sc, 2s3z, 3s5z, 1c3s5z, 8m, and bane_vs_bane. Note that all algorithms are trained under A2C framework where 8 episodes are rolled out independently during the training. Refer to Appendix for training details and map configuration.

We perform the following ablations to answer the corresponding research questions:

3.6.1 Ablation 1

Is the TD advantage gradient sufficient to optimize multi-agent actor-critics? The comparison between the naive critic and COMA will demonstrate the effectiveness of TD advantage policy gradients because the only significant difference between those two methods is that the naive critic follows a TD advantage policy gradient whereas COMA follows the COMA gradient (Equation 3.7).

3.6.2 Ablation 2

Does applying state-value factorization improve the performance of actor-critic methods? VDAC-sum and IAC, both of which do not have access to extra state information, shares an identical structure. The only difference is that VDAC-sum applies a simple state-value factorization where the global state-value is a summation of local state values. The comparison between VDAC-sum and IAC will reveal the necessity of applying state-value factorization.

3.6.3 Ablation 3

Compared with QMIX, does VDAC provide a reasonable trade-off between training efficiency and algorithm performance? We train VDAC and QMIX under A2C

training paradigm, which is proposed to promote training efficiency, and compare their performance.

3.6.4 Ablation 4

What are the factors that contribute to the performance of the proposed VDAC? We investigate the necessity of non-linear value-decomposition by removing the non-linear activation function in the mixing network. The resulting algorithm is called VDAC-mix (linear) and can be seen as VDAC-sum with access to extra state information.

3.7 Overall Results

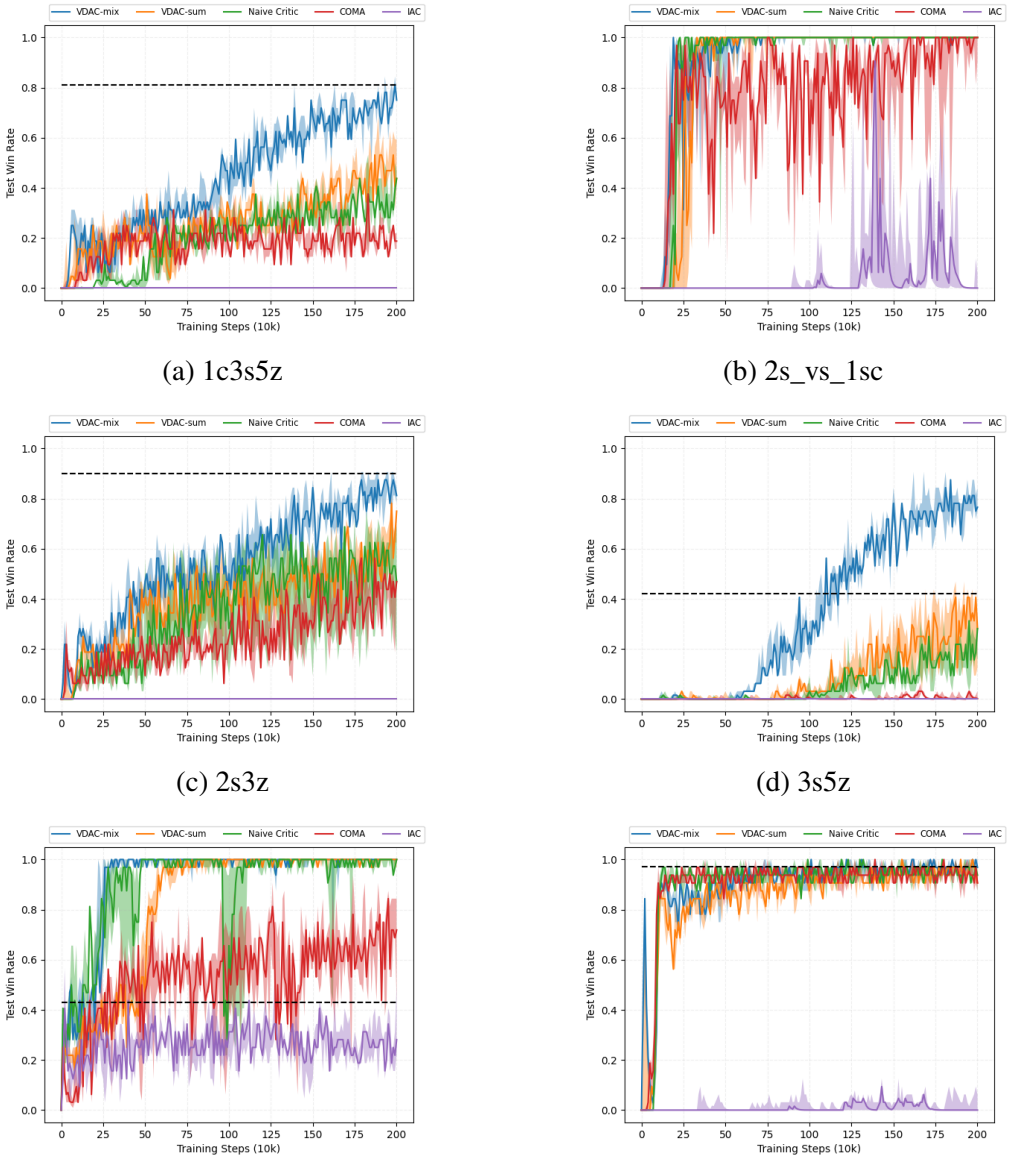
As suggested in [76], our main evaluation metric is the median win percentage of evaluation episodes as a function of environment steps observed over the 200k training steps. Specifically, the performance of an algorithm is estimated by periodically running a fixed number of evaluation episodes (in our implementation, 32) during the course of training, with any exploratory behaviours disabled. The median performance as well as the 25-75% percentiles are obtained by repeating each experiment using 5 independent training runs. Figure 3.4 demonstrates the comparison among actor-critics across 6 different maps.

In all scenarios, IAC fails to learn a policy that consistently defeats the enemy. In addition, its performance across training steps is highly unstable due to the non-stationarity of the environment and its lack of access to extra state information.

Noticeably, VDAC-mix consistently achieves the best performance across all tasks. On easy games (i.e, 8m), all algorithms generally perform well. This is due to the fact that a simple strategy implemented by the heuristic AI to attack the nearest enemies is sufficient to win. In harder games such as 3s5z and 2s3z, only VDAC-mix can match or outperform the heuristic AI. It is worth noting that VDAC-sum, which cannot access extra state information, matches the naive critic's performance on most maps.

3.7.1 Ablation 1

Consistent with [60], the comparison between the naive critic and IAC demonstrates the importance to incorporate extra state information, which is also revealed by the comparison between COMA and IAC (Refer to Figure 3.4 for comparisons between naive critic and COMA across different maps.). As shown in Figure 3.4, naive critic outperforms COMA across all tasks. It reveals that it is also viable to use a TD



(a) 1c3s5z

(b) 2s_vs_1sc

(c) 2s3z

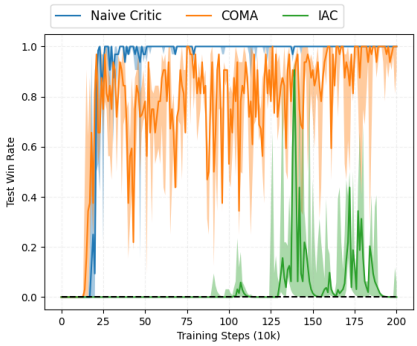
(d) 3s5z

(e) bane_vs_bane

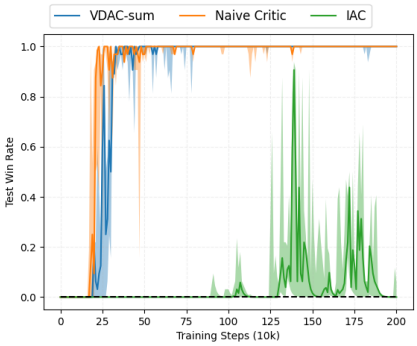
(f) 8m

Figure 3.4: Overall results: Win rates on a range of SC mini-games. Black dash line represents heuristic AI's performance

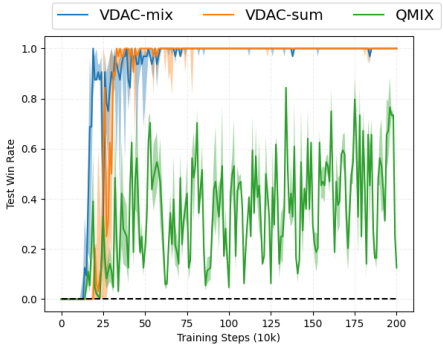
advantage policy gradients in multi-agent settings. In addition, COMA's training is unstable, as can be seen in Figure 3.5a, which might arise dues to its inability to predict accurate counterfactual action-value $Q^a(s, (\mathbf{u}^{-a}, u^a))$ for un-taken actions.



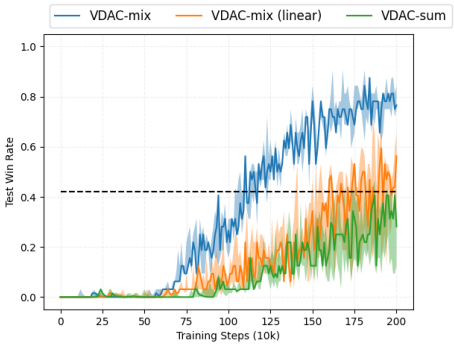
(a) 2s_vs_1sc (Ablation 1)



(b) 2s_vs_1sc (Ablation 2)



(c) 2s_vs_1sc (Ablation 3)



(d) 3s5z (Ablation 4)

3.7.2 Ablation 2

Despite the similarity in structure of VDAC-sum and IAC, VDAC-sum's median win rates at 2 million training step exceeds IAC's consistently across all maps (Refer to Figure 3.4 for comparisons between VDAC-sum and IAC across 6 different maps.). It reveals that, by using a simple relationship to enforce equation 3.9, we can drastically improve multi-agent actor-critic's performance. Furthermore, VDAC-sum matches naive critic on many tasks, as shown in Figure 3.5b, demonstrating that actors that are trained without extra state information can achieve similar performance to naive critic by simply enforcing equation 3.9. In addition, it is noticeable that, compared with naive critic, VDAC-sum's performance is more stable across training.

3.7.3 Ablation 3

Figure 3.5c shows that, under the A2C training paradigm, VDAC-mix outperforms QMIX in map 2s_vs_1sc. It is also noticeable that QMIX's performance is unstable across the training steps in map 2s_vs_1sc. In easier games, QMIX's performance can be comparable to VDAC-mix. In harder games such as 3s5z, VDAC-mix's median test win rates at 2 million training step outnumber QMIX's by 71%. Refer to Appendix for complete comparisons between VDACs and QMIX.

3.7.4 Ablation 4

Finally, we introduced VDAC-mix (linear), which can be seen as a more general VDAC-sum that has access to extra state information. Consistent with our previous conclusion, the comparison between VDAC-mix (linear) and VDAC-sum shows that it is important to incorporate extra state information. In addition, the comparison between VDAC-mix and VDAC-mix (linear) shows the necessity of assuming the non-linear relationship between the global state value V_{tot} and local state values $V^a, \forall a \in \{1, \ldots, n\}$. Refer to Appendix for comparisons between VDACs across all maps.

3.8 Conclusion

In this paper, we propose a new credit-assignment actor-critic framework that enforces the monotonic relationship between the global state-value and the shaped local state-value. Theoretically, we establish the convergence of the proposed actor-critic method to a local optimal. Empirically, benchmark tests on StarCraft micromanagement games demonstrate that our proposed actor-critic bridges the performance gap between multi-agent actor-critics and Q-learning, and our method provides a

balanced trade-off between training efficiency and performance. Furthermore, we identify a set of key factors that contribute to the performance of our proposed algorithms via a set of ablation experiments. In future work, we aim to implement our framework in real-world applications such as highway on-ramp merging of semi or full self-driving vehicles.

Large-scale multi-agent control problems are at the heart of a number of challenging problems facing society. For example in traffic management, there are over 300,000 accidents per year that occur during highway merging. The number of accidents could be significantly reduced if effective autonomous driving was broadly available in personal vehicles. MARL algorithms, like ones proposed in this paper, offer a possible solution to the autonomous driving task. Other areas of significant societal impact include healthcare, smart manufacturing, smart grids, and other transportation infrastructure.

3.9 Related Work

MARL has benefited from recent developments in deep reinforcement learning, with the field moving away from tabular methods [10] to deep neural networks [29]. Our work is related to recent advances in CTDE deep multi-agent reinforcement learning.

The degree of training centralization varies in the literature on MARL. *Independent Q-learning* (IQL) [85] and its deep neural network counterpart [84] train an independent Q-learning model for each agent. Those that attempt to directly learn decentralized policies often suffer from the non-stationarity of the environment induced by agents simultaneously learning and exploring. [28, 87] attempt to stabilize learning under the decentralized training paradigm. [31] propose a training paradigm that alternates between centralized training with global rewards and decentralized training with shaped rewards.

Centralized methods, by contrast, naturally avoid the non-stationary problem at the cost of scalability. COMA [27], takes advantage of CTDE, where actors are updated by following policy gradients that are tailored by their contributions to the system. *Multi-agent deep deterministic policy gradient* (MADDPG) [60] extends *deep deterministic policy gradient* (DDPG) [58] to mitigate the issue of high variance gradient estimates exacerbated in multi-agent settings. Based on MADDPG, [92] propose multi-agent soft Q-learning in continuous action spaces to tackle the issue of *relative overgeneralization*. *Probabilistic recursive reasoning* [93] is a method that uses a probabilistic recursive reasoning policy gradient that enables agents to recursively reason what others believe about their own beliefs.

More recently, value-based methods, which lie between the extremes of IQL

and COMA, have shown great success in solving complex multi-agent problems. VDN [82], which represents joint-action value function as a summation of local action-value function, allows for centralized learning. However, it does not make use of extra state information. QMIX [73] utilizes a non-negative mixing network to represent a broader class of value-decomposition functions. Furthermore, additional state information is captured by hypernetworks that output parameters for the mixing network. QTRAN [79] is a generalized factorization method that can be applied to environments that are free from structural constraints. Other works, such as CommNet [27], TarMAC [21], ATOC [44], MAAC [43], CCOMA [80] and BiCNet [71] exploit inter-agent communication.

The proposed VDAC method is similar to QMIX and VDN in that it utilizes value-decomposition. However, VDAC is a policy-based method that decomposes global state-values whereas QMIX and VDN, which decompose global action-values, belong to the Q-learning family. [68] address credit-assignment issue, however, under a different MARL setting, $\mathbb{C}$Dec-POMDP. COMA, which is also a policy gradient method inspired by *difference rewards* and has been tested on StarCraft II micromanage games, represents the work most closely related to this paper.

Chapter 4

Preventive Maintenance with VDAC

4.1 Background

In manufacturing systems, machines suffer from random failures as the result of degradation of parts over time [89]. These unexpected failures abruptly interrupt normal production operations, which can not only cause significant production losses but also require considerable resources for prompt maintenance actions. The maintenance procedure in response to random failures is referred to as corrective maintenance (CM). In order to reduce random failures, a common industrial practice is to proactively shut down machines for preventive maintenance (PM) according to predefined policies. However, deriving PM policies that ensure smooth and efficient production has always been a nontrivial task, since PM actions also impact the system in ways that can incur production losses and resource costs. The costs of excessive PM might outweigh the benefits, but conversely, inadequate PM can be ineffective in preventing random failures. It is challenging to balance the delicate decision trade offs that arise in PM for manufacturing systems, which are distinguished by complicated and non-linear system dynamics due to interactions among machines/buffers and interruptions from production-related activities [39].

Reinforcement learning (RL) is a machine learning technique that estimates optimal policies through interactions between an agent and the environment. RL has recently emerged as an effective method for obtaining PM policies in manufacturing systems thanks to its capability for dealing with complex and stochastic environments. The complexity and size of the studied manufacturing systems, and hence RL techniques applied, vary across the spectrum in this line of research. Model-based RL methods utilizing known transition probabilities between states are mostly restricted to two-machine-one-buffer manufacturing systems [26, 45, 91], because it is impractical to obtain the transition probabilities for larger systems. The explosion of state space with increased system size warrants the use of model-free RL

methods, which do not require a model of the environment dynamics, and deep RL, which approximates value functions or policy with deep neural networks [38, 40]. Whether model-based or model-free, prior RL work shares the approach of modeling the maintenance problem using a centralized RL framework in which all machines across the manufacturing system are precisely coordinated by a central agent (see, e.g., [26, 38, 40, 45, 91]). This commonality in modeling framework arises from the fact that standard RL is designed for a single agent, and manufacturing research aims to improve the performance of the entire manufacturing system rather than individual machines within the system. Traditional group maintenance (GM) and opportunistic maintenance (OM) are inspired by the observation that the maintenance of one unit does not result additional production loss if another unit is already under maintenance [89]. Under the centralized framework, the RL-based PM policy not only outperforms traditional GM/OM policies, but also learns to conduct GM/OM when necessary [38].

These previously outlined approaches demonstrate the technical feasibility of applying state-of-the-art RL methods in complex maintenance problems. However, there are still quite a few practical constraints in many real production systems that existing methods fail to incorporate. First, most existing approaches (see, e.g., [26, 38, 40, 45, 91]) assume that maintenance activities restore a machine to its perfect health state. These approaches fail to consider the industrial reality that there are often multiple maintenance options that can lead to different post-maintenance health states and that maintenance activities could be imperfect [41,46]. Second, existing methods fail to cover the full spectrum of production systems in the real world, as production system size dramatically varies from a two-stage machining line to automotive assembly plant with dozens of stations. The current approaches, which have issues being scaled to large manufacturing systems, are often restrained to systems of relatively small sizes. For instance, most of target systems for serial production lines range from two-machine-one-buffer to six-machine-five-buffer [26, 38, 45, 91]. This is because single-agent RL methods are prone to action space explosion as the RL agent's action size grows exponentially with number of machines considered in the system. This exponential increase in the size of the action space can lead to poor performance [29]. Third, when a centralized RL framework is adopted, these approaches implicitly assume that all the state information in manufacturing system is observable. For some large-scale manufacturing systems, system-wide data collection and transmission could be inefficient and even infeasible [16, 49]. Finally, it appears to be appealing to model each machine in a complex system as a model-free RL-agent and optimize individual performance to alleviate the problem of action space explosion. However, handcrafting individual rewards is often impractical because of the difficulty inherent in quantifying an individual component's contribution to the success or failure of the system. Hence, optimizing

individual performance might contradict the collective objective of the system. In addition, the interactions among agents are ignored since they are considered as a part of an agents' environment.

4.2 Executive Summary

To address the limitations of RL-based PM policies, we extend our previous RL framework [38] to a multi-agent setting and apply MARL to obtain optimal policies. Under this setting, the manufacturing system is modeled as a cooperative system where agents make decisions separately while collectively achieving a common goal of the system. MARL then obtains optimal policies through interactions between agents and the environment. Besides addressing the aforementioned limitations, MARL is an appealing and viable technical approach to taking on those challenges in maintenance problem among large-size manufacturing systems for the following reasons. First, similar to autonomous vehicle coordination [80] and traffic lights control [94], it is natural to model complex manufacturing systems as cooperative multi-agent systems where machines interact and cooperate with others. Second, MARL allows us to take advantage of edge computing to achieve real-time policy execution without assuming global observability as it allows each machine to individually make maintenance decision that only condition on their local observation-action trajectories. Third, MARL guarantees that separate maintenance decisions from individual machines could collectively achieve the common goal of the whole system, without the need to handcraft individual rewards for each machine. Our principal contributions are as follows:

- We settle the action space explosion issue in RL-based PM policies by proposing a deep MARL approach as opposed to centralized RL. PM decision-making based on MARL can be scaled up to large manufacturing systems while still outperforming traditional multi-unit maintenance policies;

- Guided by knowledge on manufacturing systems, we formulate the PM decision making problem using a decentralized partially observable Markov decision process (Dec-POMDP) framework, under which we are able to adopt state-of-the-art RL paradigms such as centralized training and decentralized execution (CTDE) that largely increases policy performance;

- Free from action space explosion issue, we further advance the RL applications in PM problems to more realistic industrial scenarios, for example, important practical constraints including imperfect maintenance effect, local observability and production disruptions can be considered;

- We demonstrate the proposed method by successfully implementing a state-of-the-art MARL algorithm called Value-Decomposition Actor Critic to solve the PM problem in large manufacturing systems efficiently and effectively. Machines make PM decisions independently but still learn to cooperate to perform OM/GM as reported in prior research based on centralized RL [38].

4.3 Problem Statement

4.3.1 System Description

We consider a serial production line that consists of n machines and $n-1$ buffers with limited capabilities as shown in Figure 4.1. The arrows depict the direction of material flow in the system. The material is referred to as a final product once it has been processed by all machines sequentially. Otherwise, it is said to be an intermediate part.

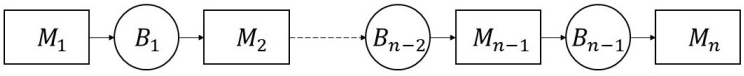


Figure 4.1: Serial Product Line Structure

A machine $M_a,\ a \in \{1, 2, \cdots, n\}$, possesses the following properties: the cycle time T_a denoting the time needed for M_a to process an intermediate part, the lifespan l_a that represents the time it takes for a machine from the perfect health status to failure and follows a known probability distribution p_a, the resource cost c_a^{PM} and duration d_a^{PM} needed for conducting a PM, the resource cost c_a^{CM} and duration d_a^{CM} needed for conducting a CM.

A buffer $B_a,\ a \in \{1, 2, \cdots, n-1\}$, is located between M_a and M_{a+1} and can be best described by the following properties: current buffer level b_a, and buffer capability c_a. An intermediate part that comes out of machine M_a enters its downstream buffer B_a if and only if $b_a < c_a$, otherwise M_a is said to be blocked. Machine M_a starts a new cycle by receiving an intermediate part from its upstream B_{a-1} if $b_{a-1} > 0$, otherwise M_a is said to be starved. Note that the first machine M_1 is never starved and the last machine M_n is never blocked.

Maintenance actions, either PM or CM, would require the machine to temporarily cease production. If a machine M_a is under maintenance, intermediate parts gradually drains in its downstream buffers and increasingly pile up in its upstream buffers, which might lead adjacent machines to idle states due to blockage or starvation. The gradual propagation of machines' idleness could finally cause production

loss of the whole system. The analytical model and system property analysis for the described production line can be found in our prior work [99].

4.3.2 Maintenance Effect and Maintenance Actions

The extent to which the maintenance action can restore a machine's health state is referred to as maintenance effect. In this work, we use Kijima Model II [46] to model the effects of different levels of maintenance actions. Let g_a denote the machine M_a's age prior to maintenance, then the machine's age immediately after the maintenance, denoted by g'_a, is given by

$$g'_a = g_a * r_a \tag{4.1}$$

where $0 \leq r_a < 1$ is the recovery factor of the maintenance on machine M_a. $r_a = 0$ indicates a perfect maintenance, since the machine age g'_a is set to be zero after maintenance. By contrast, $0 < r_a < 1$ relates to an imperfect maintenance, as the post-maintenance age g'_a starts somewhere between zero and the original age g_a. In other words, imperfect maintenance does not fully restore machine's health condition and hence earlier arrival of subsequent random failure would be expected after imperfect maintenance than a perfect one.

In this work, we consider two levels of PM for each machine, namely an imperfect (level 1) PM option with and a perfect (level 2) PM option, denoted as PM_1 and PM_2 respectively, with recovery factors $0 < r_a^{PM_1} < 1$ and $r_a^{PM_2} = 0$. Generally, imperfect PM costs less resource and has shorter duration than perfect PM. In addition, CM is conducted in response to random failures and its effect is assumed to be perfect, i.e. $r_a^{CM} = 0$. The cost and duration of CM are much larger than that of any PM options [17]. In summary, maintenance resource costs rank as $C_a^{PM_1} < C_a^{PM_2} \ll C_a^{CM}$ and maintenance duration satisfies $d_a^{PM_1} < d_a^{PM_2} \ll d_a^{CM}$.

4.3.3 Cost Analysis and System Objective

Given a time horizon T, the total system cost related to maintenance activities can be broken down into: resource costs of all CM actions $C_{CM}(T)$, resources costs of all PM actions $C_{CM}(T)$, and loss of revenue $C_{PL}(T)$ on account of system production losses. It is noted that both PM and CM would contribute to $C_{PL}(T)$ by introducing machine stoppages that in turn cause system production losses.

Let π^a denote PM policy for machine M_a, and π represent system-level joint PM policy. The PM policy π instructs when and where to conduct PM, thus largely shaping the cost break-downs and hence the total system cost. We use $C_\pi(T)$ to represent total system cost in time horizon T under a joint PM policy π, and it can be written as:

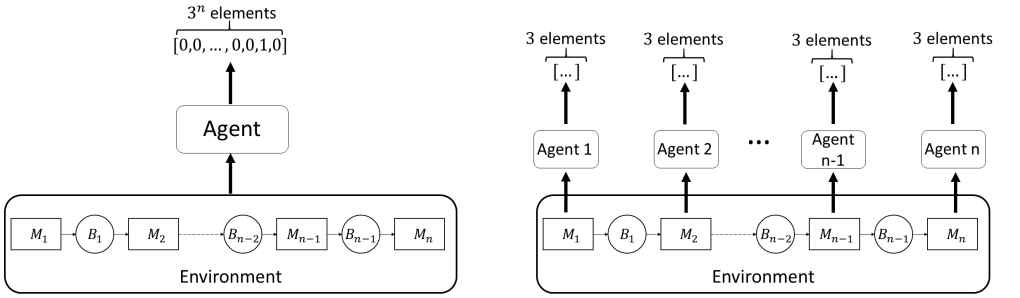
$$C_\pi(T) = C_{PM}(T) + C_{CM}(T) + C_{PL}(T) \tag{4.2}$$

The objective of this study is to find an optimal joint policy π^* to minimize the total cost C in a time horizon T through deep MARL techniques:

$$\pi^* = \arg\min_{\pi}\{C_\pi(T)\} \tag{4.3}$$

4.4 Multi-Agent Adaptive Decision Framework

Our previous study proposed a centralized adaptive decision framework that is based on DQN [38] and successfully applied it to a six-machine-five-buffer production line. However, this RL-based framework has issues being generalized to more realistic applications due to action space explosion. Consequently, the framework only consider one level of PM to limit the action space size of the studied system.



(a) Serial production line RL formulation (b) Serial production line MARL formulation

Figure 4.2: Action space comparison between RL and MARL formulations

In comparison, MARL is less prone to action space explosion as its action space is independent of number of agents in the system. This is because, under the MARL's decentralized setting, the agents make decisions independently. Figure 4.2 depicts the action spaces of a n-machine production line under RL and MARL respectively.

The following sections are organized to describe: 1) how to formulate the PM of serial production line problem to a MARL problem; and 2) how to derive PM policies effectively and efficiently with the aforementioned VDAC algorithm.

4.4.1 Dec-POMDP Formulation

State Definition

: In the CTDE framework, agents condition their actions on the local partial observations and are optimized by the gradients that are dependent on global states.

Therefore, we need to design local observations o that provide the basis for agents' actions and global states s that encapsulate information that is useful for training.

Given a machine M_a, three factors that are essential to its PM decision making are: 1) the age of machine M_a, g_a; 2) the upstream and downstream buffer status related to b_{a-1} and b_a; 3) M_a's remaining maintenance duration, d_a, if applicable. In addition, the status of M_a's immediate adjacent machines is also useful because it might relate to the blockage or starvation of machine M_a. Consequently, the local observation for machine M_a is defined as:

$$o^a = [g_a, g_{a-1}, g_{a+1}, b_{a-1}, c_a - b_a, c_{a+1} - b_{a+1}, d_a, d_{a-1}, d_{a+1}]. \tag{4.4}$$

Note that we use buffer vacancy $c_a - b_a$ instead of buffer level b_a to represent the real-time status of the downstream buffer. This is because the buffer vacancy is a more sufficient criteria than buffer level in determining if the machine is blocked or not. Furthermore, the s that represents the global state of the production line is obtained by concatenating local observations from all machines and it is written as:

$$s = [o^1, o^2, \cdots, o^n]. \tag{4.5}$$

Action Definition

: CM is triggered by a machine's random failure and is out of the machine's control. Hence, the action that can be taken for a machine is to conduct a PM or leave it as it is. In this study, we consider two levels of maintenance actions: a perfect (level 2) PM and a imperfect (level 1) PM. This leaves us an action vector of size 3 for each agent. For instance, machine M_a's action can be written as:

$$u^a = \begin{cases} 0 & \text{leave machine } a \text{ as it is} \\ 1 & \text{conduct imperfect (level 1) PM on machine } a \\ 2 & \text{conduct perfect (level 2) PM on machine } a. \end{cases} \tag{4.6}$$

This action representation can be extended to the cases where more levels of PMs are considered. Note that the action space will be 3^n for the same manufacturing system under the centralized RL framework as shown in Figure 4.2.

Reward Definition

: On one hand, machine starvation and blockage caused by random failure might lead to production loss. On the other hand, excessive PM would also result in increasing machine down time and cost. To balance the trade-off between PM and random failure, we define the reward as the negative overall cost, which consists

of production loss cost due to machine stoppages, and maintenance resource costs incurred by PM and CM. At time step t, the reward $r(t)$ is given by:

$$r(t) = -PL(t) \cdot c_p - \sum_{a=1}^{n} c_a^{PM_1} \mathbb{1}_a^{PM_1}(t) - \sum_{a=1}^{n} c_a^{PM_2} \mathbb{1}_a^{PM_2}(t) - \sum_{a=1}^{n} c_a^{CM} \mathbb{1}_a^{CM}(t), \tag{4.7}$$

where $PL(t) \cdot c_p$ represents production loss cost at step t with c_p being the unit price of a final product, $\sum_{a=1}^{M} c_a^{CM} \mathbb{1}_a^{CM}(t)$ represents cost incurred by CM at step t, $\sum_{a=1}^{n} c_a^{PM_1} \mathbb{1}_a^{PM_1}(t) + \sum_{a=1}^{n} c_a^{PM_2} \mathbb{1}_a^{PM_2}(t)$ represents overall PM cost. $\mathbb{1}_a^{PM_1}(t) = 1$ if M_a is conducting level 1 PM at time t, otherwise $\mathbb{1}_a^{PM}(t) = 0$. Similarly, $\mathbb{1}_a^{CM}(t) = 1$ if M_a is under CM at time t, otherwise $\mathbb{1}_a^{CM}(t) = 0$.

Zou et al. [99] has proved that a downtime event would lead to system-level production loss only when it impedes the slowest machine, and therefore the production loss at time step t can be evaluated by:

$$PL(t) = \frac{D(t)}{T_{n*}}. \tag{4.8}$$

Here, $n*$ indexes the slowest machine in the serial line, $D(t)$ denotes the stoppage time of the slowest machine at step t, and T_{n*} denotes the cycle time of the slowest machine.

4.4.2 Applying VDAC to obtain PM Policy

It is often elusive to handcraft rewards for individual agents in complex systems. *Difference rewards* $D^a = r(s, \mathbf{u}) - r(s, (\mathbf{u}^{-a}, c^a))$, which adopts the reward changes incurred by replacing agent a's action with a default action c^a as local rewards [95]. The intent of the difference rewards substitution is to shed light on quantifying individual rewards in cooperative multi-agent systems. Unfortunately, difference rewards requires $n - 1$ counterfactual simulations to be rolled out at every time step, which can present an extreme computational burden. While it is impossible to implement difference rewards in practice, it is easy to enforce the monotonic relation between shaped local reward D^a and $r(s, \mathbf{u})$ as indicated by difference rewards.

Inspired by difference rewards, VDAC is an on-policy multi-agent actor-critic that enforces the monotonic relation between local state-values and global state-values. Specifically, a mixing neural network represents the global state-value as a non-linear function of local state-values. To enforce the monotonic relation between local and global state-values, weights of mixing neural network are restricted to be non-negative. This constraint ensures that given agents other than agent a stay at the same local states, the global state-value should increase if agent a transmit to local states with higher local state-values.

As shown in Figure 4.2b, distributed agents are responsible for making decisions for the corresponding machine under the MARL setting.

4.4.3 VDAC Architecture

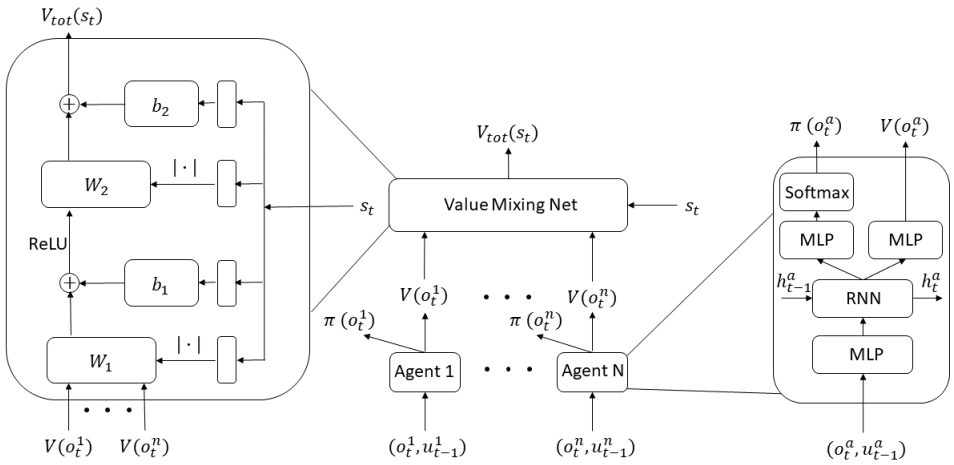


Figure 4.3: VDAC Architecture

VDAC consists of distributed actors that make decisions for designated machines and a central critic that estimate the global state-value V_{tot}. As shown in Figure 4.3, the actor network takes inputs as observations o_t^a and actions u_{t-1}^a of the previous step $t-1$, outputs a multinomial policy $\pi(o_t^a)$ for timestep t. It also estimates the state-value of the local observation $V(o_t^a)$. To capture the temporal dependencies within agents' trajectories, Gated Recurrent Unit (GRU) is Incorporated in actor networks. Note that to speed up training as well as save memory, actor networks share the same weights. The value mixing network, which takes input as local state-values $V(o_t^a)$ and outputs the global state-value $V_{tot}(s)$, serves as a central critic. To incorporate the global state information that is unavailable to actors, the parameters of the value mixing network is generated from a hypernetwork which takes input as the global state s_t. Please refer to [81] for details about VDAC method.

Actors are optimized by following gradients that depend on the central critic. Let θ_π to denote actor network parameters and θ_V to denote hypernet parameters for generality. The actor network is optimized by following the policy gradient given

by:

$$\nabla_{\theta_\pi} J = E_\pi \left[\sum_a \nabla_{\theta_\pi} \log \pi(u^a|\tau^a) \big(Q(s, \mathbf{u}) - V(s) \big) \right], \tag{4.9}$$

where $V(s)$ is estimated by a central critic and $Q(s, \mathbf{u}) = r + \gamma V(s\prime)$. The critic is optimized by minibatch gradient descent to minimize the following loss:

$$L_t(\theta_V) = \Big(y_t - V_{tot}(s_t; \theta_V) \Big)^2, \tag{4.10}$$

where $y_t = \sum_{i=0}^{k-1} \gamma^i r_t + \gamma^k V(s_{t+k})$ is the target value, k can vary from state to state and is upper-bounded by T_{max}.

Action Mask

Note that when machine M_a is under maintenance (namely, $d_a(t) > 0$), the machine cannot conduct actions that interrupt its current maintenance. Hence the only action available for machine M_a is to leave it as it is. To enforce the rules of the studied process, an action mask m_t is often applied to filter out invalid actions [76, 78]. More specifically, our multinomial policy network outputs a 3-element action probability vector vec_t, representing the probability of taking each action. Element-wise multiplication $vec_t \otimes m_t$ makes sure that the probability for invalid actions to be 0. The resulted vector is then normalized such that the sum of the vector equals to 1.

Implementation

Figure 4.5 demonstrate the procedure to implement VDAC to the maintenance problem in serial production lines. The procedure can be generally divided into two parts: 1) Obtaining experience, and, 2) optimizing parameters. In the phase of obtaining experience, multiple episodes are rolled out independently to increase experience sampling efficiency. For an agent a, its trajectory T_i at episode i is recorded $\{(o_0^a, s_0, u_0^a, r_0, a), \cdots, (o_{T-1}^a, s_{T-1}, u_{T-1}^a, r_{T-1}, a)\}_i$. In addition, its corresponding action masks $\{m_0^a, m_1^a, \cdots, m_{T-1}^a\}_i$ at episode i are also recorded. Note that the central critic is absent during this phase. During the parameter optimization phase, the data acquired in the sampling phase will be discarded once it is used to optimize the network parameters. Therefore, no replay buffer is required.

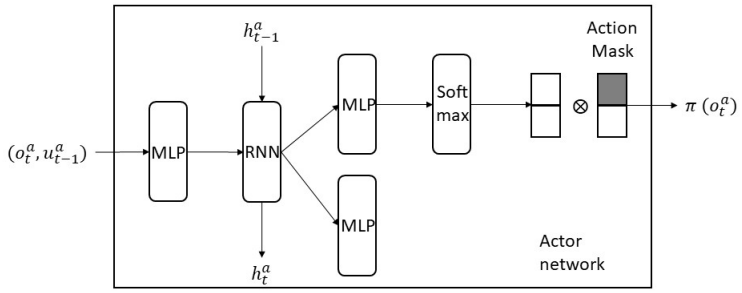


Figure 4.4: Action Mask

4.5 Numerical Study

Two numerical studies are conducted in our experiments to demonstrate the effectiveness of the proposed MARL-base policy. The first numerical study compares the proposed methods with others, utilizing a six-machine-five-buffer serial production line adopted from our previous study on DQN-based policy [38]. The second numerical study uses a ten-machine-nine-buffer serial production line to further examine the scalability of those methods.

4.5.1 Environment Description

We first test our method on a serial production line simulation that is extended from [38]. Compared with the original simulation introduced in [38], our simulation is obtained by simply adding an imperfect PM action for each machine. The proposed MARL policy is compared with the DQN-based policy as well as other traditional PM policies. Note that by adding one action for each machine, the action space of the DQN agent increases from $2^6 = 64$ to $3^6 = 729$. The simulation parameters are provided by industry collaborators and are listed in Table 4.1 and 4.2. Machine M_a's life span follows a Weibull distribution whose scale parameter and shape parameter are α_a and β_a respectively.

For each experiment run, the simulation duration is 1500 timesteps. Each run starts with randomly generated initial system states, including random machine ages

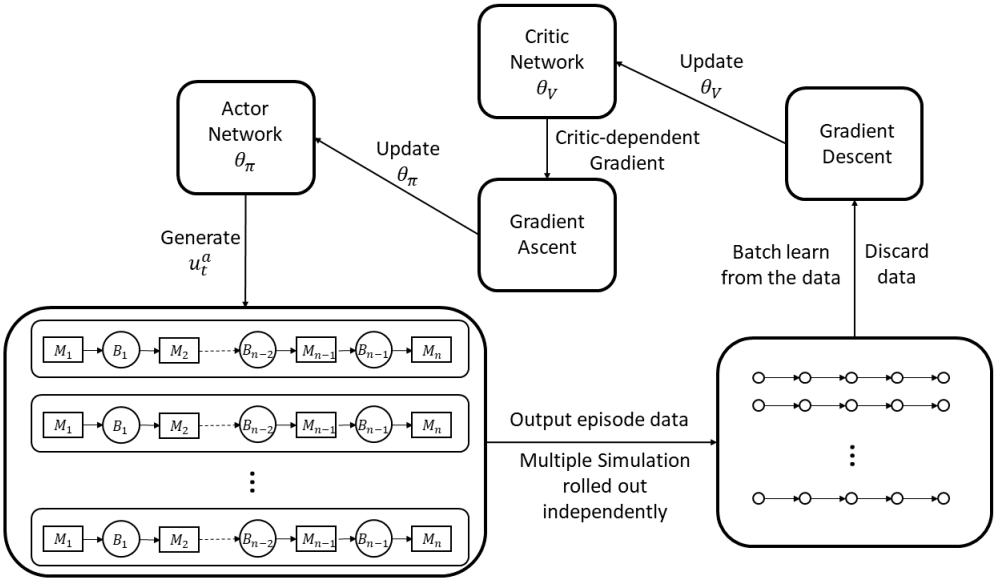


Figure 4.5: Implementation Details

and random buffer levels, to make sure that the agents' policies are independent of initial conditions. During the simulation, every time a machine finished its PM or CM, its lifespan is sampled from the Weibull distribution conditioning on post-maintenance age.

4.5.2 Baselines

Deep Q-learning (DQN): Following [38], we implement a DQN policy for multi-level PM in the serial production line. To make the DQN agent consistent with MARL agents, our DQN policy network follows DRQN [35] where a GRU [19] is utilized to memorize the past state of the system. Note that the DQN agent can access global state s whereas our MARL agents can only access partial observations. DQN also need an action mask to filter out invalid actions. At time t, invalid actions $u_t^{invalid}$ are masked out by setting $Q(s_t, u_t^{invalid}) = -\infty$.

Run-to-Failure (R2F): The Run-to-Failure scenario is used to evaluate the system performance if no PM is conducted throughout the time horizon. Each machine in the production line keeps running until it encounters a random failure. Hence, CM is the only type of maintenance that is conducted in this scenario. Any other PM policy is deemed as effective only when it improves system performance from the Run-to-Failure scenario.

Group Maintenance (GM): Given GM policy, all the machines would receive PM simultaneously. GM policy tends to reduce the impacts of PM on the whole

Table 4.1: Machine Parameters

Parameters	M_1	M_2	M_3	M_4	M_5	M_6
Cycle Time T_a (min)	1.00	0.90	1.20	1.05	1.10	1.05
Level 1 PM cost $c_a^{PM_1}$ (\$)	50.0	45.0	55.0	60.0	45.0	50.0
Level 2 PM cost $c_a^{PM_2}$ (\$)	105.0	98.0	110.0	120.0	90.0	103.0
Level 1 PM duration $d_a^{PM_1}$ (min)	5	4	4	5	6	4
Level 2 PM duration $d_a^{PM_2}$ (min)	10	9	8	11	12	9
CM cost c_a^{CM} (\$)	110.0	100.0	120.0	130.0	110.0	125.0
CM duration d_a^{CM} (min)	28	31	25	32	24	25
level 1 PM degree $r_a^{PM_1}$	0.8	0.8	0.8	0.8	0.8	0.8
level 1 PM degree $r_a^{PM_2}$	0	0	0	0	0	0
Scale parameter α_a	400	430	500	580	550	575
Shape parameter β_a	15	15	15	15	15	15

Table 4.2: Buffer Parameters

Parameters	B_1	B_2	B_3	B_4	B_5
Buffer capability b_i	8	10	12	6	8

system by enforcing concurrent machine stoppages. There is one pivotal decision variable in GM policy, which is the time interval τ for carrying out group PM. Since there is no analytical method to derive τ due to the ultra complexity of the production system, the optimal τ can be found through Monte Carlo simulation.

Opportunistic Maintenance (OM): OM policy is inspired from the observation that once there is a machine undergoing CM, other machines can receive PM without incurring extra system production losses. Under OM policy, whenever one or multiple machines fails in the serial production line, we will conduct CM on those failed machines. In the meantime, we will turn off all other operational machines for PM.

Opportunistic Group Maintenance (OGM): OGM policy is a combination of OM policy and GM policy. Similar to GM, there is also a predefined time interval τ. If there is a random failure occurs before the time interval τ arrives, OM policy would be triggered, i.e. all other operational machines are turned off for PM. If the time interval τ is reached without machine random failures, GM policy would be carried out so that all the machines receive PM simultaneously. We also leverage Monte Carlo simulation to derive the optimal τ.

4.5.3 Training Description

MARL training: The agent networks contains a GRU [19] with a 64-dimensional hidden state, with a fully-connected layer before and after. Algorithms are trained with RMSprop with learning rate 5×10^{-4}. We set $\gamma = 0.99$. The mixing network consists of a single hidden layer of 32 units, whose parameters are outputted by hypernetworks. An ELU activation function follows the hidden layer in the mixing network. The hypernetworks consist of a feedforward network with a single hidden layer of 64 units with a ReLU activation function. Throughout the paper, $Q(s_t, \mathbf{u}_t)$ is given by:

$$Q(s_t, \mathbf{u}_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}), \tag{4.11}$$

where k can vary from state to state and is upper-bounded by T.

As described in the previous section, the algorithm alternates between sampling experience and optimizing parameters. MARL model is set to train 2050000 timesteps. Eight episodes are run independently during sampling phase. Models are saved every 200000 training timesteps. The best saved model is later benchmarked with other baselines. Note that actions are drawn according to the multinomial probability outputted by the actor network during the training whereas actions with the largest probability are picked during the test.

Training experiments are conducted on a PC with a Nvidia RTX 2080 Ti graphics card, a AMD 3700X CPU and a 32 GB RAM, with one run taking approximately 1 hour.

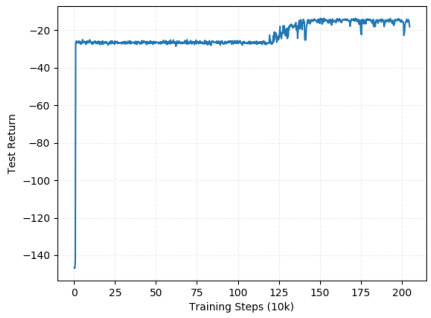
RL Training: We also train a RL agent for comparison. The Q-learning network is identical to the actor network described in the previous section except that the Q-learning network does not have a softmax layer and an additional state-value layer. The buffer size is set to be 450000. During the training, actions are selected via ϵ-greedy. ϵ is set to 1 initially and then linearly decreases to 0.05 in 50000 training steps. During the testing, the action with the largest action-value is picked. Other settings are identical to MARL training settings. Note that DQN is an off-policy algorithm, which is not compatible with the A2C framework. Therefore, one episode, instead of 8 independent episodes, is rolled out during training. DQN's training process takes approximately 4 hours with the same hardware that described above.

Baselines: OM, OGM, and GM methods cannot condition their choices of action on states of the system. Namely, they can either always conduct level-1 PM or level-2 PM. In addition, OGM and GM have a variable τ that decides when to conduct PM. Since there is no analytical method to derive τ and choice of actions, we determine the best combinations via Monte Carlo simulation: OM, OGM, and

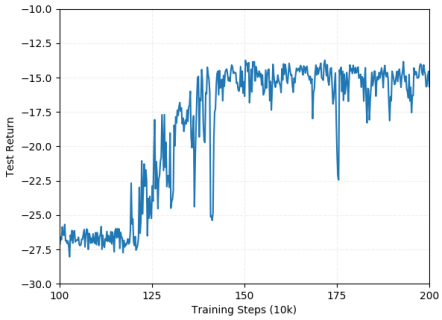
GM always take level-1 (imperfect) PM; the optimal τ is 360 for GM and 350 for OGM.

4.5.4 Evaluation

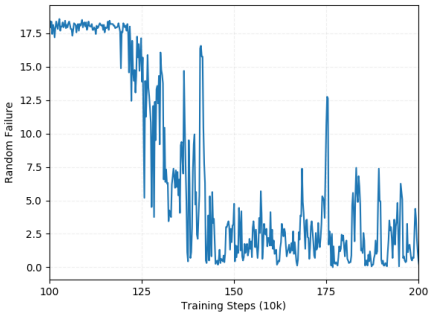
Training Monitoring: First, we monitor the training of MARL to verify if the



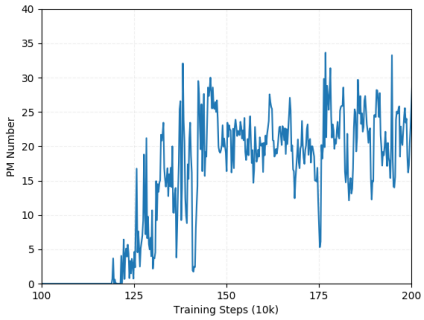
(a) Test return as a function of training step



(b) A closer look at test return



(c) Random failures counts as a function of training steps



(d) PM counts as a function of training steps

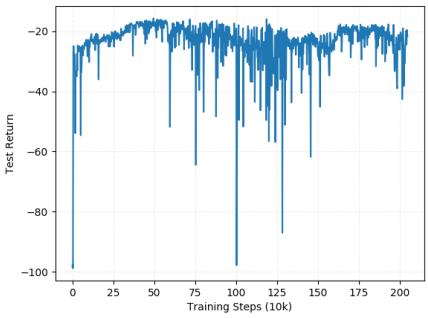
Figure 4.6: MARL Training Monitoring

algorithm converges. During the course of the training, we periodically halt the training and test the trained model at the current training step for 16 independent episodes. Hence, we can evaluate the model's performance as a function of training steps.

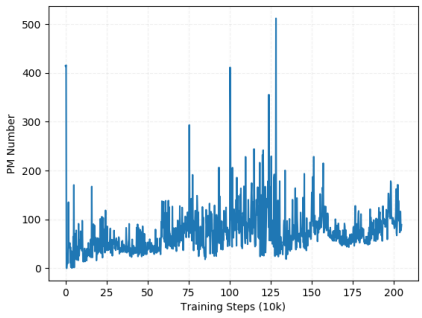
Since the VDAC's objective is to maximize the return $R_t = \sum_t^{T-1} \gamma^{t-1} r_t$, we examine the average of return as a function of training steps. As shown in Figure 4.6a, the average return plateaus until around 1.2 million training steps and then converges to higher values.

Figure 4.6c and 4.6d depict the average number of random failures and PMs per episode, respectively. They demonstrate that the VDAC algorithm learns to reduce random failure occurrences by applying timely PM at around 1.2 million training steps. Through the course of training after 1.2 million training steps, policies alternate from aggressive PM and passive PM without affecting return values drastically.

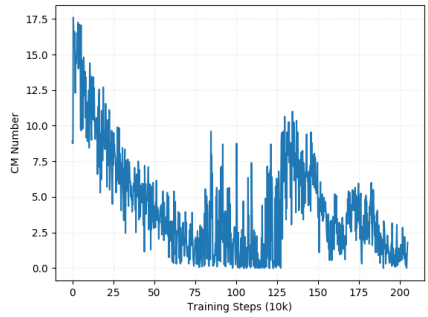
With the evidence provided by Figure 4.6, we conclude that MARL policy can converge during the 2050000-step training.



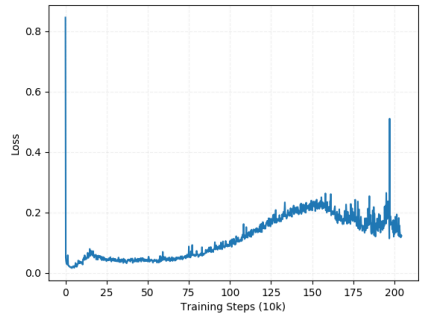
(a) Average test return per episode as a function of training step



(b) Average PM counts per episode as a function of training steps



(c) Average Random failures per episode counts as a function of training steps



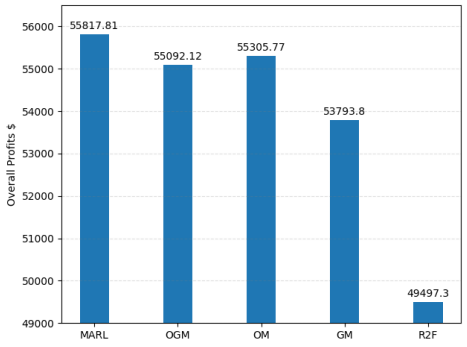
(d) DQN Loss as a function of training steps

Figure 4.7: RL Training Monitoring

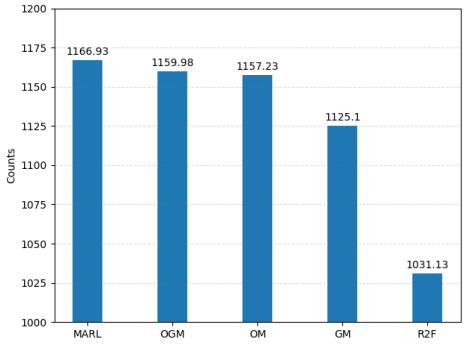
We conduct the same analysis on RL training. Figure 4.7a depicts that the test return oscillates drastically during the course of the training. Figure 4.7c shows that the RL agent learns to reduce average random failure counts at the end of the training, by excessively conducting PMs. Our further investigation on loss shows that the loss gradually increases during the course of training. Therefore, we conclude RL

policy fails to converge for this multi-level PM task. We think that this is due to the fact that the RL agent deals with an action space of size $3^6 = 729$.

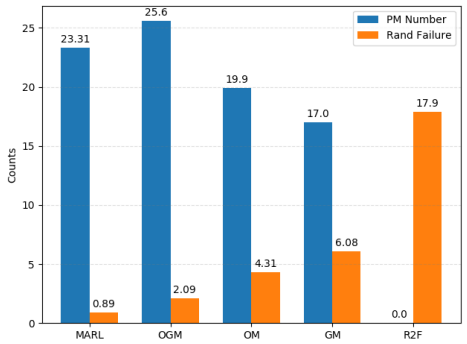
Test Benchmark: We also benchmark the saved MARL policy with other baseline policies (Note that Q-learning policy is excluded from the benchmark since it fails to converge). During the test, the initial states of machines are randomly generated by the simulation. To account for the randomness resulted from the states initialization, each method is tested for 50 episodes independently and the average performance statistics are reported. Figure 4.8 encompasses a set of comprehensive metrics that are used for test evaluation, including profit per episode, throughput numbers per episode, PM counts per episode, random failure per episode, PM cost per episode, and CM cost per episode.



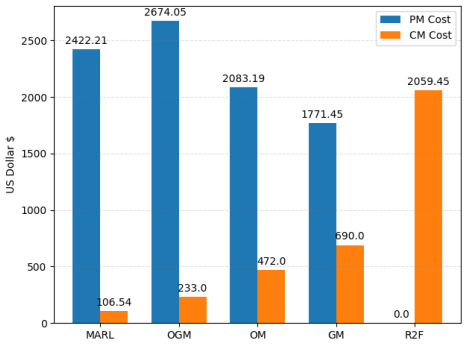
(a) Average profit per episode by methods



(b) Average number of throughputs per episode by methods



(c) Average number of PM and random failure per episode by methods



(d) Average cost of PM and CM per episode by methods

Figure 4.8: Evaluation Results

In general, all methods other than R2F are effective policies. This is because they achieved higher average profit over R2F, as shown in figure 4.8a. Among the effective policies, MARL policy reports the best average profit and throughput per

episode. OM policy perform slightly better than OGM in profit although the OM policy reports less throughput than OGM. And GM has the least profit compared with other 3 effective policies.

Table 4.3: Maintenace Cost Summary

	MARL	OGM	OM	GM	R2F
PM Cost ($)	2422.21	2674.05	2083.19	1771.45	0.00
CM Cost ($)	100.54	233.00	472.00	690.00	2059.45
Overall Cost ($)	2522.75	2907.05	2555.19	2461.45	2059.45

Metrics such as average PM counts and random failures are scrutinized to understand policies. Recall that the goal of this study is to find policies that ensure the smooth operation of the serial production line. Namely, the learned policy should mitigate the stoppage caused by random failures while not committing too many maintenance actions. The R2F policy, which passively performs CM, has the least average maintenance cost per episode of $\$2095.45$. However, it reports the least profit and throughput due to the production loss and CM cost. The OM policy is more aggressive compared with R2F in terms of performing PM. It reduces random failure from R2F's 17.9 to 6.08 per episode. As a result, its profit and throughput improve by 8.68% and 9.11%, respectively, compared with R2F. The variable τ does make OGM and GM more adaptive than R2F as is reflected by the decreasing random failure counts. OGM's PM policy is more aggressive than GM as a result of the opportunistic maintenance performed by OGM. In comparison with OM, OGM conducts 5.7 more PMs per episode and encountered 2.22 less random failures on average. And OGM spends 351.86 more US dollars on maintenance. As a consequence, although OGM produces 2.75 more products per episode than OM, it generates less profit. Compared with OGM, MARL reduces both the number of PMs and random failures. In the comparison with OM, MARL is able to perform 3.41 more PMs while incurring $\$32.44$ less overall cost.

To closely examine the learned MARL policy, we roll out an episode with random state initialization, using the learned MARL policy. Partial record that contains consecutive maintenance is shown in Tables 4.4 and 4.5.

At $t = 649$, the MARL policy conducts PM on machine 6 at an age of 497. At $t = 750$, machine 4 conducts PM at an age of 451. At $t = 849$, machine 1 goes into PM at an age of 287. The MARL policy is likely to memorize the likely break-down age for different machines and conduct timely PMs for them. This observation explains why MARL has low random failure and low maintenance cost on average as shown in Figure 4.8.

At $t = 1221$, machines 4 and 5 conduct PM simultaneously. Note that machine

Table 4.4: Partial Maintenance Record

Time (min)	Failed machines	Machine ages $g_a(t)$	Buffer levels
649	-	$[89, 57, 267, 341, 197, 497]$	$[7, 10, 0, 0, 0]$
759	-	$[199, 167, 377, 451, 307, 101]$	$[8, 10, 0, 0, 0]$
814	-	$[254, 222, 432, 44, 362, 156]$	$[7, 10, 3, 0, 0]$
847	-	$[287, 255, 25, 77, 395, 189]$	$[8, 10, 0, 0, 0]$
858	-	$[1, 266, 36, 88, 406, 200]$	$[0, 10, 0, 0, 0]$
891	-	$[34, 299, 69, 121, 21, 233]$	$[6, 10, 3, 6, 0]$
1144	-	$[287, 244, 322, 374, 274, 486]$	$[8, 10, 0, 0, 0]$
1155	-	$[1, 255, 333, 385, 285, 497]$	$[0, 9, 0, 0, 0]$
1188	-	$[34, 288, 366, 418, 318, 24]$	$[5, 10, 0, 0, 5]$
1221	-	$[67, 24, 399, 451, 351, 57]$	$[7, 9, 0, 0, 0]$

5 conducts PM at an age of 406 at $t = 858$ whereas it conducts PM at an age of 351 when $t = 1221$. The PM at $t = 1221$ can be seen as an "group maintenance" as machine 5 accommodates its maintenance schedule to that of machine 4.

Interestingly, the MARL agents favor perfect PM over imperfect PM. This is due to the fact that the cost efficiency (the amount of money spent to reset 1 unit of age) of imperfect PMs is much lower than perfect PMs. In contrast, baselines such OM, GM, and OGM, have trouble capturing the dynamics of the environment and favor imperfect PMs. This is because those methods cannot perform timely PM and because perfect PMs incur higher cost and longer machine down time. In summary, examination of the MARL policy reveals that it is able to schedule PM strategically; that is, the MARK policy conducts group maintenance and captures environment dynamics.

4.5.5 Additional Experiments

To demonstrate the scalability of our proposed framework, we extend the previous serial production line by adding 4 more machines. Parameters for the additional machines and buffers can be found in Tables 4.6 and 4.7.

The training process described in the previous section is used here as well. Note that the DQN agent now faces $3^{10} = 59049$ actions whereas the action space of MARL agents is invariant to the number of agents. Since the memory of DQN is set to 750000 units, each unit stores the action u_t taken by the RL agent and the action mask m_t that keeps track of valid actions, with u_t and m_t being 59049-element vectors. The RAM in our hardware fails to allocate enough memory for DQN's

Table 4.5: Partial Maintenance Record (continue)

Time (min)	Action $\mathbf{u}_t$	PM	GM
649	$[0, 0, 0, 0, 0, 2]$	Y	
759	$[0, 0, 0, 2, 0, 0]$	Y	
814	$[0, 0, 2, 0, 0, 0]$	Y	
847	$[2, 0, 0, 0, 0, 0]$	Y	
858	$[0, 0, 0, 0, 2, 0]$	Y	
891	$[0, 2, 0, 0, 0, 0]$	Y	
1144	$[2, 0, 0, 0, 0, 0]$	Y	
1155	$[0, 0, 0, 0, 0, 2]$	Y	
1188	$[0, 2, 0, 0, 0, 0]$	Y	
1221	$[0, 0, 0, 2, 2, 0]$	Y	Y

Table 4.6: Machine Parameters for Additional Machines

Parameters	M_7	M_8	M_9	M_{10}
Cycle Time T_a (min)	1.20	1.30	1.10	1.05
Level 1 PM cost $c_a^{PM_1}$ (\$)	55.0	60.0	45.0	50.0
Level 2 PM cost $c_a^{PM_2}$ (\$)	110.0	120.0	90.0	103.0
Level 1 PM duration $d_a^{PM_1}$ (min)	4	5	6	4
Level 2 PM duration $d_a^{PM_2}$ (min)	8	11	12	9
CM cost c_a^{CM} (\$)	120.0	130.0	110.0	125.0
CM duration d_a^{CM} (min)	25	32	24	25
Level 1 PM degree $r_a^{PM_1}$	0.8	0.8	0.8	0.8
Level 2 PM degree $r_a^{PM_2}$	0.8	0.8	0.8	0.8
Scale parameter α_a	500	400	390	470
Shape parameter β_a	15	15	15	15

Table 4.7: Buffer Parameters for Additional Buffers

Parameters	B_6	B_7	B_8	B_9
Buffer capability b_i	10	12	9	8

replay buffer. Note that the memory needed for the replay buffer grows drastically with the number of agents or actions, making the DQN impractical to deal with large and complex manufacturing systems. As for other baselines, Monte Carlo simulations are rolled out to find optimal τ and choice of PM action. OM, OGM, and GM still always take level-1 (imperfect) PM; The optimal τ is 280 for GM and 290 for OGM.

Figure 4.9 depicts the training of MARL agents in the extended 10-machine-9-buffer serial production line. The test return remains consistent until approximately 1.6 million training steps. Compared to the training process in the previous section, it takes more training steps for the agent to converge to good PM policies. This is due to the fact that the dynamics of the environment is further complicated by the additional 4 machines. Thus, the difficulty of deriving good PM policies also increases.

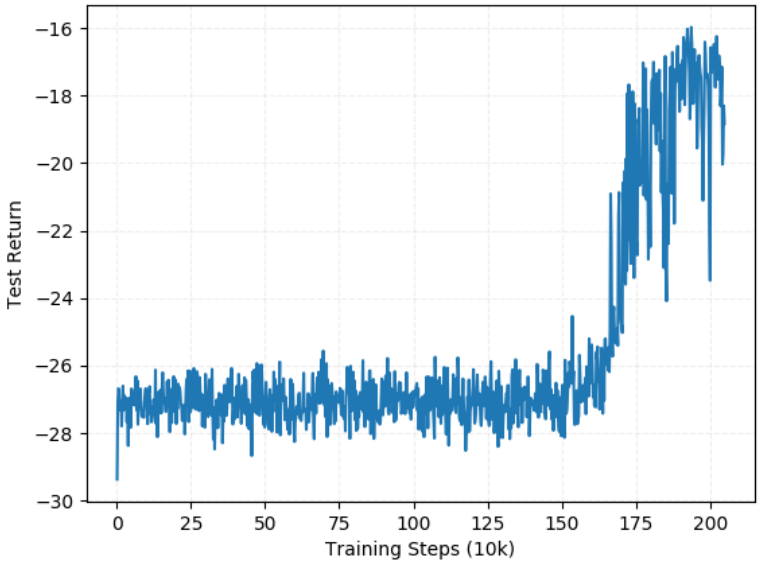
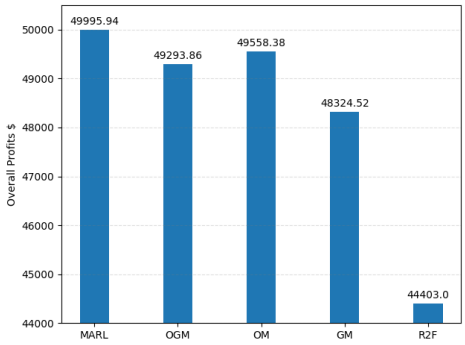


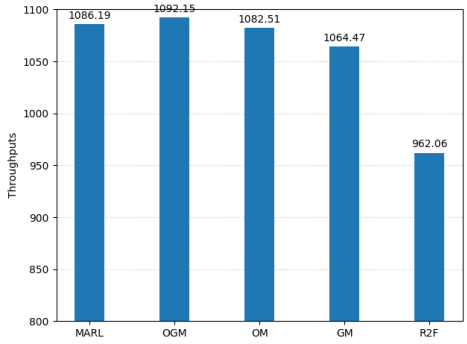
Figure 4.9: Average test return per episode as a function of training step

Consistent with the findings in the previous experiment, all policies except R2F are effective policies and the MARL policy continues to achieve the best average profit. As shown in Figures 4.10c and 4.10a, excessive PM does not necessarily lead to good profit. For instance, the MARL policy reports more random failures than OGM. However, OGM achieves low random failure by excessively conducting PMs. Consequently, the MARL policy spends $\$1000.68$ less in overall maintenance than OGM on average. It is obvious that all baselines except R2F have a tendency to commit over-PM leading to an increase in a machine's downtime and an increase in production loss. This demonstrates that the proposed MARL policies can find the trade-off between production loss and PMs.

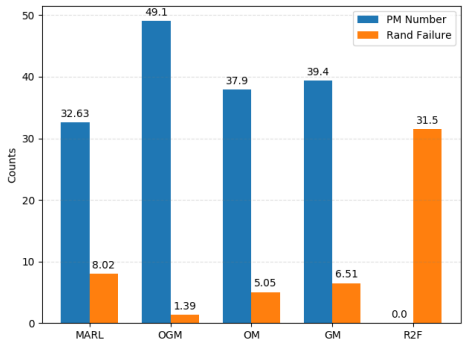
We also examine the learned policy in the new environment by rolling out an episode with random initialization. Partial records that contain consecutive maintenance are shown in Tables 4.8 and 4.9. Consistent with the policy examination in



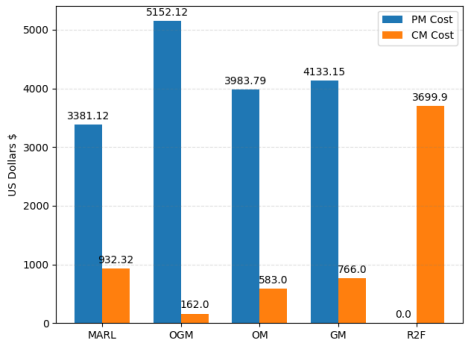
(a) Average profit per episode by methods



(b) Average number of throughputs per episode by methods



(c) Average number of PM and random failure per episode by methods



(d) Average cost of PM and CM per episode by methods

Figure 4.10: Evaluation Results for 10-machine-9-buffer Serial Production Line

the previous experiment, the learned policy in the new environment can also perform timely PM and favors perfect PM over imperfect PM. Additionally, we observe increasing occurrences of opportunistic maintenance and group maintenance, i.e. the MARL agent is learning these maintenance strategies through interaction with the system. For instance, machine 1 encounters random failure and goes through CM at time 139. At $t = 143$, machine 5 conducts PM while machine 1 is still under maintenance (CM is longer than PM in general). Machine 5's PM that takes the advantage of unscheduled failure on other machines is said to be opportunistic maintenance. The same incident can be observed at time 605. At $t = 385$ and $t = 506$, group maintenance is conducted, respectively.

Table 4.8: Partial Maintenance Record

Time (min)	Failed machines	Maint type	Machine ages $g_a(t)$
139	M_1	CM	$[340, 130, 468, 128, 254, 130, 174, 84, 116, 75]$
143	-	PM	$[0, 134, 472, 132, 258, 134, 178, 88, 120, 79]$
160	M_2	CM	$[0, 151, 489, 149, 6, 151, 195, 105, 137, 96]$
297	-	PM	$[129, 288, 111, 286, 142, 288, 332, 242, 274, 233]$
308	-	PM	$[140, 299, 122, 297, 153, 299, 3, 253, 285, 244]$
330	-	PM	$[162, 321, 144, 319, 175, 321, 25, 275, 10, 266]$
385	-	PM	$[217, 46, 199, 374, 230, 376, 80, 330, 65, 321]$
396	-	PM	$[228, 57, 210, 0, 241, 387, 91, 0, 76, 332]$
506	-	PM	$[338, 167, 320, 110, 351, 101, 201, 110, 186, 442]$
600	M_1	CM	$[432, 261, 414, 204, 82, 195, 295, 204, 280, 85]$
605	-	PM	$[0, 266, 419, 209, 87, 200, 300, 209, 285, 90]$

4.6 Conclusion

Multi-level PM scheduling in a serial production line is challenging due to the explosion of action space and the non-linear, stochastic nature of the system. We propose to model the PM decision making process in serial production lines as Dec-POMDP and demonstrate how to implement MARL to obtain PM policies. We utilize two numerical experiments to further establish the necessity of modeling the PM decision-making as multi-agent problems instead of as a single-agent problem. In the 6-machine-5-buffer experiment, the DQN policy exhibits convergence issues while the MARL policy achieves the best profit among all baselines. In the 10-machine-9-buffer experiment, the DQN method cannot be implemented efficiently due to the increasing size of its replay buffer and action space. By contrast, MARL does not suffer from this issue and continues to report the best average profit among all policies.

4.7 Related Work

Machine maintenance policies have been studied under numerous application scenarios and are characterized by the target system. Based on the structure of the studied system, the current literature can be categorized into single-unit policies and multi-unit policies.

Single-unit policies provide maintenance policies for one-unit systems. The

Table 4.9: Partial Maintenance Record (continue)

Time (min)	Action $\mathbf{u}_t$	OG	GM
139	$[0,0,0,0,0,0,0,0,0,0]$		
143	$[0,0,0,0,2,0,0,0,0,0]$	Y	
160	$[0,0,0,0,0,0,0,0,0,0]$		
297	$[0,0,0,0,0,0,2,0,0,0]$		
308	$[0,0,0,0,0,0,0,0,2,0]$		
330	$[0,2,0,0,0,0,0,0,0,0]$		
385	$[0,0,0,2,0,0,0,2,0,0]$		Y
396	$[0,0,0,0,0,2,0,0,0,0]$		
506	$[0,0,0,0,2,0,0,0,0,2]$		Y
600	$[0,0,0,0,0,0,0,0,0,0]$		
605	$[0,0,0,0,0,0,0,0,2,0]$	Y	

relation between the maintenance decisions and system maintenance cost is usually known in such systems. Therefore, it is common to model single-unit maintenance as a stochastic process, where the optimal solution can be solved to achieve certain objectives (e.g. minimizing the maintenance cost rate [59, 64, 97], maximizing the machine availability [14], maximizing machine reliability [61], etc.). Examples are age-dependent policies [90] and repair limit policies [54]. While a multi-unit system can be reduced to multiple single-unit systems, a generalization of single-unit policies to multi-unit systems is questionable due to its rigorous assumption that there is no structural dependencies between subsystems [89]. Therefore, single-unit policies are not suitable for serial production lines.

In comparison, multi-unit policies that consider dependencies between subsystems are more suitable to solve maintenance problems in large-scale production systems because the structural and operational dependencies among machines are often considered in this line of research. However, maintenance policies are often derived by exploiting the specific structure of the system, such as serial systems [25], parallel systems [8], and k-out-of-n systems [22] etc, thus are often limited to certain scenarios. Most of the multi-unit policies are based on the fundamental idea of group and opportunistic maintenance, which are inspired by the observation that the maintenance of one subsystem does not incur extra production loss if another subsystem is already under maintenance [89]. For instance, [69,77] propose a group maintenance policy to conduct multiple PM simultaneously, thus avoiding incurring additional production loss. Opportunistic maintenance policies [1, 55, 96] aim to derive a time window where insertions of PM do not result in extra production loss. In addition, those heuristic-based methods tend to consider less general production systems, where buffers among machines are not considered, thus neglecting the

delays of machine state propagation caused by buffers [100].

Recent studies that do not assume heuristics of GM and OM policies often adopt the approach of reinforcement learning, by modeling the maintenance problems as MDPs [5,38,74], semi-Markov decision processes [13,86], and POMDPs [3,11,12]. Most of these works either assume that a model is given or only derive maintenance policies for simplified multi-unit systems, which impedes those methods from be generalized to realistic scenarios. Karamatsoukis et al. assume the model of the system and derives a PM policy for a two-machine-one-buffer manufacturing system using dynamic programming (DP) [45]. Wang et al. model a two-machine-one-buffer manufacturing system as semi-MDP [91]. Ramirez et al. propose a simulation-based approximate dynamic programming approach for the optimization of PM scheduling decisions in semiconductor manufacturing systems, while suffering from the assumption that the line is perfectly balanced [72]. Arab et al. derive maintenance policies on systems with simplified dynamics, i.e. the maintenance schedule does not affect machine reliability status [6]. Choo et al. derives a hierarchical maintenance policy for parallel production lines without buffers [18]. Huang et al., which derives model-free Q-learning maintenance policies for a six-machine-five-buffer serial production [38], represents the work most closely related to this paper. Nonetheless, their PM policies do not consider imperfect maintenance effects and have issues scaling to larger manufacturing systems.

Chapter 5

Closing Remarks

This dissertation presents new machine learning techniques as well as their applications on Cyber-Physical Systems.

In the chapter 2, we identify that the current car-following models require inputs with fixed-size and fixed spatial organization, which limits the generalization of those methods. Instead of simply expanding the input size of car-following models, we propose a graph-based method that enables the ego-vehicle to learn from a flexible number of neighbors in a hierarchical fashion. Specifically, we propose the use of graphs defined by the spatial relationships between vehicles, to model traffic. We further adpat the original GCN to incorporate our assumptions about drivers. Coupled with the LSTM, our resulting frameworks can learn both temporal and spatial information for ego-vehicles. In our simulation studies, the proposed frameworks outperform others on the task of acceleration prediction.

While the proposed graph method can be used to predict ego-vehicle's trajectories in a short horizon, it can not be used for ego-vehicle's control as the prediction error accumulates during vehicle's traveling course. In the chapter 3, we study reinforcement learning that can be applied for control in multi-agent systems. We propose a new credit-assignment actor-critic framework, value-decomposition multi-agent actor-critic (VDAC), that enforces the monotonic relationship between the global state-value and the shaped local state-value. Theoretically, we establish the convergence of the proposed actor-critic method to a local optimal. Empirically, benchmark tests on StarCraft II micromanagement games demonstrate that our proposed actor-critic bridges the performance gap between multi-agent actor-critics and Q-learning, and our method provides a balanced trade-off between training efficiency and performance.

The chapter 4 presents a VDAC-based adaptive preventive maintenance scheduling framework. Current methods either suffer from the nonlinear, stochastic nature of manufacturing systems, or the issue of action space explosion. By formulating

the scheduling problem into a cooperative multi-agent problem, we are able to alleviate the aforementioned issues facing the traditional and centralized methods. We utilize two numerical experiments to further establish the necessity of modeling the PM decision-making as multi-agent problems instead of as a single-agent problem. In the 6-machine-5-buffer experiment, the DQN policy exhibits convergence issues while the MARL policy achieves the best profit among all baselines. In the 10-machine-9-buffer experiment, the DQN method cannot be implemented efficiently due to the increasing size of its replay buffer and action space. By contrast, MARL does not suffer from this issue and continues to report the best average profit among all policies.

While the research studies in this dissertation advance the corresponding subfields, they suffer from the following limitations:

- For the first chapter, the criterion to construct traffic graphs are not established. For instance, the neighbor distance τ is selected empirically for highway driving. In practice, optimal τ might exist for different driving scenarios such as highway driving, city driving, etc.

- VDAC utilizes the non-negative mixing network to assign credits among agents. However, in-depth analysis of the discrepancy between assigned credits and the ground truth is not conducted, which might help us understand the mixing network.

- The proposed framework is only tested on serial production lines, where we can directly model each machine as an agent. For complex production systems with multiple lines interconnected, we might need to resort to other modeling techniques such as modeling a group of machines as a single agent. In addition, the limitation of resources is not considered in our research.

Chapter 6

Appendix

In Chapter 3, we use all the default settings in [76]. That includes: the game difficulty is set to level 7, *very difficult*, the shoot range, observe range, etc, are consistent with the default settings. The action space of agents consists of the following set of discrete actions: move[direction], attack[enemy id], stop, and no operation. Agents can only move in four directions: north, south, east, or west. A unit is allowed to perform the attack[enemy id] action only if the enemy is within its shooting range.

Each unit has a sight range that limits its ability to receive any information out of range. The sight range, which is bigger than shooting range, makes the environment partially observable from the standpoint of each agent. Agents can only observe other agents if they are both alive and located within the sight range. The global state, which is only available to agents during centralised training, encapsulates information about all units on the map.

The observation vector also follows the default implementation in [76]: It contains the following attributes for both allied and enemy units within the sight range: distance, relative x, relative y, health, shield, and unit type. In addition, the observation vector includes the last actions of allied units that are in the field of view. Lastly, the terrain features, in particular the values of eight points at a fixed radius indicating height and walkability, surrounding agents within the observe range are also included. The state vector includes the coordinates of all agents relative to the center of the map, together with units' observation feature vectors. Additionally, the energy of Medivacs and cooldown of the rest of the allied units are stored in the state vector. Finally, the last actions of all agents are attached to the state vector.

6.0.1 Training Details and Hyperparameters

Experiments are obtained by using Nvidia RTX 2080 Ti graphics cards, with each independent run taking 1 to 3 hours depending on the scenario. Each independent run corresponds to a unique random seed that is randomly generalized at the beginning.

Table 6.1: Map Descriptions.

Map Name	Ally Units	Enemy Units
2s_vs_1sc	2 Stalkers	1 Spine Crawler
8m	8 Marines	8 Marines
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
1c3s5z	1 Colossus, 3 Stalkers & 5 Zealots	1 Colossus, 3 Stalkers & 5 Zealots
bane_vs_bane	20 Zerglings & 4 Banelings	20 Zerglings & 4 Banelings

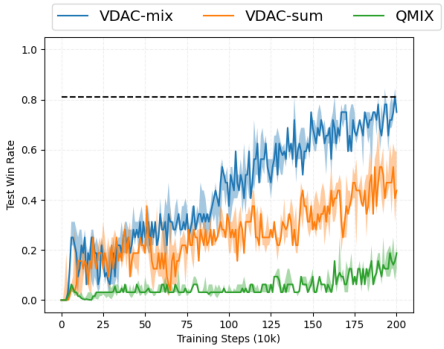
The agent networks of all algorithms resemble a DRQN [35] with a recurrent layer comprised of a GRU [19] with a 64-dimensional hidden state, with a fully-connected layer before and after. The exception is that IAC, VDAC-sum, and VDAC-mix agent networks contain an additional layer to output local state values and the policy network outputs a stochastic policy rather than action-values.

Algorithms are trained with RMSprop with learning rate 5×10^{-4}. During training, 8 games are initiated independently, from which episodes are sampled. Q-learning replay buffer stores the latest 5000 episodes for each independent game (In total, replay buffer has a size of $8 \times 5000 = 40000$). We set $\gamma = 0.99$ and $\lambda = 0.8$ (if needed). Target networks (if exists) are updated every 200 training steps.

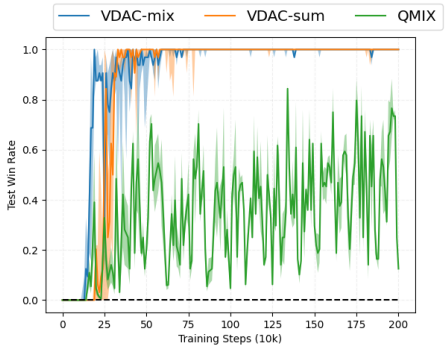
The architecture of the COMA critic is a feedforward fully-connected neural network with the first 2 layers, each of which has 128 units, followed by a final layer of $|U|$ units. Naive central critic shares the same architecture with COMA critic with an exception that its final layer contains 1 units.

The mixing network in QMIX and VDAC-mix shares an identical structure. It consists of a single hidden layer of 32 units, whose parameters are outputted by hypernetworks. An ELU activation function follows the hidden layer in the mixing network. The hypernetworks consist of a feedforward network with a single hidden layer of 64 units with a ReLU activation function.

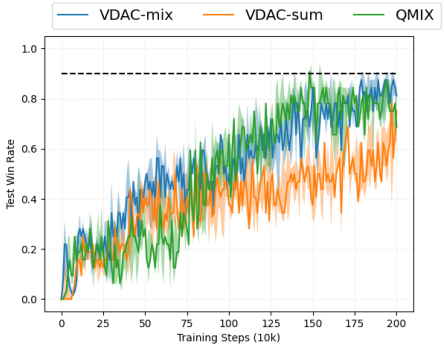
6.0.2 StarCraft II Results



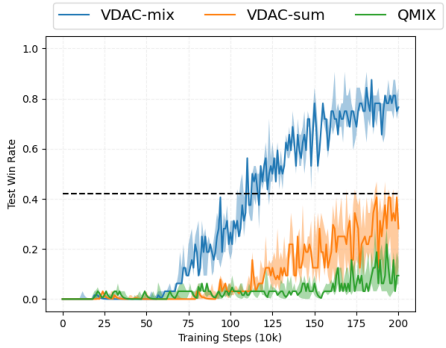
(a) 1c3s5z



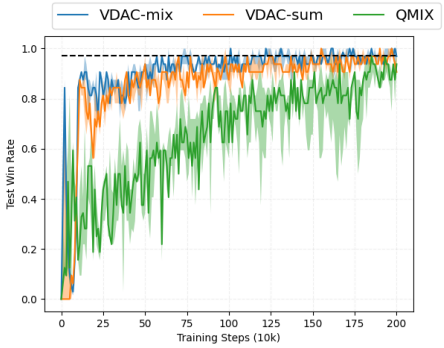
(b) 2s_vs_1sc



(c) 2s3z

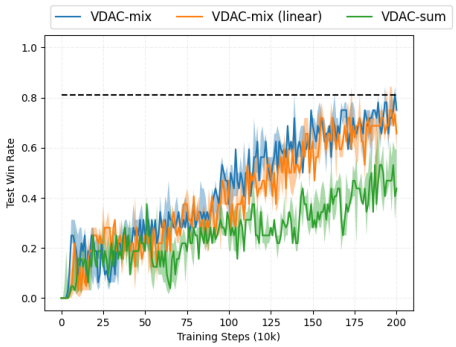


(d) 3s5z

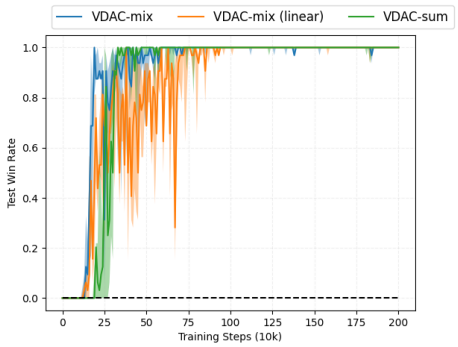


(e) 8m

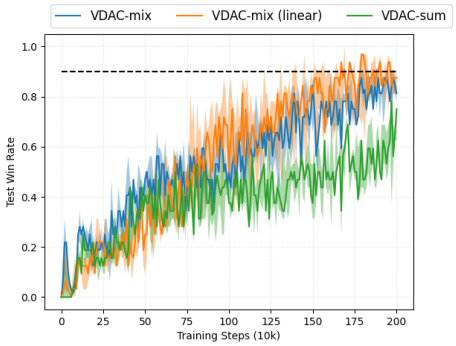
Figure 6.1: Overall results: VDACs vs QMIX under A2C



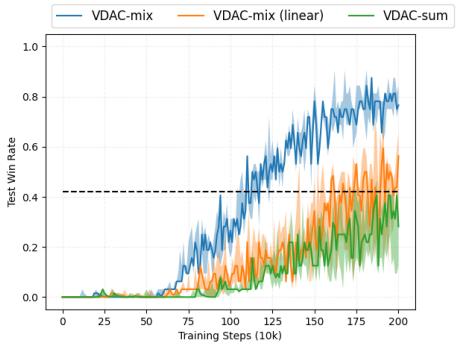
(a) 1c3s5z



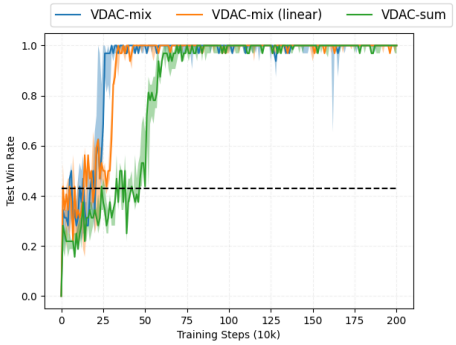
(b) 2s_vs_1sc



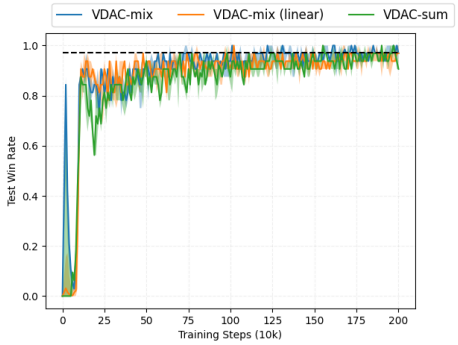
(c) 2s3z



(d) 3s5z



(e) bane_vs_bane



(f) 8m

Figure 6.2: Overall results: VDAC-mix vs VDAC-mix(linear) vs VDAC-sum

List of Figures

List of Tables

Bibliography

[1] Hasnida Ab-Samat and Shahrul Kamaruddin. Opportunistic maintenance (om) as a new advancement in maintenance approaches: A review. *Journal of Quality in Maintenance Engineering*, 20(2):98–121, 2014.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[3] Mohammad M AlDurgam and Salih O Duffuaa. Optimal joint maintenance and operation policies to maximise overall systems effectiveness. *International Journal of Production Research*, 51(5):1319–1330, 2013.

[4] Florent Altché and Arnaud de La Fortelle. An LSTM network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359. IEEE, 2017.

[5] Suprasad V Amari, Leland McLaughlin, and Hoang Pham. Cost-effective condition-based maintenance using markov decision processes. In *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.*, pages 464–469. IEEE, 2006.

[6] Ali Arab, Napsiah Ismail, and Lai Soon Lee. Maintenance scheduling incorporating dynamics of production system and real-time information from workstations. *Journal of Intelligent Manufacturing*, 24(4):695–705, 2013.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Anne Barros, Antoine Grall, and Christophe Bérenguer. Joint modelling and optimization of monitoring and maintenance performance for a two-unit

parallel system. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 221(1):1–11, 2007.

[9] Christopher M Bishop. Mixture density networks. *Technical report*, 1994.

[10] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[11] Eunshin Byon and Yu Ding. Season-dependent condition-based maintenance for a wind turbine using a partially observed markov decision process. *IEEE Transactions on Power Systems*, 25(4):1823–1834, 2010.

[12] Eunshin Byon, Lewis Ntaimo, and Yu Ding. Optimal maintenance strategies for wind turbine systems under stochastic weather conditions. *IEEE Transactions on Reliability*, 59(2):393–404, 2010.

[13] GK Chan and Sohrab Asgarpoor. Optimum maintenance policy with markov processes. *Electric power systems research*, 76(6-7):452–456, 2006.

[14] J-K Chan and Leonard Shaw. Modeling repairable systems with failure rates that depend on age and maintenance. *IEEE Transactions on Reliability*, 42(4):566–571, 1993.

[15] Robert E Chandler, Robert Herman, and Elliott W Montroll. Traffic dynamics: studies in car following. *Operations research*, 6(2):165–184, 1958.

[16] Djabir Abdeldjalil Chekired, Lyes Khoukhi, and Hussein T Mouftah. Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory. *IEEE Transactions on Industrial Informatics*, 14(10):4590–4602, 2018.

[17] T Chitra. Life based maintenance policy for minimum cost. In *Annual Reliability and Maintainability Symposium, 2003.*, pages 470–474. IEEE, 2003.

[18] Benjamin Y Choo, Stephen Adams, and Peter Beling. Health-aware hierarchical control for smart manufacturing using reinforcement learning. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 40–47. IEEE, 2017.

[19] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[20] J Colyar and J Halkias. US highway 80 dataset," Federal Highway Administration (FHWA), vol. *Tech, no. Rep*, 2006.

[21] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on Machine Learning*, pages 1538–1546, 2019.

[22] Karin S de Smidt-Destombes, Matthieu C van der Heijden, and Aart van Harten. Joint optimisation of spare part inventory, maintenance frequency and repair capacity for k-out-of-n systems. *International Journal of Production Economics*, 118(1):260–268, 2009.

[23] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. Graph neural networks for modelling traffic participant interaction. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 695–701. IEEE, 2019.

[24] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating functional knowledge in neural networks. *Journal of Machine Learning Research*, 10(6), 2009.

[25] V Ebrahimipour, A Najjarbashi, and Mohammad Sheikhalishahi. Multi-objective modeling for preventive maintenance scheduling in a multiple production line. *Journal of Intelligent Manufacturing*, 26(1):111–122, 2015.

[26] Mohamed-Chahir Fitouhi, Mustapha Nourelfath, and Stanley B Gershwin. Performance evaluation of a two-machine line with a finite buffer and condition-based maintenance. *Reliability Engineering & System Safety*, 166:61–72, 2017.

[27] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in neural information processing systems*, pages 2137–2145, 2016.

[28] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.

[29] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[30] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 6530–6539, 2017.

[31] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[32] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[33] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*, 2017.

[34] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[35] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[36] Walter Helly. Simulation of bottlenecks in single-lane traffic flow. *In International Symposium on the Theory of Traffic Flow, New York*, 1959.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[38] Jing Huang, Qing Chang, and Jorge Arinez. Deep reinforcement learning based preventive maintenance policy for serial production lines. *Expert Systems with Applications*, 160:113701, 2020.

[39] Jing Huang, Qing Chang, Jorge Arinez, and Guoxian Xiao. A maintenance and energy saving joint control scheme for sustainable manufacturing systems. *Procedia CIRP*, 80:263–268, 2019.

[40] Jing Huang, Qing Chang, and Nilanjan Chakraborty. Machine preventive replacement policy for serial production lines based on reinforcement learning. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 523–528. IEEE, 2019.

[41] Jing Huang, Qing Chang, Jing Zou, and Jorge Arinez. A real-time maintenance policy for multi-stage manufacturing systems considering imperfect maintenance effects. *IEEE Access*, 6:62174–62183, 2018.

[42] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[43] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970, 2019.

[44] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in neural information processing systems*, pages 7254–7264, 2018.

[45] CC Karamatsoukis and EG Kyriakidis. Optimal maintenance of two stochastically deteriorating machines with an intermediate buffer. *European Journal of Operational Research*, 207(1):297–308, 2010.

[46] Masaaki Kijima. Some results for repairable systems with general repair. *Journal of Applied probability*, pages 89–102, 1989.

[47] Shinya Kikuchi and Partha Chakroborty. Car-following model based on fuzzy inference system. *Transportation Research Record*, pages 82–82, 1992.

[48] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404. IEEE, 2017.

[49] Dae-Young Kim, Seokhoon Kim, Houcine Hassan, and Jong Hyuk Park. Adaptive data rate control in low power wide area networks for long range iot services. *Journal of computational science*, 22:171–178, 2017.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[52] E Kometani. Dynamic behavior of traffic with a nonlinear spacing-speed relationship. *Theory of Traffic Flow (Proc. of Sym. on TTF (GM))*, pages 105–119, 1959.

[53] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[54] Hirofumi Koshimae, Tadashi Dohi, Naoto Kaio, and Shunji Osaki. Graphical/statistical approach to repair limit replacement problem. *Journal of the Operations Research Society of Japan*, 39(2):230–246, 1996.

[55] Radouane Laggoune, Alaa Chateauneuf, and Djamil Aissani. Opportunistic policy for optimal preventive maintenance of a multi-component system in continuous operating units. *Computers & Chemical Engineering*, 33(9):1499–1510, 2009.

[56] Stéphanie Lefèvre, Chao Sun, Ruzena Bajcsy, and Christian Laugier. Comparison of parametric and non-parametric approaches for vehicle speed prediction. In *2014 American Control Conference*, pages 3494–3499. IEEE, 2014.

[57] David Lenz, Frederik Diehl, Michael Truong Le, and Alois Knoll. Deep neural networks for markovian interactive scene prediction in highway scenarios. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 685–692. IEEE, 2017.

[58] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[59] CE Love and R Guo. Utilizing weibull failure rates in repair limit analysis for equipment replacement/preventive maintenance decisions. *Journal of the operational Research Society*, 47(11):1366–1376, 1996.

[60] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

[61] Mazhar Ali Khan Malik. Reliable preventive maintenance scheduling. *AIIE transactions*, 11(3):221–228, 1979.

[62] RM Michaels. Perceptual factors in car-following. *Proc. of 2nd ISTTF (London)*, pages 44–59, 1963.

[63] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[64] Amit Monga, Ming J Zuo, and Roger W Toogood. Reliability-based design of systems considering preventive maintenance and minimal repair. *International Journal of Reliability, Quality and Safety Engineering*, 4(01):55–71, 1997.

[65] Marcello Montanino and Vincenzo Punzo. Making NGSIM data usable for studies on traffic flow theory: Multistep method for vehicle trajectory reconstruction. *Transportation Research Record*, 2390(1):99–111, 2013.

[66] Marcello Montanino and Vincenzo Punzo. Trajectory data reconstruction and simulation-based validation against macroscopic traffic patterns. *Transportation Research Part B: Methodological*, 80:82–106, 2015.

[67] Jeremy Morton, Tim A Wheeler, and Mykel J Kochenderfer. Analysis of recurrent neural networks for probabilistic modeling of driver behavior. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1289–1298, 2016.

[68] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *Advances in Neural Information Processing Systems*, pages 8102–8113, 2018.

[69] Robin P Nicolai and Rommert Dekker. Optimal maintenance of multi-component systems: a review. In *Complex system maintenance handbook*, pages 263–286. Springer, 2008.

[70] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[71] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2, 2017.

[72] José A Ramírez-Hernández and Emmanuel Fernandez. Optimization of preventive maintenance scheduling in semiconductor manufacturing models using a simulation-based approximate dynamic programming approach. In *49th IEEE Conference on Decision and Control (CDC)*, pages 3944–3949. IEEE, 2010.

[73] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[74] Charles-Antoine Robelin and Samer M Madanat. History-dependent bridge deck maintenance and replacement optimization with markov decision processes. *Journal of Infrastructure Systems*, 13(3):195–201, 2007.

[75] Tony Robinson, Mike Hochberg, and Steve Renals. The use of recurrent neural networks in continuous speech recognition. In *Automatic speech and speaker recognition*, pages 233–258. Springer, 1996.

[76] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[77] Mahmood Shafiee and Maxim Finkelstein. An optimal age-based group maintenance policy for multi-unit degrading systems. *Reliability Engineering & System Safety*, 134:230–238, 2015.

[78] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[79] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1905.05408*, 2019.

[80] Jianyu Su, Stephen Adams, and Peter A Beling. Counterfactual multi-agent reinforcement learning with graph convolution communication. *arXiv preprint arXiv:2004.00470*, 2020.

[81] Jianyu Su, Stephen Adams, and Peter A Beling. Value-decomposition multi-agent actor-critics. *arXiv preprint arXiv:2007.12306*, 2020.

[82] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[83] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[84] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[85] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[86] Curtis L Tomasevicz and Sohrab Asgarpoor. Optimum maintenance policy using semi-markov decision processes. In *2006 38th North American Power Symposium*, pages 23–28. IEEE, 2006.

[87] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

[88] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[89] Hongzhou Wang. A survey of maintenance policies of deteriorating systems. *European journal of operational research*, 139(3):469–489, 2002.

[90] Hongzhou Wang and Hoang Pham. Some maintenance models and availability withimperfect maintenance in production systems. *Annals of Operations Research*, 91:305–318, 1999.

[91] Xiao Wang, Hongwei Wang, and Chao Qi. Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system. *Journal of Intelligent Manufacturing*, 27(2):325–333, 2016.

[92] Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. Multiagent soft q-learning. In *2018 AAAI Spring Symposium Series*, 2018.

[93] Ying Wen, Yaodong Yang, Rui Luo, Jun Wang, and Wei Pan. Probabilistic recursive reasoning for multi-agent reinforcement learning. *arXiv preprint arXiv:1901.09207*, 2019.

[94] Marco A Wiering. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[95] David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pages 355–369. World Scientific, 2002.

[96] Tangbin Xia, Xiaoning Jin, Lifeng Xi, and Jun Ni. Production-driven opportunistic maintenance for batch production based on mam–apb scheduling. *European Journal of Operational Research*, 240(3):781–790, 2015.

[97] Xitong Zheng and Nasser Fard. A maintenance policy for repairable systems based on opportunistic failure-rate tolerance. *IEEE Transactions on Reliability*, 40(2):237–244, 1991.

[98] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[99] Jing Zou, Qing Chang, Jorge Arinez, Guoxian Xiao, and Yong Lei. Dynamic production system diagnosis and prognosis using model-based data-driven method. *Expert Systems with Applications*, 80:200–209, 2017.

[100] Jing Zou, Qing Chang, Yong Lei, and Jorge Arinez. Production system performance identification using sensor data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(2):255–264, 2016.