Hasso Plattner Institute
Internet-Technology and Softwarization

Doctoral Thesis

Multi-Agent Reinforcement Learning for Interactive Decision-Making

Jing Tan

Matriculation Number: 816642

Supervisor
Prof. Dr. Holger Karl

Reviewers
Prof. Dr. Anke Schmeink
Prof. Dr. Jean-Yves Le Boudec

Submitted: 11.03.2023
Accepted: 27.07.2023

Abstract

Distributed decision-making studies the choices made among a group of interactive and self-interested agents. Specifically, this thesis is concerned with the optimal sequence of choices an agent makes as it tries to maximize its achievement on one or multiple objectives in the dynamic environment. The optimization of distributed decision-making is important in many real-life applications, e.g., resource allocation (of products, energy, bandwidth, computing power, etc.) and robotics (heterogeneous agent cooperation on games or tasks), in various fields such as vehicular network, Internet of Things, smart grid, etc.

This thesis proposes three multi-agent reinforcement learning algorithms combined with game-theoretic tools to study strategic interaction between decision makers, using resource allocation in vehicular network as an example. Specifically, the thesis designs an interaction mechanism based on second-price auction, incentivizes the agents to maximize multiple short-term and long-term, individual and system objectives, and simulates a dynamic environment with realistic mobility data to evaluate algorithm performance and study agent behavior.

Theoretical results show that the mechanism has Nash equilibria, is a maximization of social welfare and Pareto optimal allocation of resources in a stationary environment. Empirical results show that in the dynamic environment, our proposed learning algorithms outperform state-of-the-art algorithms in single and multi-objective optimization, and demonstrate very good generalization property in significantly different environments. Specifically, with the long-term multi-objective learning algorithm, we demonstrate that by considering the long-term impact of decisions, as well as by incentivizing the agents with a system fairness reward, the agents achieve better results in both individual and system objectives, even when their objectives are private, randomized, and changing over time. Moreover, the agents show competitive behavior to maximize individual payoff when resource is scarce, and cooperative behavior in achieving a system objective when resource is abundant; they also learn the rules of the game, without prior knowledge, to overcome disadvantages in initial parameters (e.g., a lower budget).

To address practicality concerns, the thesis also provides several computational performance improvement methods, and tests the algorithm in a single-board computer. Results show the feasibility of online training and inference in milliseconds.

There are many potential future topics following this work. 1) The interaction mechanism can be modified into a double-auction, eliminating the auctioneer, resembling a completely distributed, ad hoc network; 2) the objectives are assumed to be independent in this thesis, there may be a more realistic assumption regarding correlation between objectives, such as a hierarchy of objectives; 3) current work limits information-sharing between agents, the setup befits applications with privacy requirements or sparse signaling; by allowing more information-sharing between the agents, the algorithms can be modified for more cooperative scenarios such as robotics.

Zusammenfassung

Die Verteilte Entscheidungsfindung untersucht Entscheidungen innerhalb einer Gruppe von interaktiven und eigennützigen Agenten. Diese Arbeit befasst sich insbesondere mit der optimalen Folge von Entscheidungen eines Agenten, der das Erreichen eines oder mehrerer Ziele in einer dynamischen Umgebung zu maximieren versucht. Die Optimierung einer verteilten Entscheidungsfindung ist in vielen alltäglichen Anwendungen relevant, z.B. zur Allokation von Ressourcen (Produkte, Energie, Bandbreite, Rechenressourcen etc.) und in der Robotik (heterogene Agenten-Kooperation in Spielen oder Aufträgen) in diversen Feldern wie Fahrzeugkommunikation, Internet of Things, Smart Grid, usw.

Diese Arbeit schlägt drei Multi-Agenten Reinforcement Learning Algorithmen kombiniert mit spieltheoretischen Ansätzen vor, um die strategische Interaktion zwischen Entscheidungsträgern zu untersuchen. Dies wird am Beispiel einer Ressourcenallokation in der Fahrzeug-zu-X-Kommunikation (vehicle-to-everything) gezeigt. Speziell wird in der Arbeit ein Interaktionsmechanismus entwickelt, der auf Basis einer Zweitpreisauktion den Agenten zur Maximierung mehrerer kurz- und langfristiger Ziele sowie individueller und Systemziele anregt. Dabei wird eine dynamische Umgebung mit realistischen Mobilitätsdaten simuliert, um die Leistungsfähigkeit des Algorithmus zu evaluieren und das Agentenverhalten zu untersuchen.

Eine theoretische Analyse zeigt, dass bei diesem Mechanismus das Nash-Gleichgewicht sowie eine Maximierung von Wohlfahrt und Pareto-optimaler Ressourcenallokation in einer statischen Umgebung vorliegen. Empirische Untersuchungen ergeben, dass in einer dynamischen Umgebung der vorgeschlagene Lernalgorithmus den aktuellen Stand der Technik bei ein- und mehrdimensionaler Optimierung übertrifft, und dabei sehr gut auch auf stark abweichende Umgebungen generalisiert werden kann.

Speziell mit dem langfristigen mehrdimensionalen Lernalgorithmus wird gezeigt, dass bei Berücksichtigung von langfristigen Auswirkungen von Entscheidungen, als auch durch einen Anreiz zur Systemgerechtigkeit, die Agenten in individuellen als auch Systemzielen bessere Ergebnisse liefern, und das auch, wenn ihre Ziele privat, zufällig und zeitveränderlich sind. Weiter zeigen die Agenten Wettbewerbsverhalten, um ihre eigenen Ziele zu maximieren, wenn die Ressourcen knapp sind, und kooperatives Verhalten, um Systemziele zu erreichen, wenn die Ressourcen ausreichend sind. Darüber hinaus lernen sie die Ziele des Spiels ohne vorheriges Wissen über dieses, um Startschwierigkeiten, wie z.B. ein niedrigeres Budget, zu überwinden.

Für die praktische Umsetzung zeigt diese Arbeit auch mehrere Methoden auf, welche die Rechenleistung verbessern können, und testet den Algorithmus auf einem handelsüblichen Einplatinencomputer. Die Ergebnisse zeigen die Durchführbarkeit von inkrementellem Lernen und Inferenz innerhalb weniger Millisekunden auf. Ausgehend von den Ergebnissen dieser Arbeit könnten sich verschiedene Forschungsfragen anschließen: 1) Der Interaktionsmechanismus kann zu einer Doppelauktion verändert und dabei der Auktionator entfernt werden. Dies würde einem vollständig verteilten Ad-Hoc-Netzwerk entsprechen. 2) Die Ziele werden in dieser Arbeit als unabhängig betrachtet. Es könnte eine Korrelation zwischen mehreren Zielen angenommen werden, so wie eine Zielhierarchie. 3) Die aktuelle Arbeit begrenzt den Informationsaustausch zwischen Agenten. Diese Annahme passt zu Anwendungen mit Anforderungen an den Schutz der Privatsphäre oder bei spärlichen Signalen. Indem der Informationsaustausch erhöht wird, könnte der Algorithmus auf stärker kooperative Anwendungen wie z.B. in der Robotik erweitert werden.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Distributed decision making is concerned with the probabilities of certain choices made by a group of interactive and autonomous agents and the outcome of these choices. In distributed decision making, each agent has one or multiple objectives it tries to maximize through interaction with other agents and the environment. The outcome can be quantified numerically and is the result of joint actions from all of the agents in the group. One-step actions lead to short-term outcomes, and a sequence of actions leads to long-term outcomes. An agent is an entity that is capable of reasoning (i.e., calculating the expected outcome), making a decision and acting upon it — it is usually a mathematical or logical abstraction of a person, but can also be an abstraction of organizations, animals, software functions, etc.

The study of distributed decision making is an interdisciplinary topic in many fields such as statistics, operations research, computer science, networking, economics, biology, etc. There are many application scenarios for distributed decision making in real life. The free market itself is a typical example of distributed decision making: to allocate limited commodities on the market such as products, service slots, energy, etc., each self-interested agent makes individual decisions that are impacted by and also impacting the outcome of other agents' decisions, e.g., our consumer behavior is influenced by the market price, which in turn is influenced by the dynamic demand-and-supply relationship that is the result of sellers and other consumers' decisions to sell or buy. This can be generalized into *resource allocation* applications, in which the agents can be both competitive and cooperative, depending on what behavior contributes to its self interest at the time of making the decision. Another typical example is in robotics, where cooperating agents each take care of a simplified subtask to reduce the complexity of the original task, such as winning a soccer match or detecting / delivering an object.

It is then not surprising that with increasing computation power, the study of distributed decision making goes hand in hand with agent-based-modeling (ABM) methods [112]. One of the first agent-based models is the von Neumann cellular automaton: a group of agents with simple internal states that distinguish them from the common environment and other agents (i.e., *autonomy*); agents can transition between different internal states through communication / interaction with other agents. Later, agent-environment interaction is also included in the modeling; additionally, the agents can observe their neighborhood with limited visibility and store that information as knowledge to aid their actions. In ABM, various types of self interests can be built into heterogeneous agents, and in a simulated environment, agent behaviors, interactions and outcomes can be studied. As the applications in question became more complex, multi-agent systems (MAS) were developed for simulation in different research disciplines such as biology, economics, physics, etc. The advantage of an MAS over classical mathematical modeling (e.g., through a set of differential equations) is the ability to describe uncertainty, disturbances and the resulting state transitions with relative ease that may be otherwise overwhelming in an analytical formulation.

Games are commonly used tools to study strategic interaction between decision makers and its outcome. Game theory as we know it today was first described by von Neumann and Morgenstern [146]: they analyzed a two-person zero-sum game and proved its equilibria in their 1944 book "Theory of Games and Economic Behavior". Today, we easily find applications of game theory in different fields such as social science, economics, computer science, etc. Using game theoretic tools, we can model group decision making and predict the probability of each outcome, and also design mechanisms to incentivize certain group behaviors and increase the probability of some preferred outcomes. The characteristics of various economic models and their implications, the drives and behaviors of participants, and the design of different mechanisms, are all topics of game theory [101].

Perhaps the most famous example of a game is the prisoner's dilemma: two people who have committed a crime together are arrested at the same time and separately questioned. If both of them confess, each gets a prison term of 5 years. If one of them confesses, the one that confesses goes free and the other one gets the maximum prison term of 10 years. If neither of them confesses, both will get 2 years. In this simplified decision-making scenario, it is obvious that the global optimal outcome is for both prisoners to remain quiet and not confess. However, using game theoretic tools, we arrive at the more realistic outcome that both prisoners will confess, assuming they are both rational.

The example of the prisoner's dilemma shows the power of game theory in analyzing and predicting the outcomes of distributed decision making: each player in the game is capable of independently observing, deciding and acting upon its decision (i.e., *self interested*). Distributed decision making does not necessarily mean the elimination of information sharing or cooperation between players — it merely means the motivation for and the mechanism of making the decision lies with each individual player.

Combining game theoretic application and MAS simulation, the question we ask is: assuming a self-interested agent wants to maximize its private utility in a shared environment with other agents, what is the optimal sequence of action that it should take? It has to act strategically, because other agents' decisions will also impact its utility. The question implies that we are no longer satisfied with only analyzing the behaviors of a group of agents, but hope to optimize one or multiple objectives with the help of game theoretic tools and multi-agent simulations. However, obtaining complete information for the optimization is problematic in a dynamic environment, and additionally, as the number of agents, objectives and possible actions increases, it becomes computationally intractable. The challenge lies in determining, for each agent independently, a strategy that best responds to other agents' changing preferences and strategies, without knowledge of this information *a priori*. In other words, the agents need to have learning capabilities to self adapt to the dynamic environment over time.

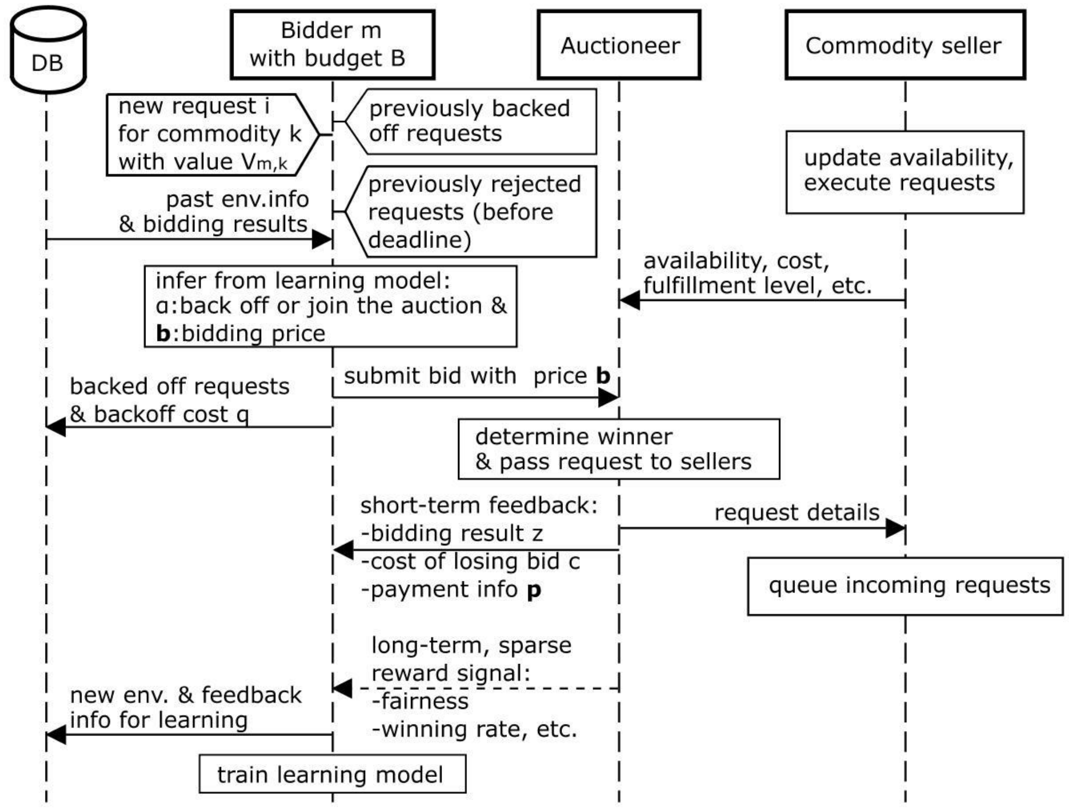
What typically characterizes such a dynamic environment is its vast state and action space, unknown and changing parameters, and lack of upper/lower-bound performance benchmarks for supervised learning. Reinforcement learning (RL) is one of the favorite learning methods in such an environment for its ability to balance between exploration and exploitation, therefore capable of learning with partial, noisy and delayed state information in a big state and action space. Its learning is based on reward signals from the environment, therefore it does not need supervision, i.e., *a priori* knowledge of what the optimal outcome should be. Instead, the consequences of an agent's action are observed and then used to improve future decisions [28]. Thus, the MAS captures the characteristics of a multi-player stochastic game, whereas an RL algorithm assumes a more realistic environment — unknown, but learnable. Multi-agent reinforcement learning (MARL) algorithms are naturally decentralized and distributed.

Game theoretic approaches augmented with MARL are already applied in some scenarios, for example, reference [130] models a power-control problem as a distributed non-cooperative game, in which the players have conflicting objectives and use Q-learning algorithm to find equilibrium. The work focuses on relaying and solves a routing problem. References [77] and [78] model communication resource allocation in a dynamic vehicle-to-vehicle network as a Bayesian coalition game and apply a learning algorithm. Their work focuses on cooperative content sharing.

MARL algorithms in a dynamic environment combined with game theoretic approaches to optimize distributed decision making is also the focus of this thesis. The author uses resource allocation in vehicular network as an example. To evaluate the performance of the algorithm, the author develops a Python discrete-event simulator based on the edge-cloud-computing paradigm and vehicle mobility (details are in Sec. 3.3).

The main contributions of the thesis are:

- The author formulates the distributed decision-making problem (specifically the resource allocation problem) as an auction and designs a utility function to incentivize both competition and cooperation, depending on resource availability. The author also proves that the static outcome is a Nash equilibrium, a maximization of social welfare and a Pareto optimal way of resource allocation (Sec. 4.4).
- The author proposes online distributed MARL algorithms to find the equilibrium of the game in a dynamic and adversarial environment, maximizing multiple objectives with long-term, delayed and sparse reward signals (Sec. 4.5, 5.4 and 6.5). The algorithms show significant individual and system performance improvement compared to benchmarks, as well as good generalization property in different environment setups (Sec. 4.6,5.5 and 6.6).
- The proposed algorithms require no information sharing between agents, no supervision of the learning process and are robust to variance in parameter initialization (Sec. 5.5.2 and 6.6.3). In fact, the agents learn to use the rules of the game to overcome disadvantages in initial parameters without predefined rules (Sec. 4.6.1). This makes them highly flexible in previously unseen environments.
- The author tests the final multi-agent multi-objective long-term algorithm on a single-board computer and demonstrate that inference can be done in milliseconds, fitting to the requirements of time-critical applications (Sec. 6.7).
- The author has open sourced the source code for both the simulator and the algorithms [1, 2, 3, 53].


DB
Bidder m with budget B
Auctioneer
Commodity seller
new request i for commodity k with value Vm,k
previously backed off requests
update availability, execute requests
past env.info & bidding results
previously rejected requests (before deadline)
availability, cost, fulfillment level, etc.
infer from learning model: a:back off or join the auction & b:bidding price
backed off requests & backoff cost q
submit bid with price b
determine winner & pass request to sellers
short-term feedback: -bidding result z -cost of losing bid c -payment info p
request details
queue incoming requests
long-term, sparse reward signal: -fairness -winning rate, etc.
new env. & feedback info for learning
train learning model

Bidders join a repeated auction with one auctioneer and multiple commodity sellers. A bidder m decides to join the auction for commodity type k with bid i or to back off with cost q, depending on past information and current observations. One auctioneer determines the winner, sends back bidding results z and c, payment p, short-term reward (at the end of each auction round), and other long-term rewards such as fairness (at the end of a long interval). Commodity sellers execute requests passed on by the auctioneer. Only the bidders can learn.

Figure 1.1: Auction mechanism

1.2 Methodology

1.2.1 Application scenario

As mentioned in the previous section, the author of this thesis uses resource allocation as an application scenario to study distributed decision making. The system is an abstraction of the classic edge cloud computing architecture. It is set up as an auction with multiple bidders, one auctioneer, and multiple commodity sellers (Fig. 1.1). The bidders do not share information with other bidders or commodity sellers, they only communicate with the auctioneer. This study focuses on the behavior of the independent bidders, conceived of as *agents*. Each bidder has one or multiple objectives to achieve in the auction. As in real life, the bidder's preference of objectives may change over time, in which case it needs to learn the Pareto frontier of a multi-objective optimization problem to respond quickly to changes [121].

Commodities can be products or services. To be produced or executed, each commodity has its own specification and resource needs (in terms of material and time). Over time, a bidder randomly receives requests for one of multiple types of commodities and tries to fulfill the request within a given deadline. All commodities of the same type are equivalent. Availability of commodities is limited for each discrete time slot and decreases after a successful bid. Maximum commodity availability is fixed per type.

Bidders have private *valuations* for each type of commodity (i.e. the benefit it derives from winning the commodity). Each bidder's direct *payoff* from the auction is its valuation minus the price it pays the auctioneer for the commodity and other additional costs. One of the bidder's decision objectives from joining the auctions is to maximize its average payoff over time. If the bidder bids low and loses, it suffers additional costs such as overhead for bidding and rebidding. If it bids high and wins, it has reduced payoff due to high payment. Instead of bidding at a particular time, a bidder also has the option to back off (i.e. delay its bid [33]), hoping for less competition for the commodity, but, on the other hand, using up time towards the fixed deadline and thus making the bid more urgent.

Besides maximizing payoff, the bidder can have other objectives, such as maximizing winning rate, or incentivized to consider system objectives, such as overall fairness among all bidders. The importance of each objective to the bidder is private and expressed through a preference weight vector, which can change over time. The bidder independently learns a bidding strategy to maximize its *utility*: all of its objectives multiplied by the preference weights. Due to frequent changes to the weights, the bidder has to find the Pareto frontier of a multi-objective problem rather than maximizing only one scalarized objective. The thesis studies the performance of different learning algorithms in each bidder.

Auctioneer and commodity sellers: in this thesis, the system has one auctioneer that determines the winners of auctions. The auction is repeated in each discrete time step, as long as there is commodity availability. The auctioneer passes winning bidders' requests to the commodity sellers with the lowest selling price. If no seller has that commodity available, the bids are rejected. For a rejected bid, the bidder can rebid: the maximum permitted rebidding rate has to trade off between higher offloading success rate and additional communication overhead to the system. For simplicity, the author allows rebidding once, the same as in [140]. If the request is accepted by the auctioneer, but not executed by the commodity seller within its deadline, the seller drops the request and informs the auctioneer and the bidder. Both rejected and unfulfilled demands are considered failures. Communication from auctioneer to the bidder after the auction round includes bidding outcome, payment, and other information related to the bidder's objectives, such as long-term bidding success / failure rate, overall availability / resource utilization, fairness, etc. The bidder can decide whether or not to use the feedback for learning. For example, one bidder may have low preference for system objectives and ignore the information; another one may find the information useful.

The commodity sellers can dynamically adjust their price of the commodities. Since the study focuses on the behavior of the bidders, the author of this thesis makes the commodity sellers passive (i.e., not learning-capable) and uses a simplified pricing strategy with load-balancing effect: a seller with higher availability sells at a lower price; the requests flow towards the seller with higher availability until the resource availability at all sellers converge to a common value.

To make the system in this thesis resemble the fast-paced edge cloud computing architecture in real life, the author adds transmission delay between bidders, the auctioneer and commodity sellers, and randomizes resource requirement, queueing and processing time for request execution. Each bidder learns its optimal bidding strategy despite noisy state information in a dynamic environment. Sec. 3.2 describes a more concrete example with vehicular network applications.

1.2.2 Step-by-step problem formulation

This thesis starts with the problem formulation of a second-price auction in its basic form. Then, it adds complexity to the problem formulation step by step, until at the end the problem formulation fully reflects the long-term multi-objective problem that the thesis addresses.

Sec. 2.2.3 briefly explains various types of second-price auctions as background. The problem formulation becomes more complex in each step:

- Basic form of a second-price auction, slightly altered to add budget constraint.
- Combinatorial second-price auction with budget constraint.
- Simultaneous single-item second-price auction with budget constraint.
- Simultaneous single-item second-price auction with budget constraint and partial information.

Sec. 4.3 provides the complete formulation of the short-term single-objective optimization problem that is based on 1) the simultaneous single-item second-price auction with budget constraint and partial information, and 2) the potential game.

This formulation will be further extended in Sec. 5.3 into a long-term single-objective optimization problem, and finally in Sec. 6.4, into a long-term multi-objective optimization problem.

1.2.3 Step-by-step solution approach

The sequence of problems described in Sections 4.3, 5.3 and 6.4 becomes more and more complex, hence, the solution algorithms to solve these problems become more and more sophisticated. In this thesis, the author will present these solution algorithms step by step.

In Sec. 4.5, the solution algorithm to the short-term single-objective optimization problem, named *DRACO*, is a stand-alone MARL module with a fictitious self-play (FSP) wrapper to improve convergence properties.

In Sec. 5.4, the solution algorithm named *MALFOY* adds a curiosity module for long-term sparse reward and a credit assignment module for integration of long-term and short-term objectives, parallel to the original MARL module with FSP.

In Sec. 6.5, the final solution algorithm for long-term multi-objective optimization, named *MOODY*, splits the training into two phases and uses federated learning for gradient updates.

All three algorithms are trained and evaluated in the same environments, which will be described in detail in Chapter 3.

1.2.4 State-of-the-art analysis

State-of-the-art analysis is split into three parts. Sec. 4.1 lists previous research on short-term single-objective optimization for resource allocation, including centralized and distributed approaches, heuristics and learning algorithms, and application-specific approaches. Sec. 5.1 introduces previous research on long-term single-objective optimization methods. Sec. 6.3 gives a thorough background introduction on multi-objective optimization, and explains in detail some single-model and multi-model methods.

1.3 Related papers by the author

The author's published papers and paper in peer review follow the same step-by-step development of the research topic.

[140] designs an interaction mechanism based on second-price auction for the short-term, single-objective resource allocation problem. It proves the theoretical results of equilibria existence, welfare maximization and Pareto optimality in a stationary environment; then it goes on to propose and demonstrate the effectiveness of the MARL algorithm in a dynamic environment. The paper also shows interesting characteristics of a distributed decision-making: the right incentives can improve the system overall performance without sacrifices to individual objectives, and through learning, the agents can overcome disadvantages in initial parameterization and achieve similar results at the end (Chapter 4). Source code is available on github [1].

[139] targets delayed and sparse reward signals and considers long-term effects of agent decisions. It reformulates the problem, and proposes an algorithm with three modules. Then it models two classic auction mechanisms and shows the performance of each module, respectively. The results are included in Section 5.5.1 of the thesis. Source code is available on github [53].

[141] extends the analysis in [139] to the same realistic V2X setup described in Chapter 3. It shows the long-term algorithm's improved performance and generalization properties, as well as its robustness to initial parameters. The results are included in Sec. 5.5.2 of this thesis. Source code is available on github [2].

Finally, one paper (as of the time of writing) in peer review, *Multi-Objective Optimization Using Adaptive Distributed Reinforcement Learning* targets not only long-term delayed and sparse reward signals, but also the learning of multiple objectives, i.e., the Pareto frontier. The paper uses a two-phase training approach based on MAML [52], and demonstrates how the initial model trained through federated learning can quickly adapt to any change in objectives in the test environment with one-shot learning. To address practicality concerns, the paper also proposes several methods to improve computation performance and demonstrates the algorithm's performance on a single-board computer. The results are included in Chapter 6 of this thesis. Source code is available on github [3].

1.4 Thesis outline

The previous section introduces the context and motivation of studying the interdisciplinary area of single and multi-objective optimization through combined MARL and game-theoretic approaches. The rest of this thesis describes the proposed algorithms. The thesis is structured as follows.

Chapter 2 provides preliminaries and related work in the areas of game theory, multi-agent systems and reinforcement learning. Chapter 3 introduces an example resource allocation application in vehicular network (vehicle-to-everything, or V2X) that can benefit from such a game-theoretic MARL algorithm, and the modeling of the system. It also describes the V2X simulation environment for all algorithms. Chapter 4 introduces the short-term single-objective RL algorithm and analyzes the evaluation results. Chapter 5 extends the solution into a long-term single-objective learning algorithm. Finally, Chapter 6 extends it further into a long-term multi-objective learning algorithm. Chapter 7 concludes the thesis and discusses future research directions.

Chapter 2

Background

2.1 Chapter outline

Sec. 2.2.1 briefly introduces the background on game theory, including game theoretic terms used in the following chapters, such as utility, payoff, social welfare, etc.

Sec. 2.2.2 detailizes the scope of the thesis, or types of the games studied. Specifically, the second-price auction mechanism that is the basis of the mechanism design is introduced in Sec. 2.2.3. Then, the concept of potential games is introduced in Sec. 2.2.4. Both types of games are used in the subsequent chapters to design the interaction mechanism.

Sec. 2.3.1 introduces Markov decision process (MDP), Bellman equations, and classic single-agent learning algorithms such as dynamic programming, temporal difference learning, policy gradient, etc. Specifically, actor-critic reinforcement learning algorithm is explained in detail. The algorithm is the basis of the algorithms designed in this thesis.

Sec. 2.3.2 introduces the challenges multi-agent reinforcement learning (MARL) faces (e.g., non-stationarity), and some online-learning algorithms developed to meet the challenges, such as no-regret and best-response algorithms. This is the basis for design of the MARL algorithms in the subsequent chapters.

Sec. 2.4 concludes the chapter.

2.2 Games

2.2.1 Preliminaries

Game theory studies the interaction of multiple *economic agents* who act according to their *preferences* to maximize a *utility*. It also studies the *outcome* of that interaction.

An *economic agent*, or *agent* for short, is an entity that has one or multiple *objectives* and is capable of making and acting on independent decisions to maximize its *utility*. A *utility* is the scalarized objective achievement score based on the agent's preference of one or multiple objectives. The achievement of each objective can be measured by a real number, a higher value represents a higher degree of achievement of the objective. An example of an objective is the economic payoff: in this study, the author uses the term *payoff* to refer to a player's economic gain by participating in a game. A *preference* is a set of real numbers, a higher value represents a higher priority for the corresponding objective. The maximization of the utility is through a *utility function* that maps the agent's preference and objective achievements to a real number, such that utilities with different objective achievements and preferences can be ranked. This real number is also referred to as the *reward* – a term used in reinforcement learning (Sec. 2.3.1).

A *game* is a situation in which an agent, as a player, aims to maximize its utility by predicting and reacting to other agents' behaviors. Games can be roughly divided into two categories: sequential games and simultaneous games. In a sequential game, the agent acts when it has information of other agents' previous actions. In a simultaneous game, all agents act without information of other agents' actions. A sequential game or extensive-form game can be represented by a directed graph. Each node in the graph represents a point of action selection; a simultaneous game or normal-form game can be represented by a matrix. We can also categorize games by agents' access to information: in a game with perfect information, the agent knows all previous actions of all agents, the rule of the game, as well as all agents' utilities. Otherwise, it is a game with imperfect information. A third way to categorize games is through the type of agent interaction: if agents integrate a global objective into their private objectives, the game is at least partially cooperative; otherwise, the game is non-cooperative or competitive. An *outcome* of the game is the set of utilities of all agents, and *social welfare* is the sum of utilities achieved by all agents.

Based on the definitions above, we can see that an economic agent has the following characteristics:

1) it has objectives, preferences and a utility function.

2) it can calculate the probability of a sequence of actions (a *strategy*) leading to a certain outcome.

3) it can interact with other agents and/or the environment through action.

4) in the definition of an economic agent, the author adds another characteristic that is typically assumed in game theoretic literature: the economic *rationality* (Sec. 2.2.3). An economically rational agent always follows the sequence of actions that it believes to be the most probable to yield the most preferred outcome.

To analyze such a game, one of the most commonly used tools is equilibrium. For example, the agents, each with its strategy, reach a Nash Equilibrium (NE) when no one can improve its utility without changing other agents' strategies. This set of strategies is called the NE strategies, and none of them should be a strictly dominated strategy (i.e. a strategy that always delivers a worse outcome than an alternative strategy).

2.2.2 Scope

Among distributed decision-making mechanisms such as auction, posted price and negotiation, an auction is most suitable in a dynamic and competitive environment where the number of agents and their preferences vary over time and private valuations of commodities are dispersed [47, 125]. Among various forms of auction, a second-price sealed-bid auction maximizes welfare rather than revenue and has limited information sharing; these assumptions befit the requirement of many real-life applications. Specifically, the approach in this thesis is based on Vickrey-Clarke-Groves (VCG) for second-price combinatorial auction [145]. In the case of this thesis, the author uses simultaneous combinatorial auction as a simplified version of VCG — each bidder bids for all commodities separately and simultaneously, without having to specify its preference for any bundle [49]. It assumes no correlation between commodities.

In the rest of this section, the author incrementally extends the problem formulation from a classic second-price auction to a more complex problem that resembles the short-term single-objective problem addressed in Sec. 4.3. Sec. 4.3 provides the complete formulation of the short-term single-objective problem. This formulation will be further extended in Sec. 5.3 into a long-term single-objective problem, and finally, in Sec. 6.4, into a long-term multi-objective problem. In this section:

- Basic form of a second-price auction is introduced, then slightly altered to add a budget constraint (Sec. 2.2.3).

- Combinatorial second-price auction with budget constraint is described. The VCG mechanism is introduced, as well as its intractability (Sec. 2.2.3).
- To ensure feasibility, simultaneous single-item second-price auction with budget constraint is presented (Sec. 2.2.3).
- To further reduce the complexity, the same problem is described with partial information requirements. The need for a learning algorithm motivates researches in repeated auctions (Sec. 2.2.3).
- To incentivize cooperation, potential game is introduced (Sec. 2.2.4).

2.2.3 Second-price auction

Basic form

The author first describes the problem as an optimal allocation problem in a normal form game, based on the definitions from Sec. 1.2.1:

Definition 2.1. *A bidder $m \in M$ receives a continuously distributed valuation of the commodity $v_m \in [l, h]$ and chooses its strategy to submit bidding price $b_m = f_m(v_m)$ from its strategy set F_m. Any strategy function $f_m(\cdot)$ is increasing: for $i < j, \forall i, j \in [l, h]$, $f_m(i) < f_m(j)$. If a bidder wins the commodity, it pays a price p_m, and its utility is $u_m = v_m - p_m$. The goal is to find an allocation of the commodity that maximizes total welfare $\sum\limits_{m \in M} u_m$.*

Second price sealed-bid auction is first described by Vickrey in 1961 [145], and it can be proven that it is an optimal allocation which maximizes the total welfare at Nash equilibrium, and that it is a weak dominant strategy for rational bidders to bid their true valuation of the commodity [25].

The winning bidder $m = 1$ in a second-price auction pays the 2nd highest biding price: $p_1 = f_2(v_2)$, and its utility is $u_1 = v_1 - f_2(v_2)$.

In some cases, a budget constraint B_m is added to the basic form such that the bidder cannot overbid, i.e., $f_m(h) \leq B_m$. With budget constraint, it can also be proven that there exists a Nash equilibrium, and that it is Pareto optimal [135]. This applies to simple, rational multi-player, n^{th}-price sealed-bid auctions. Sec. 4.4 describes the theoretical results based on the method from [135], with adaptations to suit the use case in this thesis. The case in this thesis is different from [135] in the definition of utility function.

The author of this thesis includes the second price as part of the utility function, rather than the constant used in [135]. The author also adds a penalty cost for losing the bid.

In this thesis, the bidders need to bid for multiple commodities as a consumption bundle. This falls into the category of combinatorial auction. In theory, v_m is known to the bidders before they join the auction. In this study, v_m is given to the bidder, drawn from a uniform random distribution such that the valuation differs within 25% of the maximum valuation among all bidders. This is not always a realistic assumption, as bidders in real life may not know the precise valuation of each commodity. In future work, the bidder may have the ability to learn the valuation function during repeated auction.

Combinatorial second-price auction

The combinatorial problem is described as follows.

Definition 2.2. *Bidder $m \in M$ bids for a consumption bundle $S \in 2^K$ among commodities set K. Its valuation on the bundle $v_m(S)$ is drawn from a distribution. It bids price b_m for the bundle, and if it wins, it pays a payment p_m, its utility is $u_m(S) = v_m(S) - p_m$. $v_m(S)$ is private information to each bidder; it is non-negative and depends only on the commodities (i.e. no externalities influencing the valuation of the same set of commodities). The goal is to maximize $\sum\limits_{m \in M} u_m(S)$.*

In a combinatorial auction, each bundle can include multiple commodities, and one type of commodity can be included in multiple bundles. The valuation is given per bundle, it depends on the specific combination of commodities in the bundle and is not always linear to the valuation of each commodity. For example, if a buyer bids for a bundle of tulips of different colors, the buyer's valuation of a bouquet may be exponential to the size of the bouquet, or the buyer would value a certain color combination higher than any single-colored bouquet.

For combinatorial auctions, the VCG mechanism is known to be an incentive-compatible and optimal allocation mechanism which maximizes the social welfare [105]. Formally, a VCG mechanism is defined as follows.

An auction with the set of bidders M has possible outcome $w \in \{w_1, w_2, \cdots, w_n\}$. Each bidder $m \in M$ has private valuation information $v_m(w)$ on each of the outcomes. Its utility is $u_m(v_m, w) = v_m(w) - p_m(w)$, where $p_m(w)$ is its payment. The goal of the mechanism is to find an outcome w^* such that $w^* = \arg\max\limits_{w} u_m(v_m, w), \forall m \in M$.

The mechanism also calculates the best outcome without winner m in the auction, denoted w^*_{-m}; $-m$ denotes all other bidders in the auction except bidder m. Bidder m's payment is defined as the difference in the welfare caused by m's participation:

$$p_m(w) = \sum_{j \neq m} u_j(v_j, w^*_{-m}) - \sum_{j \neq m} u_j(v_j, w^*) \tag{2.1}$$

However, the mechanism requires the bidders to know their utility for each possible outcome, and in order to achieve optimal allocation, the mechanism requires bidders to truthfully submit their utilities [105]. Moreover, such mechanisms are computationally intractable, as the number of actions available to a bidder grows [49] (i.e. in the case of continuous action space, which is also the case in this thesis, with continuous bidding prices).

One way to simplify the problem of bidders' unknown utilities is by simultaneous (or item-bidding) combinatorial auction. Each bidder bids for all K commodities separately, without having to specify its preference for any other consumption bundles [49].

Theoretical bounds of the welfare achievable in a simultaneous combinatorial second-price auction with no overbidding and subadditive utilities are proven in [23] to be lower bounded at $\frac{1}{2 \ln k}$ of the optimal allocation (i.e. the welfare-maximizing allocation through a VCG mechanism), where k is the number of commodities, and upper bounded at 1/2 of the optimal with incomplete information. Authors of [49] improved the lower bound to 1/4 for more commodities.

Simultaneous single-item combinatorial second-price auction

Definition 2.3. *Bidder $m \in M$ bids price vector $b = b_1, \cdots, b_{|S|}$ simultaneously for each commodity in a subset $S \in 2^K$ among commodities set K. It has a valuation on winning any combination of the commodities $v_m(S)$. If it wins, it pays a payment p_m, its utility is $u_m(S) = v_m(S) - p_m$. The goal is to maximize $\sum_{m \in M} u_m(S)$.*

Simultaneous single-item auction mechanisms still have the problem of computational intractability. To further reduce the complexity, the author of this thesis uses online learning algorithms for efficient allocation, without knowledge of full information of the game. Learning algorithms depend on only partial information; they learn the distribution of players' utilities and payment through feedback in each round of a repeated game.

Incomplete-information games

Games with incomplete information are also known as *Bayesian games*. In a Bayesian game, the agents are assumed to have only partial information of the utilities. Their beliefs of their own or other agents' utilities follow a probability distribution, which can be updated as the agent obtains more information. There are three types of rationality with different assumptions of the agent's knowledge [102]. An *ex-ante* individual rationality (IR) is based on the assumption that an agent does not have knowledge of either its own or other agents' precise valuation (of commodities); instead, it only has knowledge of both distributions, and it is individually rational if it joins the auction only when it expects a positive utility averaged over its belief of both distributions. Similarly, *interim* IR is based on the assumption that the agent has knowledge of their own valuation, but not of others, they would join the auction if its expected utility averaged over its belief of other agents' valuation distribution is positive. The strongest assumption is in *ex-post* IR, agents have complete information of the game. To motivate agents to join the auction voluntarily, they have to be either *ex-ante IR*, or *interim IR*, or *ex-post IR*. The author of this thesis designs a mechanism where the agent has no prior knowledge of other agents, but knows precisely its own valuation. It is therefore interim IR.

A simultaneous single-item combinatorial second-price repeated auction with partial information can be formulated as follows.

Definition 2.4. *Each bidder $m \in M$ has a private value $v_{(m,k)}$ for each commodity $k \in K$. At each time t, bidder m draws its action (bidding price for each of the commodities) $\mathbf{b}_m^t \in \mathbb{R}^{|K|}$ from a mixed strategy π_m^t. Its utility for any outcome of the auction is $u_m(\mathbf{b}_m^t)$. The auction repeats for T periods. The goal is to maximize $\frac{1}{T}\sum_{t=1}^{T}\sum_{m\in M} u_m^t$.*

Such games have Bayesian equilibria. Authors of [32] prove that even finding an approximate equilibrium is NP-hard. However, they also show that we can relax a Bayesian equilibrium into a coarse correlated equilibrium, which does not have to be best responses, but can be computed in polynomial time [32]. This is also called no-regret dynamics.

2.2.4 Potential games

Monderer and Shapley, in their 1996 paper [99], introduced the concept of *potential games*. Potential games are games that associate social welfare with individual agent utilities through a potential function. It is often used as basis for cooperative games, such as in [92].

Definition 2.5. *Let M be the set of players, A be the set of actions, u be the set of utilities. The game $G(M, A, u)$ is an exact* potential game *if and only if there exists a potential function $\phi(A) : A \to \mathbb{R}$ such that for all $m \in M$, $u_m(b_m, b_{-m}) - u_m(b'_m, b_{-m}) = \phi_m(b_m, b_{-m}) - \phi_m(b'_m, b_{-m}), b \in A$ [99].*

All bidders' utilities are associated through the potential function, and change in a single bidder's utilities is reflected exactly in (or, in the case of a weighted potential game, as a fraction of) the system utility.

Authors of [45] prove that in a potential game with set of player M and discrete set of strategies A per player, the worst case complexity of computing a best response is on the order of MA^{M-1}, and the average complexity of random potential games is $e^{\gamma}M + O(M)$, where $\gamma = \lim_{n\to\infty}(-\ln n + \sum_{k=1}^{n} \frac{1}{k})$ is the Euler constant.

Sec. 4.4.2 designs an auction mechanism that is reduced to a potential game under certain conditions. Using these results, the author demonstrates how the mechanism incentivizes individual bidders to consider system goals in their utilities.

2.3 Multi-agent reinforcement learning

2.3.1 Single-agent RL

RL problem formulation has roots in dynamical systems theory. It is the control of a partially known *Markov decision process (MDP).*

An MDP is one type of probabilistic sequential decision models. Such a model can be represented by a directed graph, with a set of decision-making points (nodes), each with its state and a set of actions. Each state-action pair has an immediate reward or cost, as well as a transition probability to transition to another decision-making point. A decision is made to choose an action from the available set of actions, then based on the transition probability, the decision maker arrives at the next decision-making point and receives a reward or a cost, accordingly. Over time, the decision maker accumulates rewards / costs from its sequence of decisions. The special characteristic of a MDP that distinguishes it from other sequential decision models is the fact that everything related to a decision-making point, i.e., its available action set, the rewards and costs and transition probabilities, only depends on the current state-action pair and independent from any past states or actions [118].

An RL algorithm solves problems that can be typically described as an MDP. Its goal is to find an optimal policy that traverses a sequence of state-action pairs and returns the maximum cumulative reward of an MDP. In this study, the *policy* of the RL algorithm is equivalent to the *strategy* (Sec. 2.2.1) of a game – it is the RL terminology for a sequence of actions.

RL differs from supervised learning. Supervised learning trains a model that extrapolates and generalizes given situations and specifications, so that it responds correctly to new inputs that are not present in history. RL problems are interactive: when it is hard to obtain examples of all situations and with correct labels, RL agents observe reward signals as consequence of their actions and learn in previously unknown environments. RL makes tradeoff between exploitation and exploration: finding better rewards by trying different states and actions, but also by improving on its past decisions. RL is also different from unsupervised learning: unsupervised learning's goal is to find hidden structures in data, whereas RL's goal is to maximize a reward signal.

RL is different from evolutionary methods commonly used to solve optimization problems (e.g.genetic algorithm, simulated annealing, etc.): optimization methods never estimate value functions; instead, they apply static policies to separate instances of possible environments and chooses the policy with the best score. Evolutionary methods are specially advantageous when policy space is small or structured. They do not view policies as a function linking states and actions. In other words, evolutionary methods *search*, RL agents *learn*. The latter can be more efficient if the result depends on individual interaction with the environment as they focus on fitting actual environment model to their belief of the model (learn despite uncertainty).

There are four elements of RL: policy π, reward r, state value v or state-action value q, and, optionally, a model of the environment, expressed as the probability $p(s', r|s, a)$ of getting reward r by transitioning from state s to s' with action a. We define the expected future return G_t at time t of a MDP to be the total of future rewards over time:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1} \tag{2.2}$$

where γ is the discount factor for future rewards, R_t is the reward value at time t.

The *value* of any MDP state s following policy π can be defined recursively by its possible future states:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}] \\ &= \sum_a \pi(A_t = a|S_t = s) \cdot \sum_{s',r} p(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2.3}$$

where $S_t = s$ is the current state, $S_{t+1} = s'$ is the possible next state, $p(\cdot)$ is transition probability from s to s' through current action $A_t = a$, R_{t+1} is the next reward, r is the possible reward associated with the tuple (s, a, s').

Based on Eq. 2.3, we define the optimal state value as the maximum expected value achievable with the optimal policy:

$$\begin{aligned} v^*(s) &= \max_\pi v_\pi(s) \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v^*(s')] \end{aligned} \tag{2.4}$$

Eq. 2.4 is the Bellman optimality equation for state value. Similarly, we can also define state-action pair value and optimal value as:

$$\begin{cases} q_\pi(s,a) &= \mathbb{E}_\pi[G_t|(s,a)] = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \\ q^*(s,a) &= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q^*(s',a')] \end{cases} \tag{2.5}$$

The two Bellman optimality equations are equivalent when π is the optimal policy: getting the best state value under the optimal policy is the same as choosing the best action from that state: $v^*(s) = \max_a q_{\pi^*}(s,a)$. For finite MDPs, the Bellman optimality equations have a unique solution independent of the policy. This is a system of N equations, N being the number of states; and the system can be solved for the unknown variables $v^*(s)$ if transition probabilities $p(s', r|s, a)$ are known (i.e., complete knowledge of the environment). Otherwise, when the environment is unknown (i.e., incomplete $p(s', r|s, a)$ knowledge), we need an RL algorithm that balances between 1) improving the existing best performance from past experience (i.e., "exploitation") and 2) finding potentially better policies in under-explored space (i.e., "exploration") through trial and error. During this process, the state / state-action values are incrementally updated. If all states and actions are visited infinite times, the average (expected) value converges to the real value.

Dynamic programming

Dynamic programming (DP) assumes a perfect model of the environment. DP is also called a *model-based* algorithm. In RL, "model" refers to the model of the environment and the reward. More specifically, if $p(s', r|s, a)$ is known or predicted, it is model based; otherwise it's *model free*.

DP computes optimal policies by using value functions to organize and structure the search for good policies. They are exact solutions with finite states, actions and reward sets. DP algorithms are obtained by turning Bellman equations into update rules for improving approximations of the desired value functions. All updates done in DP algorithms are called *expected* updates because they are based on an expectation over all possible next states rather than on a sample next state. This is done through policy iteration. In every iteration, policy evaluation and policy improvement steps are alternately applied.

Policy evaluation works as follows. Given a policy, the expected value of the state can be calculated through a value function:

$$v_{k+1}(s) = \mathbb{E}[R_{t+1} + \gamma v_k(s')|s_t] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2.6}$$

where k is the index of update iteration. Iterative policy evaluation converges in a finite MDP to v_π as $k \to \infty$, as proven by [119].

The *policy improvement theorem* [138] is based on the theorem that if we follow a random policy π except in one state s, where we choose the best action a^* instead of

a_π, we would get a state value $v(s)$ that is at least as high as the random policy, and the new policy will be better than the original one. Extending this theorem, it can be proven [138] that if we greedily choose the best action in every state, the final policy after the iterations is an optimal policy.

$$\begin{cases} \pi(s) & = \arg\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] \\ v_\pi(s) & = \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] \end{cases} \tag{2.7}$$

Temporal difference learning

Temporal difference (TD) learning is a model-free RL method with guaranteed convergence if step-size parameter is sufficiently small. TD learning uses bootstrapping to estimate the value functions. Specifically in TD(0) method, where the estimation is only based on one-step return from the future state, the target of the current value function called the *TD target* is written as $R_{t+1} + \gamma v(s')$, where γ is the discount rate of future return. When a new sample of state value $v(s)$ becomes available, TD learning method updates the state value via TD error:

$$\delta_t = R_{t+1} + \gamma v(S_{t+1} = s') - v(S_t = s), \tag{2.8}$$

and the updated state value becomes:

$$v(s) = v(s) + \alpha\delta_t \tag{2.9}$$

As shown in Eq. 2.9, the TD method requires an explicit computation of each state value. It belongs to a group of methods called the tabular methods that update single state values during the traversal of states and actions. The state being updated goes a small step α (also called the *learning rate*) towards the TD target. As state and action spaces grow, especially with continuous states and actions, tabular methods become inefficient.

Discounted and average rewards

Discounted rewards are used in episodic MDPs. With continuing tasks and infinite time steps, average reward is used instead. Using the difference between current reward and average reward, we redefine the Bellman equations:

$$\begin{cases} v_\pi(s) & = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r - \bar{r}(\pi) + v_\pi(s')] \\ q_\pi(s,a) & = \sum_{s',r} p(s',r|s,a)[r - \bar{r}(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s',a')] \end{cases} \tag{2.10}$$

where $\bar{r}(\pi)$ is the average reward. Similarly, the differential form of the TD error is:

$$\begin{cases} \delta_t & = R_{t+1} - \bar{R}_{t+1} + v(S_{t+1} = s') - v(S_t = s) \\ \delta_t & = R_{t+1} - \bar{R}_{t+1} + q(S_{t+1} = s', A_t = a) - q(S_t = s, A_t = a) \end{cases} \tag{2.11}$$

This thesis uses average reward for continuing tasks.

Value function approximation

When the state and action spaces are big or the environment is unknown, updating single state values is inefficient. Instead, we use function approximation with a limited number of parameters. During traversal of states and actions, the function parameters are updated, thus updating many state values at once. Depending on the function, the approximation method can be arbitrarily complicated — we now have a much wider range of approximation tools for value prediction.

Approximate value function is no longer represented by a table (as in tabular methods), but as a parameterized function with weight vector $\mathbf{w} \in \mathbb{R}^d$ where the dimension of the weight can be much smaller than the number of states. In this method, the value function is written as $\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$ as the approximate value of state s given weight vector $\mathbf{w}$.

Policy approximation

Policy approximation methods learn a parameterized policy that can select actions without a value function. The performance of the approximated policy is the value of the first state for episodic MDPs or the average reward for MDPs of *continuing tasks* without a terminal state. A *terminal state* is the end state of an episode in an MDP. The advantages of parameterization of policies over parameterization of state or state-action values are: 1) the approximation of policy can eventually reach a deterministic policy, whereas with ϵ-greedy action selection over action values, the ϵ prevents reaching the absolute deterministic policy; 2) it enables selection of actions with arbitrary probabilities (i.e., stochastic policies), and action-value methods are not designed for finding stochastic optimal policies; 3) policy may be a simpler function to approximate than value functions; 4) choice of policy parameterization may be a good way to incorporate prior expert knowledge on the form of the policy; 5) it has a stronger convergence guarantee, because the action probabilities change smoothly as a function of the learned parameter of a continuous policy parameterization. The gradient of the parameterized policy is written as [138]:

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a \nabla_\theta q_\pi(s,a)\pi(a|s,\theta) \tag{2.12}$$

where the reward function $J(\theta)$ represents the expected return following policy $\pi(\theta)$. It can be proven that the gradient of $J(\theta)$ with regard to its parameters θ is only related to the gradient of the parametric policy approximation function to θ, and there is no need for calculating the gradient of the state distribution $\mu(s) = \sum_{k=0}^{\infty} Pr(s_0 \rightarrow s, k, \pi)$, or the sum of probability of transitioning from state s_0 to state s in k steps under policy π.

There are many policy approximation RL algorithms. The rest of this section briefly introduces the actor-critic method that is most relevant to the proposed RL algorithm. The actor-critic method is an extension of the REINFORCE and REINFORCE-with-baseline methods, which are also briefly introduced below.

The REINFORCE method

The REINFORCE method is a stochastic gradient ascent method that requires the sampling of gradient proportional to Eq. 2.12. Note that $\mu(s) = \sum_{k=0}^{\infty} Pr(s_0 \rightarrow s, k, \pi)$ is the expected number of times s is traversed following π. Therefore, we only need to sample states following policy π. With sampled state S_t, the gradient can be rewritten as:

$$\nabla_\theta J(\theta) \propto \sum_s \mu(s) \sum_a \nabla_\theta q_\pi(s,a)\pi(a|s,\theta) = \mathbb{E}_\pi \big[\sum_a q_\pi(S_t,a)\nabla_\theta \pi(a|S_t,\theta) \big]$$

$$= \mathbb{E}_\pi \big[\sum_a q_\pi(S_t, a)\pi(a|S_t, \theta) \frac{\nabla_\theta \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \big] \tag{2.13}$$

Action A_t is sampled according to the policy distribution, we replace the sum over all the random actions a and probabilities $\pi(a)$ with sampled action A_t:

$$\nabla J(\theta) = \mathbb{E}_\pi \big[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \big] \mathbb{E}_\pi \big[G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \big] \tag{2.14}$$

since expected return given state and action is $\mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)$. Therefore, according to gradient method, the parameter update is:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) = \theta + \alpha G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} = \theta + \alpha \gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta) \tag{2.15}$$

The fraction is reduced to the derivative of the natural log function because $\frac{d \ln f(x)}{dx} = \frac{f'(x)}{f(x)}$.

Algorithm 1 REINFORCE

```
1: Initialize a differentiable parameterized policy (e.g. Gaussian) $\pi(a|s, \theta)$
2: Use Monte Carlo method to generate whole episodes of state-chains until terminal
   state.
3: while true do
4:     Calculate expected return $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$.
5:     Update policy parameter $\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(a|s, \theta)$.
6: end while
```

REINFORCE with baseline

A baseline $b(s)$ that can vary in value with regard to the state, but does not change with regard to the actions, helps to reduce the high variance of the above REINFORCE method. Also because it is invariant with regard to the actions, it can be simply added to $\nabla J(\theta)$ as:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla\big(q_\pi(s,a) - b(s)\big)\pi(a|s,\theta) \tag{2.16}$$

A natural choice of the baseline would be the estimated state value: $b(s) = \hat{v}(s, \mathbf{w})$. With the added term, we get parameter update:

$$\theta \leftarrow \theta + \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla \ln \pi(A_t|S_t, \theta) \tag{2.17}$$

To update the parameters of the value function $\mathbf{w}$, we calculate the gradient as:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{1}{2}\alpha\nabla\big[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\big]^2 \tag{2.18}$$

where the true value $v_\pi(S_t)$ can be approximated by the bootstrapped value from $\hat{v}(S_t, \mathbf{w}_t)$ with: $v_\pi(S_t) \approx R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_t)$.

Algorithm 2 REINFORCE with baseline

```
1: Initialize, input differentiable $\pi(a|s,\theta)$ as well as differentiable $v(s,\mathbf{w})$
2: Use Monte Carlo method to repeatedly generate episodes of state chains till terminal
   state.
3: **while** true **do**
4:     Calculate expected return $G \leftarrow \sum_{k=t+1}^{T} R_k$
5:     Calculate $\delta = G - \hat{v}(s,\mathbf{w})$
6:     Update value parameter $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}}\delta\gamma\nabla\hat{v}(s,\mathbf{w})$
7:     Update policy parameter $\theta \leftarrow \theta + \alpha_\theta\delta\gamma\nabla \ln \pi(a|s,\theta)$
8: **end while**
```

The actor-critic method

The actor-critic (AC) method is an improvement of REINFORCE-with-baseline, able to bootstrap and learn incrementally. Based on Eq. 2.17, we substitute G with the Bellman update:

$$\theta \leftarrow \theta + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla \ln \pi(A_t|S_t, \theta) \tag{2.19}$$

Algorithm 3 Actor-critic

```
1: Initialize, input differentiable $\pi(a|s,\theta)$ as well as differentiable $v(s,\mathbf{w})$
2: Randomly initialize $\theta$ and $\mathbf{w}$. Randomly initialize first state $S_0$ in each episode.
3: Loop until terminal state
4: **while** true **do**
5:     Choose $a$ from $\pi(\cdot|s,\theta)$, observe next state $s'$ and reward $r$
6:     Calculate $\delta = r + \gamma\hat{v}(s',\mathbf{w}) - \hat{v}(s,\mathbf{w})$
7:     Update value parameter $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\gamma^t\delta\nabla\hat{v}(s,\mathbf{w})$
8:     Update policy parameter $\theta \leftarrow \theta + \alpha^{\theta}\gamma^t\delta\nabla\ln\pi(a|s,\theta)$
9:     $s \leftarrow s'$, increase t by 1
10: **end while**
```

Note that in Alg. 3, $\delta = R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$ is the TD error for episodic tasks. In fact, δ can also be other values, such as *advantage*: $A(t) = q(S_t, A_t) - v(S_t)$, or the TD error for continuing tasks: $\delta = R_{t+1} - \bar{R}_{t+1} + \hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$ (Eq. 2.11).

The steps calculating δ and updating value function parameters $\mathbf{w}$ are called *critic*, which serves to calibrate the *actor* — the policy gradient algorithm that updates θ. If we use a neural network for non-linear value function estimation, $\mathbf{w}, \theta$ are parameters of two neural networks. In the neural network, the derivative with respect to the parameters is calculated from the loss function. The loss function for the critic is $\frac{1}{2}\big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\big]^2$, for the actor is $\delta\ln\pi(A_t|S_t,\theta_t)$.

To estimate continuous actions or a stochastic policy, i.e., which action to take according to a probability density function, the actor model learns the parameters of the distribution, and the action is sampled accordingly. Typical distribution to represent continuous actions is a Gaussian distribution:

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}}\exp\Big(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}\Big) \tag{2.20}$$

where the mean $\mu(s,\theta)$ and the standard deviation $\sigma(s,\theta)$ are functions of input s and learned parameters θ:

$$\begin{cases} \mu(s,\theta_\mu) & = \theta_\mu^T x_\mu(s) \\ \sigma(s,\theta_\sigma) & = \exp(\theta_\sigma^T x_\sigma(s)) \end{cases} \tag{2.21}$$

where x is an extracted feature vector from states, and gradient for the mean and the standard deviation parameters can be derived as follows:

$$\begin{cases} \nabla \ln \pi(a|s, \theta_\mu) & = \dfrac{1}{\sigma(s, \theta_\sigma)^2}(a - \mu(s, \theta_\mu))x_\mu(s) \\ \nabla \ln \pi(a|s, \theta_\sigma) & = \Big(\dfrac{(a - \mu(s, \theta_\mu))^2}{\sigma(s, \theta_\sigma)^2} - 1\Big)x_\sigma(s) \end{cases} \tag{2.22}$$

In this thesis, continuous action space and a Gaussian distribution are used.

2.3.2 Multi-agent RL

MARL is commonly used for learning stochastic games. Such games have multiple players and multiple states; therefore, they differ from a classic MDP with a single player and multiple states and also from a matrix game with multiple players and a single state. Agents in a stochastic game may not know transition or reward functions, but are required to select actions and observe received reward and gain information [127].

Formally, a stochastic game can be represented by $(n, S, A_1, \cdots, A_n, T, \gamma, R_1, \cdots, R_n)$ where n is the number of players, $T : S \times A_1 \times \cdots \times A_n \times S \to [0, 1]$ is the transition function, $A_i (i = 1, \cdots, n)$ is the action set for player i, $\gamma \in [0, 1]$ is the discount factor, and $R_i : S \times A_1 \times \cdots \times A_n \times S \to \mathcal{R}$ is the reward function for player i, Π_i is the set of strategies available to player i, and state value $V_i^*(s) = V_i(s, \pi_1^*, \cdots, \pi_n^*)$ is the expected sum of discounted rewards for player i given current state s and all players' strategies:

$$\begin{cases} V_i^*(s) & = \sum_{a_1, \cdots, a_n \in A_1 \times \cdots \times A_n} Q_i^*(s, a_1, \cdots, a_n)\pi_i^*(s, a_1) \cdots \pi_n^*(s, a_n) \\ Q_i^*(s, a_1, \cdots, a_n) & = \sum_{s' \in S} T(s, a_1, \cdots, a_n, s')[R_i(s, a_1, \cdots, a_n, s') + \gamma V_i^*(s')] \end{cases} \tag{2.23}$$

where $\pi_i^*(s, a_i) \in \mathrm{PD}(A_i)$ is a probability distribution over action a_i under player i's NE strategy. $T(s, a_1, \cdots, a_n, s') = Pr\{s_{k+1} = s' | s_k = s, a_1, \cdots, a_n\}$ is the probability of the next state being s' given s and a.

The goal is to find a strategy $\pi_i : S \to A_i$ that maximizes player i's discounted future reward.

Learning in non-stationary environment

Three challenges in MARL are 1) non-stationary environment, 2) exploitation from adaptive adversarial behavior [27], because MDP assumes a stationary environment [61] and 3) continuous / infinite-dimensional action space.

To meet these challenges, research focuses on two learning dynamics: no-regret and best-response [67]. No-regret and best-response algorithms typically fall into the category of online-learning algorithms [71]: environment data p_t drawn from an unknown distribution becomes available in a sequential order. At each time step t, the online learning algorithm with model parameters ω_t receives signal v_t. Based on a historical record of $p_k, v_k, b_k, u_k, k \in \{1, \cdots, t-1\}$, the model decides on the optimal action b_t and predicts the expected utility $\hat{u}_t(b_t, \omega_t)$. After new data becomes available, the model calculates the actual utility $u_t(b_t, p_t)$ and the loss function $l(\hat{u}_t, u_t, \omega_t)$. Then, some learning algorithm is applied to update the model parameters $\omega_{t+1} = f(l(\hat{u}_t, u_t, \omega_t), \omega_t)$ such that the loss function is minimized over time.

No-regret algorithms

No-regret algorithms try to address the non-stationarity issue by finding correlated equilibria. No-regret algorithms usually guarantee convergence of expected utility with finite action space; however, they guarantee best response only in certain conditions [18].

ϵ-coarse correlated equilibrium is a distribution π over action profiles such that for every player i and every action b'_i, we have $\mathbb{E}_{b\sim\pi}[u_i(b'_i, b_{-i})] - \mathbb{E}_{b\sim\pi}[u_i(a)] \leq \epsilon$ [110].

This means, for an action profile $b \in \mathbb{R}^{MK}$ drawn from the ϵ-coarse correlated equilibrium, if the actions in b are proposed to each bidder, there is no gain for any bidder to deviate from the proposed action b_i and play b'_i, because their utility will otherwise be smaller (or at least within a very small ϵ).

Through the no-regret dynamic, bidders can find correlated equilibria [58]: A bidder has a history of auction outcomes as well as their utility function. At time t, bidder i compares the performance of the proposed strategy from the algorithm with the benchmark strategy — either from external experts or from itself when it swaps all action i with j in the historical action sequence. If the regret (difference in performance) reduces to zero over time, the algorithm is a no-regret algorithm, which leads the game to reach a correlated equilibrium.

Best-response algorithms

There are two types of adversarial learning environments [32]: adaptive and non-adaptive adversarial learning. A non-adaptive adversary draws from the same distribution π_{-i}, oblivious of the actions played by bidder i.An adaptive adversary draws actions π^t_{-i} based on historical records $((b_i^1, p_i^1), \cdots, (b_i^{t-1}, p_i^{t-1}))$, and the distribution of their actions π^t_{-i} is time-variant. An adaptive environment is non-stationary in nature, which poses convergence and learning efficiency problems for a learning algorithm.

The best-response algorithms try to address the adaptive adversary issue by finding the optimal response to the opponents to maximize utility. However, these algorithms cannot provide a convergence guarantee in complex environments.

MARL algorithms in this category include e.g. WoLF-IGA [29], which converges with stationary opponents by using a variable learning rate; [151] extends the Q-learning algorithm to a non-stationary environment where the Q-values also depend on other agents' discrete actions and shows that the resulting policy is the policy with the highest utility. To combine both rationality and convergence, authors of [18] propose an RL algorithm that is best response, but converges against stationary opponents and achieves constant bounded expected regret against non-stationary opponents.

Infinite action space

Another dimension of complexity comes from a continuous (i.e. infinite-dimensional) action space. For example [13], in an adaptive adversarial multi-armed bandit problem with N strategies and T time steps, the convergence rate to the best strategy is on the order of $O(\sqrt{\frac{\log N}{T}})$, which is only feasible when N is relatively small. Previous research on online learning in continuous action space is done with both no-regret [97] and best-response [61, 115] algorithms. In this thesis, the action space is the continuous bidding price. The author uses the results from previous research to improve the convergence property of the proposed algorithms.

Independent vs. joint MARL

The simplest form of MARL is independent learning, when an agent treats all opponents as a non-stationary environment and learns its model independently from them [61, 157]. This method reduces the modeling and computation complexity; however, it does not guarantee an equilibrium and independently trained models have overfitting problems [80]. The authors of [80] study the effect of overfitting and compare the performance of independent learning and joint learning. They propose to simulate an empirical game to discover strategies for the full game and estimate their utilities, then use a RL algorithm to learn a meta-strategy for selecting policies.

Joint learning does not have to be cooperative. Authors of [115] propose ways to convert system-level utilities into individual ones, and the system, as a potential game, converges to Nash equilibria.

2.4 Conclusion of the chapter

This chapter discusses existing techniques step by step, from the basic form of second-price auction to the auction mechanism proposed in the next chapter. The author will use the technique of a simultaneous single-item combinatorial second-price auction, combined with a potential game, with consideration of a continuous action space and non-stationary environment to model the problem in the following chapters. Furthermore, the author will use classic single-agent and multi-agent RL algorithms introduced in this chapter as the basic blocks for the proposed learning algorithms.

Chapter 3

Application scenario and simulation environment

3.1 Vehicular network (V2X)

For evaluation of the proposed algorithms, the author further concretizes the system model introduced in Sec. 1.2.1 towards a vehicular network (V2X). V2X applications are characterized by huge number of users, dynamic nature and diverse Quality of Service (QoS) requirements [95]. They are also computation-intensive, e.g., inferring from large neural networks [63] or solving non-convex optimization problems [15, 42]. These applications currently reside in the vehicle's onboard units (OBU) for short latency and low communication overhead. Even with companies such as Nvidia developing OBUs with high computation power [107], post-production OBU upgrades are typically not commercially viable; and irrespective of local OBU power, the ability to offload tasks to edge/cloud via multi-access edge computing (MEC) devices increases flexibility, protecting vehicles against IT obsolescence. Hence, offloading computation requirements to edge or cloud devices is a key technique for future V2X scenarios [4, 7, 22, 159]. We hence need to solve a computation-resource allocation problem among multiple distributed vehicles.

Currently, computation offloading decisions are strictly separated between the user and the operating side [90]. Users decide what to offload to optimize an individual goal, e.g., latency [16] or energy efficiency [86]. Apart from expressing their preference through a predefined, static and universal QoS matrix [94], users cannot influence how their tasks are prioritized. The operating side centrally prioritizes tasks and decides resource allocation to optimize a system goal that is based on the QoS matrix, but not always the same as the users' goals, e.g., task maximization [40] or load balancing [147].

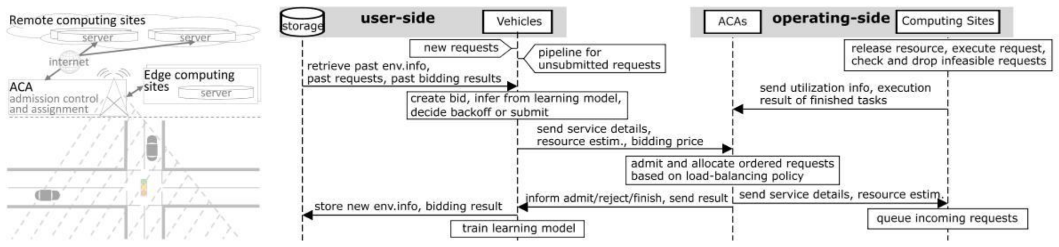
This separation poses problems for both user and operating sides, especially in the V2X context. V2X users have private goals [133], are highly autonomous [93], are reluctant to share information or cooperate, and are disobedient to a central planner [48]. They want flexible task prioritization and influence on resource allocation without sharing private information [82]. On the operating side, edge cloud computing architecture introduces signaling overhead and information delay in updating site utilization [90]; coupled with growing user autonomy and service customization, traditional centralized optimization methods for resource allocation become challenging due to unavailability of real-time information and computational intractability.

We, hence, need an interaction mechanism between user and operating sides based on incentives, not rules, and an algorithm that makes decentralized decisions with partial and delayed information in a dynamic environment. There are several challenges with such a mechanism. Users may game the system, resulting in potentially worse outcomes, both overall and individually [108] — the first challenge is how to incentivize user behavior such that users' private goals are aligned to the system goal while preserving user autonomy. The second challenge is finding an algorithm that efficiently learns from partial information with just enough feedback signals, keeping information sharing at a minimum. The third challenge is to trade off between optimality and convergence, while keeping computation and communication complexity tractable [48].

To summarize: the distributed and dynamic environment of the V2X network is a challenging but very interesting application scenario for this study. It is an instance of the general model described in Sec. 1.2.1. Therefore, it meets the requirement of a realistic testbed for the proposed algorithms. In the following chapters, the author will use V2X as the example environment and V2X applications such as motion planning and image segmentation as example applications.

3.2 V2X system model

Corresponding to Fig. 1.1 in Sec. 1.2.1, in V2X, user-side vehicles act as *bidders* to request services; operating-side admission control and assignment (ACA) units (e.g., road side units or base stations) are *auctioneers*, they control admission of service requests and assign them to different computing sites — the *commodity sellers*, which own resources and execute services [152] (Fig. 3.1a). The author of this thesis proposes changes only to 1) the algorithm deciding admission and assignments, and 2) the in-


Remote computing sites
server
server
internet
ACA
admission control
and assignment
Edge computing
sites
server
storage
user-side
Vehicles
ACAs
operating-side
Computing Sites
new requests
pipeline for
unsubmitted requests
retrieve past env.info,
past requests, past bidding results
create bid, infer from learning model,
decide backoff or submit
send service details,
resource estim., bidding price
release resource, execute request,
check and drop infeasible requests
send utilization info, execution
result of finished tasks
admit and allocate ordered requests
based on load-balancing policy
inform admit/reject/finish, send result
send service details, resource estim.
store new env.info, bidding result
train learning model
queue incoming requests

(a) Topology (b) Message sequence

Figure 3.1: V2X system model

teraction mechanism. In addition, most signaling needs in the proposed approach are covered by the ISO 20078 standard on extended vehicle web services [5]; additional fields required to pass bidding information are straightforward to implement. Channel security is not the focus of this study.

Based on the abstract concepts given in Sec. 1.2.1, the corresponding V2X entities are explained below.

3.2.1 Service request as bid

The cloud-native paradigm decomposes services into tasks that can be sequentially deployed [6]. A service request comprises 1) a task chain, with varying number, type, order and resource needs of tasks, 2) a deadline, and 3) priority of the service request. The author of this thesis considers a system with custom-tailored services placed at different computing sites in the network; the properties of these services are initially unknown to the computing sites. This enables extension to use cases into new areas, e.g., self-driving [4, 7]. The thesis considers independent services, e.g., in self-driving, segmentation and motion planning services can be requested independently. Note that although services are independent, the tasks of a service depend on each other.

The author conceives of a vehicle's service request as a *bid* in an auction. Besides the service details, a bid includes the bidding price and the vehicle's estimated resource needs.

3.2.2 User side: bidders

The thesis focuses on the behavior of vehicles, conceived of as *agents*. A vehicle acts autonomously and privately: it shares no information with other vehicles and only very limited information with the ACA (Sec. 4.4.1 and 4.5.1). A vehicle is mapped to the *bidder* in an auction game (see Sec. 1.2.1). Same as the bidder introduced in Sec. 1.2.1, its decision objective is to maximize average utility from joining the auction, and it has to balance between two options: i) back off [33] and try later or ii) submit the bid immediately to the ACA. In this thesis, 1) vehicles are incentivized to balance between backoff and bidding through a cost factor; 2) a suitable backoff time is learned from state information, not randomly chosen; 3) learning is only based on information visible to the vehicle.

The *bidding price* used by the vehicle in the auction is an abstraction of vehicle's prioritization of their service requests: a higher bidding price means a higher priority. The abstraction makes it possible to compare priority of different custom-tailored service requests. In this thesis, the upper bound for the bidding price is given to each vehicle at initialization as a *budget*. In Chapter 4, the budget is reset after every auction round; in Chapters 5 and 6, the budget can be accumulated over time for a fixed duration. In reality, such an economic incentive as a given budget to each vehicle may need to be managed more carefully, but it is not within the scope of this work.

The thesis studies the learning algorithm in each vehicle. Passive, non-learning vehicles are used as benchmark to quantify how learning affects performance. Learning essentially sets the priority of a service request. This priority is used by the ACA to order requests; it is simply constant for non-learning vehicles, resulting in a first-in, first-out processing order.

3.2.3 Operating side: auctioneer and commodity sellers

The ACA unit and computing sites are the operating side (Fig. 3.1b). The ACA estimates available capacity of the various computing sites, based on which the ACA makes admit/reject decisions for each service request. Upon admission, it assigns the request to a computing site according to a load-balancing policy. Due to information delay, execution uncertainty, system noise, etc., the resource utilization information at different sites is not immediately available to the ACA unit — specifically, when each computing site would serve more than one ACA unit or share its resource with other applications, their actual resource availability may differ from the ACA's estimate, therefore, the ACA's admit/reject decision is only provisional. If the computing site decides that an assigned service request cannot be executed before its deadline, the computing site drops the service request and sends an "offloading failure" result to ACA. If the ACA estimates that a service request cannot be executed by any computing sites before its deadline, the service request is rejected, and vehicles can rebid for a maximum number of times. Vehicles receive feedback on bidding and execution outcome, payment, and resource utilization (Sec. 4.4.1). Any service request that is not executed before its deadline is counted as an “offloading failure”.

The operating side does not have *a priori* knowledge of the type, priority, or resource requirement of service requests. For example, if at run time, a site receives a previously unknown service, it uses an estimate of resource needs provided by the vehicle. Over time, a site updates this estimate from repeated executions of the same service. Extension to a more sophisticated form of learning is left to future work.

The total service time of a request is the sum of processing, queueing and transmission time. Each computing site may offer all services but with different resource profiles (i.e., amount and duration needed), depending on the site’s configuration. Site capacity is specified in abstract time-resource units: one such unit corresponds to the volume of a request served in one time unit at a server, when given one resource unit.

3.3 Simulation setup

The author of this thesis develops a Python discrete-event simulator with varying number of vehicles of infinite lifespan, one road side unit with ACA and one edge computing site, and one remote computing site (extension to multiple ACA units and computing sites is left to future work). The edge and remote sites have different resource profiles. To imitate a realistic, noisy environment, the remote site is some distance to the ACA unit, such that the state information is only updated with a non-negligible delay caused by data transmission. The author also adds a small, normally distributed noise to this delay (truncating negative values), as well as to the actual resource required for a service. Each vehicle is randomly and independently initialized with a budget of “high” or “low” with 50% probability. For the operating-side load-balancing policy, the author applies state-of-the art resource-intensity-aware load balancing (RIAL) [132] with slight modifications. The method dynamically balances load among computing sites through resource pricing that is correlated to the site’s load, and loads are shifted to “cheaper” sites. Finally, the author compares the performance of active agents (the proposed algorithms on the user side, RIAL on the operating side) to passive agents (only RIAL on the operating side).

Table 3.1: Realistic setup differences

Environment Parameters	Training Setup	Test Setup
service request type	F1: 80 resource units (abstracted from CPU and memory usage) needed within 100 time steps (milliseconds) F2: 80 resource units needed within 500 time steps	
service arrival rate	F1: every 100 time steps; F2: every 500 time steps. Arrival of the first request is uniform randomly distributed in the first 100/500 steps.	
data size	uplink: F1: 0.4 Mbit, F2: 4 Mbit; downlink: F1: 0 (negligible), F2: 0.4 Mbit	
latency	802.11ac: 65m radius, maximum channel width 1.69 Gbps, throughput=$-26 \times$ distance + 1690 Mbps [128]	
computing site capacity	60 resource units per time step (low contention)	10-20 units (high contention)
vehicle arrival rate	1 every 2.2 seconds	1 every 1 second
vehicle speed	10 km/h when driving	30 km/h
vehicle count	22-29 and slow changing	14-30 and bursty[a]

[a]The author regulates burstiness by adjusting vehicle speed, arrival rate and traffic light phases

3.3.1 Realistic setup for all algorithms

All of the proposed algorithms are evaluated in a realistic V2X setup: a 4-way traffic intersection with realistic mobility model for vehicles and with random incoming service requests. In this setup, the author adopts the data patterns of segmentation and motion planning applications extracted from various self-driving data projects [43] or referenced from relevant studies [30, 35]. The author also uses Simulation of Urban Mobility (SUMO) [20] to create a more realistic mobility model of a single junction with a centered traffic light; the junction is an area downloaded from open street map. Assuming the 802.11ac protocol, the ACA unit is placed in the middle of the intersection, and the edges are limited to be within 65m of the ACA. The streetmap is with two lanes per street per direction. SUMO uniformly at random creates a vehicle at any one of the four edges.

Parameters of the realistic setup are according to [30, 35, 43] and listed in Table 3.1.

3.4 Performance metrics

This section introduces the common performance metrics used for performance evaluation in the following chapters.

- **Offloading failure rate (OFR):** Ratio of failed offloading requests, either rejected by ACA or not executed before deadline, compared to all requests. For simplification, this thesis ignores the failed service requests that are admitted by the ACA but not executed by computing sites before deadline. Empirical results confirm that this is a close approximation of OFR, because with the proposed algorithms, the author achieves a system responsiveness (i.e., the ratio of admitted requests successfully executed before deadline) of 99%.
- **Resource utilization:** Ratio of resources utilized at computing sites = (sum of utilized resource units in all resource types and all computing sites in the current time step) / (sum of total resource units in all resource types and all computing sites at any time). By measuring resource utilization, we get a better picture of the algorithms' load-balancing effect.
- **Rebidding overhead:** If a bid is rejected before deadline, the vehicle can bid again. More rebidding causes communication overhead, but less rebidding reduces the chance of success. The thesis studies this tradeoff, comparing the average number of actual rebiddings per vehicle within maximum permitted rebidding (MP). As an example: if a bidder bids three times until the service request is admitted, there are two rebiddings. Ideally, the bidder would need zero rebidding. MP is a metric to avoid unlimited rebidding.
- **Payoff:** Individual vehicle's valuation of the service request minus the second-price payment, cost of bidding and cost of backoff. Payoff does not have any practical implication and is only used to incentivize certain vehicle behaviors. In a repeated auction, we expect individual payoff to increase over time as vehicles learn the best bidding strategy. By measuring payoff, we see how well the vehicles are learning.
- **Fairness:** System overall fairness is defined as the J-index [68] of payments over the last $\mathcal{T}$ time steps: Fairness = $\frac{(\sum\limits_{m} \sum\limits_{t-\mathcal{T}} tp_m^t)^2}{|M| \sum\limits_{m} (\sum\limits_{t-\mathcal{T}} tp_m^t)^2}, \forall m \in M$, where M are the vehicles. J-index is commonly used to measure fairness in networking. It is the reciprocal of the original normalized Herfindahl–Hirschman Index [122] for measuring market concentration.

Chapter 4

Short-term Learning

4.1 Motivation

As mentioned in Sec. 3.1, the challenges of distributed decision making in V2X resource allocation are **C1**) how to incentivize user behavior while preserving their autonomy; **C2**) how to find an algorithm that efficiently learns from partial information with limited feedback signals; **C3**) how to trade off between optimality and convergence, while keeping computation and communication complexity tractable.

Although distributed methods are the focus of this thesis, the author mentions some centralized optimization methods here (i.e., on the operating side of the V2X model, see Fig. 3.1b). Centralized approaches, such as [8, 79] for resource allocation and [36, 40, 88] for offloading are suited to core-network and data-center applications when a central ACA can be set up and data can be relatively easily obtained. Previous studies of decentralized systems address some of the issues in centralized approaches. References [24, 126] propose distributed runtime algorithms to optimize system goals but disregard user preferences. [79] uses heuristics at runtime and [8] decouples the central optimization problem into smaller problems, but these solutions still assume complete information which is not available in the setting of this thesis.

Authors of [50] and [57] demonstrate the advantage of using economic models for many types of multi-resource allocation problems. Efforts are made to jointly optimize private and system goals through game theoretic approaches — although they naturally deal with *decentralized incentives*, they often require complete information of the game to *centrally execute* the desired outcome. E.g., reference [37] proposes a decentralized offloading game and solves for the Nash equilibrium through an incremental decision update mechanism that is analogous to steepest descent. The mechanism converges fairly quickly because the variable is one-dimensional. The update mechanism requires all users to broadcast all of their decisions for all users to reach equilibrium. Reference [34] models the non-cooperative users' offloading decisions as a generalized Nash equilibrium game and solves the convex optimization problem numerically. Although the problem can be solved asynchronously and distributedly, it is a static model in which all user and node profiles are known *a priori*. When the user or node profiles are not homogeneous, the model's complexity increases as the number of participants grow. Other game-theoretic methods such as [38, 77, 78] only consider cooperative resource sharing or offloading. Reference [130] only considers discrete actions. Reference [82] learns with partial information, but it reduces complexity by assuming single service type and arrival rate. The approach of this thesis also differs from [74] as it considers a multi-dimensional continuous action space with multiple service types, and both cooperative and competitive behaviors. Sec. 2.3.2 highlighted the advantage of some learning algorithms in non-stationary environment [29, 142, 151], as well as their limitations [80, 157]. The proposed approach in this thesis is designed to reduce computation complexity and improve convergence property in a non-stationary environment.

In this chapter, the author proposes a distributed decision-making mechanism based on second-price sealed-bid auctions (see Sec. 2.2.3) that successfully addresses the challenges **C1** to **C3**, using the V2X context as an example. The mechanism utilizes the feedback signal to incentivize cooperative behavior among competitive, self-interested agents and speed up learning. The author proves that in the stationary case, the outcome of this mechanism is a Nash equilibrium (NE) and a maximization of social welfare; under specific conditions, it is also a *Pareto-optimal* allocation of resources (**C1**). For the dynamic case, the author proposes a MARL algorithm, for its ability to learn with partial, noisy and delayed information, and a single reward signal (**C2**). Specifically, the core RL algorithm in this thesis learns the best-response strategy updated in a fictitious self-play (FSP) method (Sec. 4.5.1). FSP addresses strategic users' adaptiveness in a dynamic environment by evaluating state information incrementally and by keeping a weighted historical record [60]. It is easier to implement than the method proposed in [29], especially with a large state and action space (**C3**).

4.2 Chapter outline

Table 4.1 lists the focuses of this chapter. As the author introduces more algorithms in the subsequent chapters, the table will contain more information for comparison between the algorithms and setups.

Sec. 4.3 further extends the step-by-step problem formulations from Sec. 2.2.3, making the problem concrete to finally reflect the complexity of the short-term single-objective resource allocation problem the author addresses in this thesis. Sec. 4.4 describes the interaction mechanism design and the corresponding utility function, and provides proof that in a stationary environment, the mechanism has Nash equilibria, is a maximization of social welfare and a Pareto-optimal way of resource allocation. Sec. 4.5 introduces the first proposed algorithm in this thesis for learning to optimize a short-term, single objective in a dynamic environment. The evaluation results of the first proposed MARL algorithm are in Sec. 4.6. Sec. 4.7 concludes the chapter.

The training and evaluation environments and simulation parameters are as described in Chapter 3. This includes a concrete resource allocation system modeled after vehicular networks (Sec. 3.1 and 3.2), simulation setup, performance metrics, and the realistic setup with mobility data (Sec. 3.3).

Table 4.1: Chapter4 outline

	Chapter4: DRACO
Alg. type	short term
Obj. type	single objective
Synthetic seup	Simulated second-price forward auction in V2X application scenario, with varying resource capacity, service request types, number of rebidding (Sec. 4.6.1). Purpose: analyze algorithm performance with randomized inputs and a wide range of environment parameters.
Realistic setup	Same as in Sec.3.3.1. Purpose: analyze and compare performance in simulated traffic.

4.3 Problem formulation

The author formulates the problem based on Sec. 2.2.3 and 2.2.4. Table 4.2 summarizes the notation.

Let M be the set of vehicles (bidders) and K the set of commodities (service types), each type with n_k^t available service slots at time t in computing sites. Bidder $m \in M$'s demand for each service type $k \in K$ at t is represented by a bid indexed $i_k^t \in I$. From its bidding strategy π_m, bidder m draws its actions $\alpha_m^t = \{\alpha_{m,k}^t \in \{0,1\}, \forall k \in K\}$ and

Table 4.2: Short-term problem symbol definition

Sym.	Description	Sym.	Description
$k \in K$	service type/commodity	n_k	k's availability
$i \in I$	service request/bid	v	bid value
$m \in M$	vehicle/bidder	p	payment
z	bidding outcome	u	utility
α	backoff decision	b	bidding price
c	lost bid penalty	q	backoff cost
β	utilization	B	budget

$\mathbf{b}^t_m = \{b^t_{m,k} \in \mathbb{R}_+, \forall k \in K\}$ for each service type. α^t_m is the vector of backoff decision, $\mathbf{b}^t_m$ is the vector of bidding price. More specifically, bidder m's options for each bid i^t_k are: 1) back off ($\alpha^t_{m,k} = 0$) with a backoff cost $q^t_{m,k}$, or 2) bid ($\alpha^t_{m,k} = 1$) with the bidding price $b^t_{m,k}$.

From bidder m's perspective, the competing bidders (denoted $-m$) draw their actions from a joint distribution π^t_{-m} that is an unknown function of $(\mathbf{p}^1, \cdots, \mathbf{p}^{t-1})$, where $\mathbf{p}^t \in \mathbb{R}^{|K|}_+$ is the vector of final prices at the end of time t. Bidder m observes the new $\mathbf{p}^t$ as feedback. If bidder m wins its bid i^t_k indicated by bidding outcome $z^t_{m,k} = 1$, it pays p^t_k to the auctioneer. If it loses ($z^t_{m,k} = 0$), it pays 0 to the auctioneer but has a cost associated with losing the bid, denoted by $c^t_{m,k}$.

The auction repeats for T periods, in every auction round, bidder m's utility (Sec. 2.2.3) is $u^t_m(\alpha^t_m, \mathbf{b}^t_m, \mathbf{p}^t, \mathbf{z}^t_m, \mathbf{c}^t_m, \mathbf{q}^t_m)$, its goal is to maximize the long-term utility:

$\mathcal{U} = \frac{1}{T}\sum\limits_{t=1}^{T} u^t_m, T \longrightarrow \infty$. An illustration of the auction mechanism is in Fig. 4.1.

For any k, when availability $n^t_k < \sum\limits_{m\in M} \alpha^t_{m,k}$, there is more demand than available service slots. We call such a situation "high contention". When $n^t_k \geq \sum\limits_{m\in M} \alpha^t_{m,k}$, we call it "low contention". In a dynamic environment, n^t_k depends on utilization at $t-1$ and existing demand at t. Due to noise and transmission delay in a realistic environment, this information is inaccurate and outdated when it becomes available to the ACA unit for admission control (Fig. 1.1).

Ideally, an auction is incentive-compatible. Unfortunately, with budget constraint and costs, the second-price auction considered here is no longer incentive-compatible. But the author still uses this type of auction as it maximizes social welfare and optimally allocates resources (Sec. 4.4.2 and 4.4.3). The payment signal is used as additional feedback to aid the bidders' learning process (Sec. 4.5.2).

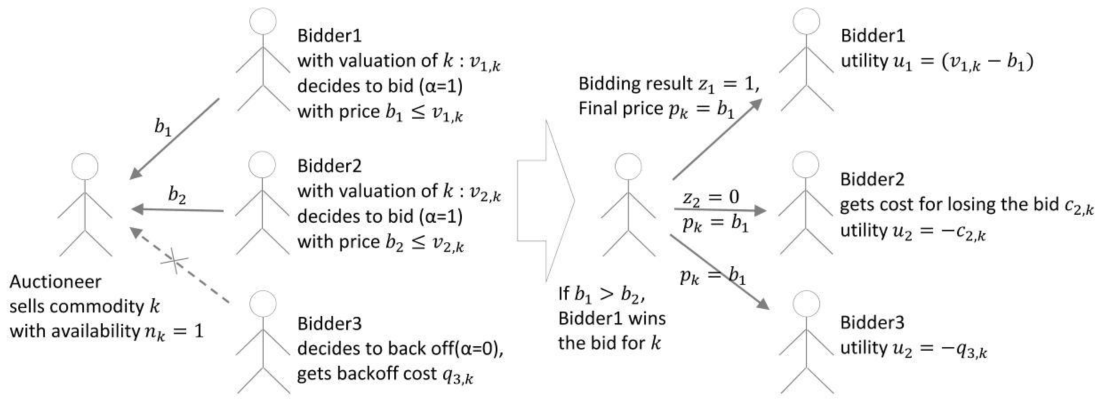

Bidder1
with valuation of k : $v_{1,k}$
decides to bid (α=1)
with price $b_1 \leq v_{1,k}$
b_1
Bidder2
with valuation of k : $v_{2,k}$
decides to bid (α=1)
with price $b_2 \leq v_{2,k}$
b_2
Auctioneer
sells commodity k
with availability $n_k = 1$
Bidder3
decides to back off(α=0),
gets backoff cost $q_{3,k}$
Bidding result $z_1 = 1$,
Final price $p_k = b_1$
$z_2 = 0$
$p_k = b_1$
$p_k = b_1$
If $b_1 > b_2$,
Bidder1 wins
the bid for k
Bidder1
utility $u_1 = (v_{1,k} - b_1)$
Bidder2
gets cost for losing the bid $c_{2,k}$
utility $u_2 = -c_{2,k}$
Bidder3
utility $u_2 = -q_{3,k}$

Figure 4.1: An illustration of the auction mechanism

4.4 Theoretical results in a stationary environment

Sec. 4.4.1 defines bidder's utility function; in Sec. 4.4.2 and 4.4.3, the author proves the existence of NE, maximization of welfare and Pareto optimality in the stationary case, for both low and high contention situations. Notation is in Table 4.2. For readability, we drop notation for time t in the stationary case.

4.4.1 Utility function

In this section, we first build up the utility function based on the payoff of classic second-price auction. Then, we add costs for backoff and losing the bid, incentivizing tradeoff between higher chance of success and lower communication overhead. Finally, we add the system resource utilization goal to the individual utility.

This thesis assumes each bidder has a given private, fixed valuation of commodity type k denoted $v_{m,k}$ that is 1) linear to the bidder's estimated resource needs for k and 2) within the budget. The first condition guarantees Pareto optimality (Corollary 4.6.1); the second avoids overbidding under rationality (Theorem 4.2). The thesis does not consider irrational or malicious agents, e.g., whose goal is to reduce social welfare even if their own individual outcome may be hurt.

The author of this thesis uses the second-price auction mechanism (Sec. 2.2.3) as a solution to the problem described in Sec. 4.3. In each auction round, the auctioneer records the price b_k^* of the highest losing bid among all bidders, which is also the $(n_k + 1)$th highest overall bidding price. For $n_k = 1$ this would be the second highest price, hence the name “second-price auction”. The final price $p_{m,k} = b_k^*$ is broadcast to all bidders, which is also the amount the n_k winning bidders (ties are broken randomly) pay to the auctioneer. A bidder’s utility is:

$$u_{m,k} = \alpha_{m,k} \cdot (z_{m,k} \cdot (v_{m,k} - p_{m,k}) - (1 - z_{m,k}) \cdot c_{m,k}) + (1 - \alpha_{m,k}) \cdot q_{m,k} \tag{4.1}$$

Adding cost $c_{i,k}$ penalizes rebidding and gives an incentive only to bid when there is a need and chance of success, especially in high contention. This cost reflects actual overhead associated with rebidding, e.g., system-wide communication overhead, or individual energy overhead. Together with $q_{i,k}$, the bidder is incentivized to trade off between long backoff time and risky bidding. In this chapter, the two cost factors c and q are hyperparameters to initialize the bidders. Chapter 6 lets each bidder decide independently and in private its risk preference for backoff and losing the bid.

To further align the bidder’s objective with system overall objective (**C1**), we include system resource utilization β in the utility. This incentivizes bidders to minimize system utilization. In this chapter, bidders are initialized with a given preference weight W for the system resource utilization objective. In Chapter 6, each bidder has its own preference for the objective, and a preference of 0 means the bidder does not consider the objective. Evaluation result in Sec. 6.6 shows that consideration of system objectives helps the bidders achieve their individual objective as well.

The complete utility definition is:

$$u_m = \sum_{k \in K} u_{m,k} + W \cdot (1 - \beta), \forall m \in M \tag{4.2}$$

To calculate Eq. 4.2, the bidder needs only these feedback signals: bidding outcome z, final price p and system utilization β, addressing **C2**.

4.4.2 Low contention

Low contention is much more common in networking and presumably also in future V2X applications as resources are often abundantly available. Although drivers would not know when there would be low contention, with a learning algorithm, they can "guess" the situation from observations and past bidding results, and learn the best action α in such situations.

Notation for commodity type k is omitted. The set of actions A for bidder m is α_m. The author will use the concept of potential functions [99] to show that in low contention, the proposed interaction mechanism is a potential game with NE:

Definition 4.1. *$G(M, A, u)$ is an exact potential game if and only if there exists a potential function $\phi(A) : A \longrightarrow \mathbb{R}$ s.t. $\forall m \in M$, $u_m(\alpha_m, \alpha_{-m}) - u_m(\alpha'_m, \alpha_{-m}) = \phi_m(\alpha_m, \alpha_{-m}) - \phi_m(\alpha'_m, \alpha_{-m}), \alpha \in A$.*

Lemma 4.2. *Players in a finite potential game that jointly maximize a potential function end up in NE.*

Proof. See [99]. □

Theorem 4.1. *Bidders with utility as Eq. 4.2 participate in a game as described in Sec. 4.3 in low contention, the game is a potential game, and the outcome is an NE.*

In low contention, $p = 0$, as all bids are accepted. u_m is reduced to

$u_m(\alpha_m, \alpha_{-m}) = (v_m + q_m) - \alpha_m q_m + W\Big(1 - \sum\limits_{j \in M} \alpha_j \cdot \frac{\omega_j}{C}\Big)$, where $\omega_j \in \mathbb{R}^{|K|}$ is each bidder's resource requirement, C is system capacity. Thus, the auction is reduced to a potential game with discrete action space $\alpha_m \in \{0, 1\}^{|K|}$ and potential function

$$\phi(\alpha_m, \alpha_{-m}) = \sum_{j \in M} (v_j + q_j - \alpha_j q_j) + W\Big(1 - \sum_{j \in M} \alpha_j \cdot \frac{\omega_j}{C}\Big), \forall m \in M.$$

Next, we prove that $u_m(\alpha_m, \alpha_{-m}) - u_m(\alpha'_m, \alpha_{-m}) = \phi(\alpha_m, \alpha_{-m}) - \phi(\alpha'_m, \alpha_{-m})$, and hence it is a potential game, and bidders maximizing their utilities u_m also maximize the potential function ϕ. Since $\alpha_m \in \mathbb{R}^{|K|}$, it is a finite potential game. According to Lemma 4.2, the outcome is an NE.

Proof. To simplify, we substitute with:

$$\begin{cases} Q_m = v_m + q_m \\ A_m = \alpha_m q_m \\ A_{-m} = \sum\limits_{j \in M, j \neq m} \alpha_j q_j \\ B_m = \alpha_m \omega_m \\ B_{-m} = \sum\limits_{j \in M, j \neq m} \alpha_j \omega_j \end{cases}$$

and rewrite:

$$\begin{cases} u_m(\alpha_m, \alpha_{-m}) = Q_m - A_m + W - \frac{W}{C}(B_m + B_{-m}) \\ u_m(\alpha'_m, \alpha_{-m}) = Q_m - A'_m + W - \frac{W}{C}(B'_m + B_{-m}) \\ \phi(\alpha_m, \alpha_{-m}) = \sum\limits_{j \in M} Q_j - (A_m + A_{-m}) + W - \frac{W(B_m + B_{-m})}{C} \\ \phi(\alpha'_m, \alpha_{-m}) = \sum\limits_{j \in M} Q_j - (A'_m + A_{-m}) + W - \frac{W(B'_m + B_{-m})}{C} \end{cases}$$

$$\begin{aligned} \implies u_m(\alpha_m, \alpha_{-m}) - u_m(\alpha'_m, \alpha_{-m}) &= -(A_m - A'_m) - \frac{W}{C}(B_m - B'_m) \\ &= \phi(\alpha_m, \alpha_{-m}) - \phi(\alpha'_m, \alpha_{-m}) \end{aligned}$$

□

In low contention, the computation offloading problem becomes a potential game. This enables us to use online learning algorithms such as in [115] that converge regardless of other bidders' behaviors. The NE is a local maximization of the potential function: each bidder finds a balance between its backoff cost and the incentive to reduce overall utilization. Empirical results in Sec. 3 confirm that over time this results in a more balanced load.

4.4.3 High contention

In high contention, α is used in a repeated auction to avoid congestion and ensure better reward over time. To simplify the proofs, we will consider only the time steps where $\alpha = 1$ (bidder joins auction). In high contention, we assume a high enough utilization β, such that the last term in Eq. 4.2 can be omitted, to further simplify the utility function in the proof. u_i is reduced to:

$$u_m = \sum_k \Big(z_{m,k} \cdot (v_{m,k} - p_{m,k}) - (1 - z_{m,k}) \cdot c_{m,k} \Big) \tag{4.3}$$

Theorem 4.2. *In a second-price auction, where bidders with utility as Eq. 4.2 compete for service slots as commodities in high contention, 1) bidders' best response is of linear form, 2) the outcome is an NE and 3) welfare is maximized.*

To prove Theorem 4.2, the author will 1) describe the basic model of the second-price auction; 2) prove the linear form of the best response; 3) prove the existence of Nash equilibrium and maximization of social welfare.

Basic model

For simplicity, we use $|M| = 2$ and $|K| = 1$. It is an extension from [135]. Unlike [135], this thesis includes in the utility definition the second-price payment and cost for losing a bid. Based on [135], it can also be easily extended to multiple bidders.

2 bidders receive continuously distributed valuations $v_m \in [\text{low}_m, \text{high}_m], m \in \{1, 2\}$ for 1 commodity and choose their strategies $f_m(v_m)$ from the strategy sets F_1 and F_2. The resulting NE strategy pair is (f_1^*, f_2^*). Any strategy function $f_m(v_m)$ is increasing in v_m, with $f_m(\text{low}_m) = f_m^{\text{low}}$ and $f_m(\text{high}_m) = f_m^{\text{high}}$, f_m^{low} and f_m^{high} are minimum and maximum values of f_m. We also assume the users have budgets B_m, and they cannot bid more than the budget. Cost for losing the bid is c_m. The inverse function of $f_m(v_m)$ is:

$$\begin{cases} h_m(y_m) = h_m^{\text{low}}, \text{if } y_m \leq f_m^{\text{low}} \\ h_m(y_m) = f_m^{-1}(y_m), \text{if } f_m^{\text{low}} < y_m < f_m^{\text{high}} \\ h_m(y_m) = h_m^{\text{high}}, \text{if } y_m \geq f_m^{\text{high}} \end{cases}$$

where h_m^{low} and h_m^{high} are minimum and maximum values of the inverse function h_m.

For a given f_1, if bidder 2 chooses a bidding function f_2, according to Eq. 4.3, the expected utility for bidder 2 is

$$u_2(f_1, f_2) = \mathbb{E}_{v_1,v_2}[(v_2 + c_2) \cdot 1_{f_2(v_2)\geq f_1(v_1)}] - \mathbb{E}_{v_1,v_2}[f_1(v_1) \cdot 1_{f_2(v_2)\geq f_1(v_1)}] - c_2 \tag{4.4}$$

where $1_{f_2(v_2)\geq f_1(v_1)} = 1$, if $f_2(v_2) \geq f_1(v_1)$, otherwise 0.

To simplify, we define

$$\begin{cases} E_1 & = \mathbb{E}_{v_1,v_2}[(v_2 + c_2) \cdot 1_{f_2(v_2)\geq f_1(v_1)}] \\ E_2 & = \mathbb{E}_{v_1,v_2}[f_1(v_1) \cdot 1_{f_2(v_2)\geq f_1(v_1)}] \end{cases} \tag{4.5}$$

Hence, $u_2(f_1, f_2) = E_1 - E_2 - c_2$. E_2 is the expected second price payment when bidder 2 wins, and the payment should be no greater than $\min(f_2^{\text{high}}, B_2)$. Since to avoid overbidding, the thesis assumes $f_2^{\text{high}} \leq B_2$, the set of feasible bidding functions for bidder 2 given f_1 is $S_2(f_1) = \{f_2 \in F_2 | u_2(f_1, f_2) \geq 0, E_2 \leq f_2^{\text{high}}\}$.

For the condition $u_2(f_1, f_2) \geq 0$ to hold, at any point where $1_{f_2(v_2) \geq f_1(v_1)} = 1$, we need to have $v_2 \geq f_1(v_1)$, which is a sufficient condition of $u_2(f_1, f_2) \geq 0$. This is true because f_2 is bidder 2's bidding signal, to avoid overbidding, $f_2(v_2) \leq \min(f_2^{\text{high}}, v_2)$, therefore $f_1(v_1) \leq v_2$. We thus simplify the above equation to: $S_2(f_1) = \{f_2 \in F_2 | E_2 \leq f_2^{\text{high}}\}$.

The problem is formulated into a utility maximization problem: $\max\limits_{f_2 \in S_2(f_1)} u_2(f_1, f_2)$. f_2 is a best response of bidder 2, if $u_2(f_1, f_2) \geq u_2(f_1, f_2'), \forall f_2' \in S_2(f_1)$. An NE strategy pair (f_1^*, f_2^*) has the selected strategies as each other's best responses and also maximizes the total utility (social welfare).

Form of the best response

Theorem 4.3. *Given bidder 1's bidding strategy* $\begin{cases} f_1 \in F_1 \\ f_1(h_1^{low}) = f_1^{low} \\ f_1(h_1^{high}) = f_1^{high} \end{cases}$*, bidder 2's best response has the form* $\begin{cases} f_2(v_2) \leq f_1^{low} & \text{for } v_2 \in [h_2^{low}, \theta^{low}] \\ f_2(v_2) = j_2 \cdot v_2 + d_2 & \text{for } v_2 \in [\theta^{low}, \theta^{high}] \\ f_2(v_2) \geq f_1^{high} & \text{for } v_2 \in [\theta^{high}, h_2^{high}] \end{cases}$*,*

where $\theta^{low}, \theta^{high} \in [h_2^{low}, h_2^{high}]$ *and* $j_2\theta^{low} + d_2 = f_1^{low}, j_2\theta^{high} + d_2 = f_1^{high}$.

Proof. Given f_1 and bidder 2's bid y_2, probability that bidder 2 wins the bid is:

$P_2^{win}(y_2) = P(f_1(v_1) \leq y_2) = P(v_1 \leq h_1(y_2)) = \int_{h_1^{\text{low}}}^{h_1(y_2)} \mathbf{p}_1(v_1) \mathrm{d}v_1$, where $\mathbf{p}$ is the probability density function, and P is the cumulative function.

Bidder 2's optimization problem is to find a bidding function $y_2 = f_2(v_2)$ to:

$$\begin{aligned} &\max_{f_2}(E_1 - E_2) \\ &\quad = \mathbb{E}_{v_1,v_2}[(v_2 + c_2) \cdot 1_{f_2(v_2) \geq f_1(v_1)}] - \mathbb{E}_{v_1,v_2}[f_1(v_1) \cdot 1_{f_2(v_2) \geq f_1(v_1)}] \\ &\quad = \int_{h_2^{\text{low}}}^{h_2^{\text{high}}} \int_{h_1^{\text{low}}}^{h_1(f_2(v_2))} \Big(v_2 + c_2 - f_1(v_1)\Big) \mathbf{p}_2(v_2) \mathbf{p}_1(v_1) \mathrm{d}v_1 \mathrm{d}v_2 \qquad (4.6) \\ &s.t. E_2 \leq f_2^{\text{high}} \end{aligned}$$

To solve the optimization problem, we write the Lagrangian function with multiplier λ:

$$\max_{f_2} \mathcal{L}(v_2, \lambda) = E_1 - E_2 - \lambda(E_2 - f_2^{\text{high}})$$

$$= \int_{h_2^{\text{low}}}^{h_2^{\text{high}}} \left[\int_{h_1^{\text{low}}}^{h_1(f_2(v_2))} \Big(v_2 + c_2 - (1+\lambda) f_1(v_1)\Big) \mathbf{p}_1(v_1) \mathrm{d}v_1 \right] \mathbf{p}_2(v_2) \mathrm{d}v_2 + \lambda f_2^{\text{high}} \tag{4.7}$$

Equation 4.7 provides an upper bound for Equation 4.6.

To maximize $\mathcal{L}$, for each v_2, we need to find the f_2 that maximizes

$\int_{h_1^{\text{low}}}^{h_1(y_2)} \Big(v_2 + c_2 - (1+\lambda) f_1(v_1)\Big) \mathbf{p}_1(v_1) \mathrm{d}v_1, y_2 = f_2(v_2).$

For any given v_2, the above formula is the area below the function $\ddagger = v_2 + c_2 - (1 + \lambda) f_1(v_1)$, when v_1 moves in the range from h_1^{low} to $h_1(y_2)$. As f_1 is monotonously increasing, $\ddagger$ is monotonously decreasing. Therefore, to maximize the area below $\ddagger$, $h_1(y_2)$ should simply be chosen as the intersection of $\ddagger$ and the x-axis, or $v_2 + c_2 - (1 + \lambda) f_1(h_1(y_2)) = 0$:

$y_2 = f_1(f_1^{-1}(y_2)) = f_2(v_2) = \frac{1}{1+\lambda} v_2 + \frac{c_2}{1+\lambda}, \forall y_2 \in [f_1^{\text{low}}, f_1^{\text{high}}]$,

or $v_2 \in [(1+\lambda) \cdot f_1^{\text{low}} - c_2, (1+\lambda) \cdot f_1^{\text{high}} - c_2]$.

Since $f_2(v_2)$ is monotonously increasing,

$$\begin{cases} f_2(v_2) \leq f_1^{\text{low}}, \text{ if } v_2 \in [h_2^{\text{low}}, (1+\lambda) \cdot f_1^{\text{low}} - c_2] \\ f_2(v_2) \geq f_1^{\text{high}}, \text{ if } v_2 \in [(1+\lambda) \cdot f_1^{\text{high}} - c_2, h_2^{\text{high}}] \end{cases}$$

Monotonously increasing f_1 and f_2 makes Equation 4.6 a convex problem, therefore equations 4.6 and 4.7 are equivalent, maximizing the Lagrangian function also maximizes the bidder's utility function.

Theorem 4.3 implies that the best response of bidder 1 and 2 are both of the linear form. □

Existence of Nash equilibrium

Theorem 4.4. *When best response form is $f_1(v_1) = j_1 v_1 + d_1$ and $f_2(v_2) = j_2 v_2 + d_2$, we can always find a pair (j_1, j_2) such that both bidders' bidding range $[f_m^{low}, f_m^{high}]$*

would be satisfied in NE.

Proof. An NE exists if there is a pair (j_1, j_2) that satisfies the two constraints:

$$\mathbb{E}_{v_1,v_2}[f_1(v_1) \cdot 1_{f_2(v_2) \geq f_1(v_1)}] \leq f_2^{\text{high}}, \mathbb{E}_{v_1,v_2}[f_2(v_2) \cdot 1_{f_1(v_1) \geq f_2(v_2)}] \leq f_1^{\text{high}}.$$

The following proves that such a pair exists. To simplify the proof, we choose $c_1 = c_2 = c$, given the linear best response forms and bidders' bidding functions, and symmetric to Eq. 4.5, we define

$$\begin{cases} E_3 = \mathbb{E}_{v_1,v_2}[(v_1 + c_1) \cdot 1_{j_1 v_1 \geq j_2 v_2}] \\ E_4 = \mathbb{E}_{v_1,v_2}[(j_2 v_2) \cdot 1_{j_1 v_1 \geq j_2 v_2}] \end{cases} \tag{4.8}$$

and define bidder 1's feasible strategy set as a function of j_2: $S_1(j_2) = \{j_1 \in [0, \infty) | E_4 \leq f_1^{\text{high}}\}$. Due to its linear form, and according to Eq. 4.3, bidder 1's set of best responses is:

$$\mathbf{b}_1 = \underset{f_1 \in S_1(j_2)}{\arg\max}(E_3 - E_4 - c_1) = \underset{y \in S_1(j_2)}{\arg\max}\, \mathbb{E}_{v_1,v_2}[v_1 - j_2 v_2 \cdot 1_{y v_1 \geq j_2 v_2}]$$

the utility $u_1(y) = \mathbb{E}_{v_1,v_2}[v_1 - j_2 v_2 \cdot 1_{y v_1 \geq j_2 v_2}]$ is a non-decreasing function of y defined on the set $S_1(j_2)$. To prove the existence of NE, we use Kakutani fixed point theorem.

Theorem 4.5 (Kakutani fixed point theorem [109]). *Let A be a non-empty, compact and convex subset of some Euclidean space R^n. Let $\varphi : A \rightarrow 2^A$ be an upper hemicontinuous set-valued function on A with the property that $\varphi(x)$ is non-empty, closed and convex $\forall x \in A$. Then φ has a fixed point.*

The author proves Lemmas 4.3-4.8 below, to show that the thesis meets the conditions of Theorem 4.5. Hence, $\varphi : S_1 \rightarrow \mathbf{b}_1 \in 2^{S_1}$ has a fixed point, and there exists NE (Theorem 4.4). □

Lemma 4.3. *Bidder 1 strategy set $A = S_1(j_2) = \{j_1 | \mathbb{E}_{v_1,v_2}[(j_2 v_2 + d_2) \cdot 1_{j_1 v_1 \geq j_2 v_2}] \leq f_1^{high}, j_1 \in [0, \infty)\}$, $\forall j_2 \in [0, \infty)$ is non-empty, convex, compact.*

Proof. $S_1(j_2)$ is a strategy set and naturally non-empty. The product of all players' strategy sets are therefore also non-empty. For any given j_2, any combination of a feasible strategy's parameter still creates a feasible strategy (due to its linear form). Therefore $S_1(j_2)$ is convex. The set $S_1(j_2)$ contains all of its limits, therefore it is a closed set. Due to bidding range and budget, it is also bounded. The product of all players' strategy sets are therefore closed and bounded. According to Heine-Borel Theorem, the sets are compact. □

Definition 4.4. *: A set-valued function u defined on a convex set $S_1(j_2)$ is quasiconcave if every upper level set of u is convex, or $P_{j_1} = \{j_1 \in S(j_2) : u(j_1) \geq a\}$ is convex $\forall a \in \mathbb{R}$.*

Lemma 4.5. *The correspondence $\varphi : S_1 \longrightarrow 2^{S_1}$, where $\varphi(S_1) = \mathbf{b}_1$ is convex, $\forall s \in S_1$.*

Proof. First, we prove utility u_m is quasiconcave.

Let σ_m^1, σ_m^2 be two best responses in bidder m's best response set $\mathbf{b}_m$, we have utilities

$$\begin{cases} u_m^1 = u_m(\sigma_m^1, \sigma_{-m}) \geq u_m(\tau_m, \sigma_{-m}), \forall \tau_m \in S_m \\ u_m^2 = u_m(\sigma_m^2, \sigma_{-m}) \geq u_m(\tau_m, \sigma_{-m}), \forall \tau_m \in S_m \end{cases}$$

Hence, $\lambda u_m^1 + (1-\lambda)u_m^2 \geq u_m(\tau_m, \sigma_{-m}), \lambda \in [0,1]$.

Given any $a \in \mathbb{R}$, if we create a upper level set p_a containing all $j_1 \in S(j_2)$ that meet the condition of having a utility $u_i \geq a$, and if p_a is always a convex set, then u_i is quasiconcave. This is apparent, as $u_1(j_1) = E_3 - E_4 = \mathbb{E}_{v_1,v_2}[(v_1 - j_2 v_2) \cdot 1_{j_1 v_1 \geq j_2 v_2}]$ is continuous and non-decreasing in j_1. If $j_1 v_1 \geq j_2 v_2$ and $j_1' v_1 \geq j_2 v_2$, we would always have $\lambda j_1 v_1 \geq \lambda j_2 v_2$ and $(1-\lambda) j_1' v_1 \geq (1-\lambda) j_2 v_2$ for any $\lambda \in [0,1]$. Adding both sides of the inequation respectively: $(\lambda j_1 + (1-\lambda) j_1') v_1 \geq j_2 v_2$, which means $\lambda j_1 + (1-\lambda) j_1'$ is also a member of p_a, or that any p_a is convex.

Since the utility function u_1 is defined on convex set S_1 and all of its upper level set is convex, the utility function is quasiconcave. Also, as u_i is quasiconcave, we have

$$u_m(\lambda\sigma_m^1 + (1-\lambda)\sigma_m^2, \sigma_{-m}) \geq \lambda u_m^1 + (1-\lambda)u_m^2 \geq u_m(\tau_m, \sigma_{-m}).$$

Therefore $\lambda\sigma_m^1 + (1-\lambda)\sigma_m^2$ is also a best response, it is in the $\mathbf{b}_m$ set. $\mathbf{b}_m$ is therefore convex valued. Finally, φ is convex if and only if each $\mathbf{b}_m$ is convex. Any combination of best responses will still be a best response. □

Definition 4.6 (Upper hemicontinuity [109]). *Correspondence $S : \Psi \longrightarrow \Xi$ is upper hemicontinuous, if for every $\psi_1 \in \Psi$ and $\epsilon > 0$, $\exists \delta > 0$ s.t.: if $\psi_2 \in \Psi$ and $||\psi_2 - \psi_1|| < \delta$, then $S(\psi_2) \subset B_\epsilon(S(\psi_1))$, where $B_\epsilon(x)$ denotes the ϵ-ball around x. Correspondence S is lower hemicontinuous, if for any open set $U \subset \Xi$ with $S(\psi_1) \cap U \neq \emptyset$, $\exists \epsilon > 0$, s.t. $\forall \psi_2 \in B_\epsilon(\psi_1)$, $S(\psi_2) \cap U \neq \emptyset$.*

Lemma 4.7. *let bidder 2's feasible strategies j_2 be in a set Ψ, let bidder 1's strategies $A = S_1(j_2), j_2 \in \Psi$ be in a set Ξ. The correspondence: $S_1 : \Psi \longrightarrow \Xi$ is continuous at all j_2.*

Proof. $\forall j_2 \in \Psi$, and a ϵ-ball around $S_1(j_2)$, we can find a range δ around j_2, s.t. any $j_2' \in \Psi, ||j_2' - j_2|| < \delta$, has $S_1(j_2')$ within the ϵ-ball around $S_1(j_2)$. This is apparent, since for any given best response parameter j_2' in the neighborhood of j_2, the corresponding strategy set in $S_1(j_2)$ would be a set of j_1' that is in the neighborhood of j_1 (upper hemicontinuous). It is proven in [46] that if the graph $G(S_1)$ is convex when $S_1(j_2)$ is monotone increasing, then S_1 is lower hemicontinuous. In this thesis, due to the linear form, and according to Lemma 4.3, S_1 is lower hemicontinuous. Therefore, S_1 is continuous [46]. □

Theorem 4.6 (Berge's maximum theorem [109])**.** *Let Ξ, Ψ be topological spaces, $u_1 : \Xi \times \Psi \to \mathbb{R}$ be a continuous function on the product space, and $S_1 : \Psi \to \Xi$ be a compact-valued correspondence s.t. $S_1(j_2) \neq \emptyset, \forall j_2 \in \Psi$.*
Define $u_1^*(j_2) = \sup\{u_1(j_1, j_2) : j_1 \in S_1(j_2)\}$, $\sup$ being the maximum operator of u, and the set of maximizers $S_1^* : \Psi \to \Xi$ by: $S_1^*(j_2) = \arg\sup\{u_1(j_1, j_2) : j_1 \in S_1(j_2)\} = \{j_1 \in S_1(j_2) : u_1(j_1, j_2) = u_1^*(j_2)\}$. If S_1 is continuous (i.e., both upper and lower) at j_2, then u_1^* is continuous and S_1^* is upper hemicontinuous with nonempty and compact values.

Lemma 4.8. *Correspondence $\varphi : S_1 \to 2^{S_1}$, where $\varphi(S_1) = S_1^* = \mathbf{b}_1$, is upper hemicontinuous with non-empty and compact values, and has a closed graph.*

Proof. According to Theorem 4.6, since S_1 is continuous (Lemma 4.7), non-empty and compact (Lemma 4.3), the correspondence φ is upper hemicontinuous with non-empty and compact values. It is apparent that best response set is a closed subset of the strategy set S on all $s \in S$. Therefore b_i is closed valued. A closed-valued upper hemicontinuous correspondence has a closed graph. □

Lemmas 4.3, 4.5 and 4.8 apply to the strategy sets of all players. According to the lemmas, the author can prove that this setup meets the conditions of Theorem 4.5, therefore the game has NE.

Pareto optimality

When bidders bid for service slots, the required resources are allocated. Theorem 4.2 proves the existence of Nash equilibria that maximizes welfare (total utility of bidders), without any guarantee of convergence. It also does not guarantee the optimality of the resource allocation, unless the following conditions are met: if bidders' valuation of the commodity is linear to its resource requirement, and all bidders have some access to resources (fairness).

Corollary 4.6.1. *In a second-price auction, where M bidders with utility as Eq. 4.2 compete in high contention, the outcome is an optimal resource allocation, if the bidders' valuation of commodities is linear to resource requirement and all bidders have a positive probability of winning.*

The setup of this thesis meets both conditions assumed by Corollary 4.6.1. 1) Valuation of the service request is a linear function of the resource needed: $v_1 = g_1\omega_1 + k_1, v_2 = g_2\omega_2 + k_2$, g, k are constants, ω is amount of resource required. The allocation rule under NE is: $A^*_{v_1,v_2} = 1$, if $j_1v_1 + d_1 \geq j_2v_2 + d_2$, otherwise 2. 2) In this setup, both bidders have at least some access to the resources, as a form of fairness. The author defines the fairness constraint to be: $\mathbb{E}[\omega_1|_{A_{v1,v2}=1}]/\mathbb{E}[\omega_2|_{A_{v1,v2}=2}] = \gamma \in \mathbb{R}_{>0}$.

Theorem 4.7. *The allocation $A^*_{v_1,v_2}$ maximizes overall resource allocation $\omega_1 + \omega_2$, subject to the fairness constraint, when the valuations are linear functions of resources. Or, the NE of the game achieves optimal resource allocation.*

Proof. Find the Lagrangian multiplier λ^* that satisfies the fairness constraint with NE allocation $A^*_{v_1,v_2}$. Define g, k as: $g_1 = (1 + \lambda^*)/j_1$, $k_1 = -d_1/j_1$, and $g_2 = (1 - \gamma\lambda^*)/j_2$, $k_2 = -d_2/j_2$. Then we can rewrite the allocation: $A^*_{\omega_1,\omega_2} = 1$, if $\omega_1(1+\lambda^*) \geq \omega_2(1 - \gamma\lambda^*)$, otherwise 2. The rest of the proof is the same as in [135]. □

4.5 Proposed solution

The author proposes DRACO, a **D**istributed **R**einforcement-learning algorithm with **A**uction mechanism for **C**omputation **O**ffloading, to solve the problem described in Sec. 4.3. The proposed algorithm for the dynamic environment is introduced in Sec. 4.5.1 and 4.5.2. Notations are in Table 4.3.

4.5.1 The fictitious self-play (FSP) algorithm

The term "fictitious play" comes from a learning algorithm in game theory: a player "imagines" how an opponent plays a stationary strategy, by learning from a historical distribution. The term "self-play" is a machine learning concept where the agent uses an old copy of itself (and the old policy) as the opponent. The fictitious self-play (FSP) method puts the two together: the agent learns a *behavioral strategy* that is a decision

Table 4.3: Short-term solution symbol definition

Sym.	Description	Sym.	Description
$k \in K$	service type/commodity	n_k	k's availability
$i \in I$	service request/bid	v	bid value
$m \in M$	vehicle/bidder	p	payment
z	bidding outcome	u	utility
α	backoff decision	b	bidding price
c	lost bid penalty	q	backoff cost
β	utilization	B	budget
$h \in H$	resource types	$\omega_{i,h}$	i's requirement of h
Q	service deadline	ρ	service request details
$\mathbf{e}_m$	m's env. variables	rl_m^t	m's present state for RL
sl_m^t	m's present state for SL	P_{-m}^t	other bidders' state at t
$\mathbf{a}$	action, $\mathbf{a} = (\alpha, b)$	S_m^t	complete state for RL
θ	actor parameters	$\mathbf{w}$	critic parameters

strategy purely based on its own past observations and actions regardless of other bidders' strategies. Thus, FSP addresses the convergence challenge of a best-response algorithm (**C3**). It balances exploration and exploitation by replaying its own past actions; then it cautiously plays the behavioral strategy mixed with best response [60].

The method consists of two parts: a supervised learning (SL) algorithm learns the bidder's own behavioral strategy ψ, and an RL algorithm learns its best response ζ to other bidders. The bidder has η probability of choosing best response action $\mathbf{a} = \zeta$ (with $\lim_{t\to\infty} \eta_t = 0$), otherwise it chooses behavioral strategy $\mathbf{a} = \psi$. The action includes backoff decision α and bidding price b. If $\alpha = 1$, the bidder submits the bid; otherwise, the bidder backs off.

FSP only converges in certain classes of games [81]. The application in this thesis belongs to a very general class of games that is multi-player, general-sum game with infinite strategies. In this thesis, FSP may not converge to an NE. However, empirical results show that by applying FSP, overall performance is greatly improved compared to using only RL. The FSP is described in Alg. 4.

Input to SL includes bidder m's service requests — service type, resource amount required and deadline: $\rho_m^t = \{(k_i, \omega_{i,h}, Q_i) | i \in I, h \in H\}$ (m can create multiple bids, each an independent request for service type k_i; ρ_m^t is the set of all of m's bids at t), and current environment information visible to m, denoted e_m^t (e.g., number of bidders in the network and system utilization β^t). Labels to the SL are bidder m's past actions. The training dataset contains all of bidder m's historical observations and actions. Specifi-

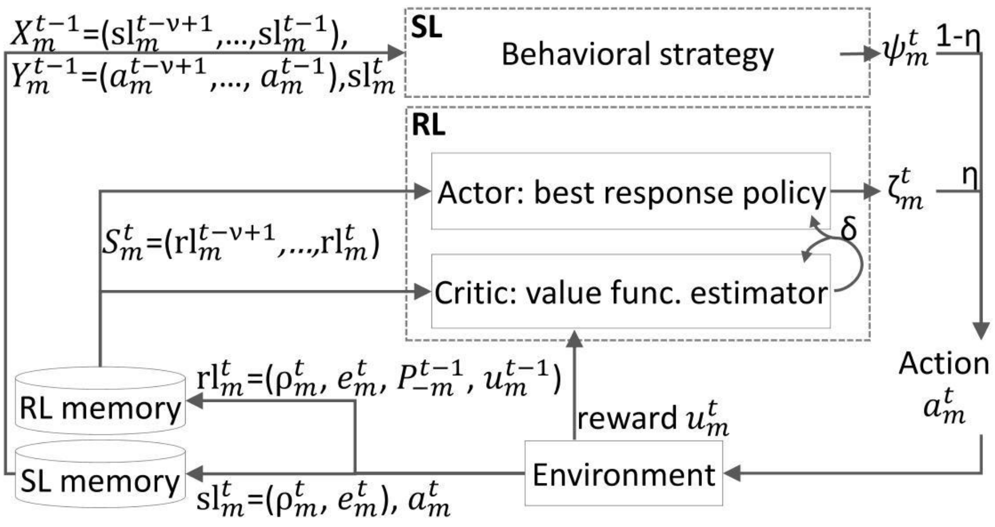

X_m^{t-1}=($\mathrm{sl}_m^{t-\nu+1}$,...,sl_m^{t-1}),
Y_m^{t-1}=($a_m^{t-\nu+1}$,..., a_m^{t-1}),sl_m^t
SL
Behavioral strategy
ψ_m^t
1-η
RL
Actor: best response policy
ζ_m^t
η
δ
S_m^t=($\mathrm{rl}_m^{t-\nu+1}$,...,rl_m^t)
Critic: value func. estimator
Action
a_m^t
rl_m^t=(ρ_m^t, e_m^t, P_{-m}^{t-1}, u_m^{t-1})
RL memory
reward u_m^t
SL memory
Environment
sl_m^t=(ρ_m^t, e_m^t), a_m^t

Figure 4.2: RL and SL algorithms

cally, the input $\mathrm{sl}_m^t = (\rho_m^t, e_m^t)$ and actual action $\mathbf{a}_m^t$ are stored in SL memory to train the regression model. Output of the regression model is the sequence of actions that comprises the behavorial strategy ψ_m^t. During learning, the SL updates its model parameters such that the predicted actions are as close to the actual actions as possible. The author uses a multilayer perceptron in the implementation of SL.

Input to RL is constructed from bidder m's present state rl_m^t. rl_m^t includes 1) ρ_m^t; 2) e_m^t; 3) previous other bidders' state P_{-m}^{t-1}, represented by the final price p_k, or $P_{-m}^t = \mathbf{p}^t = \{p_k^t | k \in K\}$, that is broadcasted by the auctioneer / ACA to all bidders at the end of each auction round; and 4) calculated utility u_m^{t-1} according to Eq. 4.2. To consider historical records, we take ν most recent states to form the complete state input to RL: $S_m^t = \{\mathrm{rl}_m^\tau | \tau = t - \nu + 1, \cdots, t\}$. RL outputs best response ζ_m (Fig. 4.2). The input consists of bidder's private information and easily obtainable public information, e.g., environment data and past prices, thus addressing **C2**.

4.5.2 The RL algorithm

Authors of [74] use VCG and a learning algorithm for the bidders to adjust their bidding price based on budget and observation of other bidders. The proposed approach in this thesis is similar in that it estimates other bidders' state P_{-m} from payment information and uses the estimate as basis for a policy.

The proposed approach differs from [74] in several major points. The author of this thesis uses a continuous space for bidder states (i.e., continuous value for payments). As

Algorithm 4 FSP

```
1: Initialize $\psi_m, \zeta_m$ arbitrarily, $t = 1, \eta = 1/t, \nu, P_{-m}^{t-1} = \mathbf{0}, u_m^{t-1} = 0$, observe $e_m^t$, create $\mathrm{rl}_m^t$, $\mathrm{sl}_m^t$
   and add to memory
2: while true do
3:     Take action $\mathbf{a}_m^t = (1-\eta)\psi_m^t + \eta\zeta_m^t$
4:     Receive $P_m^t$, calculate $u_m^t$, observe $\rho_m^{t+1}, \mathbf{e}_m^{t+1}$
5:     Create and add state to RL memory: $\mathrm{rl}_m^{t+1}$
6:     Create and add state to SL memory: $(\mathrm{sl}_m^{t+1}, \mathbf{a}_m^t)$
7:     Construct $S_m^t, S_m^{t+1}$, calculate $\zeta_m^{t+1} = \mathrm{RL}(S_m^t, S_m^{t+1}, u_m^t)$
8:     Calculate $\psi_m^{t+1} = \mathrm{SL}(\mathrm{sl}_m^{t+1})$
9:     $t \leftarrow t+1, \eta \leftarrow 1/t$
10: end while
```

also mentioned in [74], a finer-grained state space yields better learning results. Moreover, the author considers multiple commodity/service types, which is more realistic and therefore has a wider range of applications. Further, the proposed approach does not explicitly learn the transition probability of bidder states. Instead, this approach uses historical states as input and directly determines the bidder's next action.

The thesis uses the actor-critic algorithm [138] for RL (Alg. 5). The **critic** learns a state-value function $V(S)$. Parameters of the function are learned through a neural network that updates with $\mathbf{w} \leftarrow \mathbf{w} + \gamma^w \delta \nabla \hat{V}(S, \mathbf{w})$, where γ is the learning rate and δ is the temporal difference (TD) error. For a continuing task with no terminal state, the average reward is used to calculate δ [138]: $\delta = u - \bar{u} + \hat{V}(S', \mathbf{w}) - \hat{V}(S, \mathbf{w})$. In this thesis, the reward is utility u. The author uses exponential moving average (with rate λ) of past rewards as $\bar{u}$.

The **actor** learns the parameters of the policy π in a multidimensional and continuous action space. Correlated backoff and bidding price values are assumed to be normally distributed: $F(\mu, \Sigma) = \frac{1}{\sqrt{|\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu))$. For faster calculation, instead of covariance Σ, the algorithm estimates a lower triangular matrix L ($LL^T = \Sigma$). Specif-

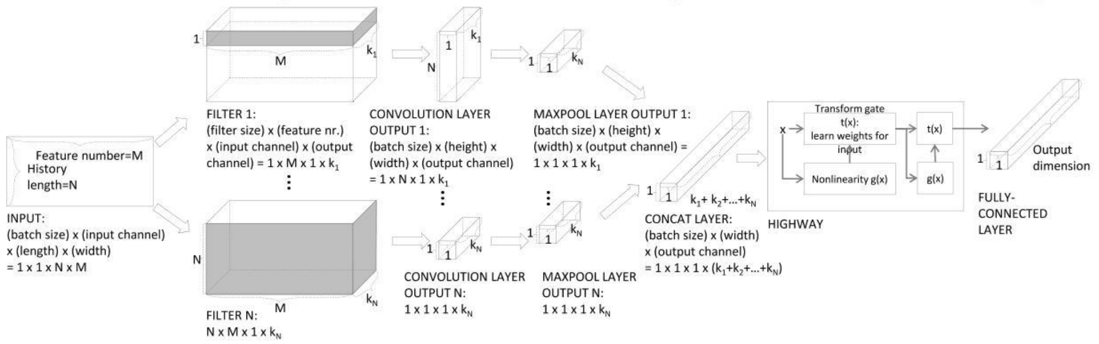

1
M
k1
FILTER 1:
(filter size) x (feature nr.) x (input channel) x (output channel) = 1 x M x 1 x k1
1
k1
N
CONVOLUTION LAYER OUTPUT 1:
(batch size) x (height) x (width) x (output channel) = 1 x N x 1 x k1
1
1
kN
MAXPOOL LAYER OUTPUT 1:
(batch size) x (height) x (width) x (output channel) = 1 x 1 x 1 x k1
Feature number=M
History length=N
INPUT:
(batch size) x (input channel) x (length) x (width) = 1 x 1 x N x M
N
M
kN
FILTER N:
N x M x 1 x kN
1
1
kN
CONVOLUTION LAYER OUTPUT N:
1 x 1 x 1 x kN
1
1
kN
MAXPOOL LAYER OUTPUT N:
1 x 1 x 1 x kN
1
1
k1+ k2+...+kN
CONCAT LAYER:
(batch size) x (width) x (output channel) = 1 x 1 x 1 x (k1+k2+...+kN)
x
Transform gate t(x): learn weights for input
Nonlinearity g(x)
t(x)
g(x)
HIGHWAY
1
1
Output dimension
FULLY-CONNECTED LAYER

Figure 4.3: Stacked CNN with highway structure

Algorithm 5 RL

```
1: Initialize $\theta, w$ arbitrarily. Initialize $\lambda$
2: while true do
3:     Input $t$ and $S_m^t, S_m^{t+1}$ constructed from RL memory
4:     Run critic and get $\hat{V}(S_m^t, \mathbf{w}), \hat{V}(S_m^{t+1}, \mathbf{w})$
5:     Calculate $\bar{u}_m = \lambda \bar{u}_m$ and $\delta$ (utility $u$ is reward $R$)
6:     Run actor and get $\mu(\theta), \Sigma(\theta)$
7:     Sample $\zeta_m^{t+1}$ from $F(\mu, \Sigma)$, update $\mathbf{w}$ and $\theta$
8: end while
```

ically, the actor model outputs the mean vector μ and the elements of L. Actor's final output ζ is sampled from F through: $\zeta = \mu + L\mathbf{y}$, where $\mathbf{y}$ is an independent random variable from standard normal distribution. Update function is $\theta \leftarrow \theta + \gamma^\theta \delta \nabla \ln \pi(\mathbf{a}|S, \theta)$. The algorithm uses $\frac{\partial \ln F}{\partial \mu} = \Sigma(\mathbf{x} - \mu)$ and $\frac{\partial \ln F}{\partial \Sigma} = \frac{1}{2}(\Sigma(\mathbf{x} - \mu)(\mathbf{x} - \mu)^T \Sigma - \Sigma)$ for backpropagation. This is a common approach for continuous action space.

The objective is to find a strategy that, given input S_m^t, determines $\mathbf{a}$ to maximize $\frac{1}{T-t}\mathbb{E}[\sum_{t'=t}^{T} u_m^{t'}]$. To implement the actor-critic RL, the proposed algorithm uses a stacked convolutional neural network (CNN) with highway [134] structure similar to the discriminator in [160] for both actor and critic models. Each CNN in the stack convolves on the dimensions of sequence length (i.e., the number of historical data in the time-series) and feature vector. These CNN layers have diverse kernel sizes to cover different lengths of history and extract features, and it is easily parallelizable, compared to other sequential networks. Since state information is temporally correlated, such a sequential network extracts features better than multilayer perceptrons. The highway structure directs information flow by learning the weights of direct input and performing non-linear transform of the input. An illustration of the structure is in Fig. 4.3.

In low contention, authors of [115] prove that an actor-critic RL algorithm converges to NE in a potential game. In high contention, although the author of this thesis proves the existence of an NE in the stationary case (Sec. 4.4.3), the convergence property of the proposed algorithm in a stochastic game is not explicitly analyzed. The author will show it through empirical results in Sec. 4.6.

4.6 Evaluation

We test DRACO with the V2X system model as introduced in Sec. 3.2. The training and test environments are as described in Sec. 3.3, including the operating-side load-balancing allocation solution RIAL. The performance metrics are as introduced in Sec. 3.4.

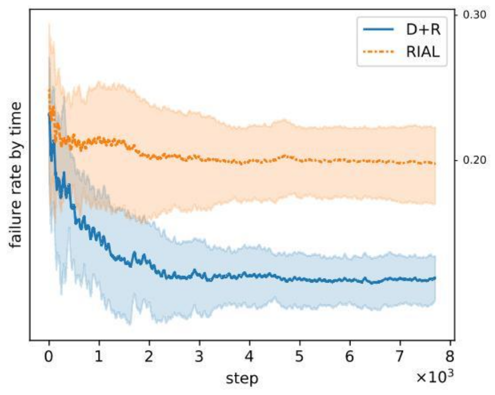
The test is in two steps. First, we comprehensively study the performance of active agents in a synthetic setup with randomized inputs and a wide range of environment parameters. Then, we test it in the realistic setup as described in Sec. 3.3.1.

4.6.1 Synthetic setup results

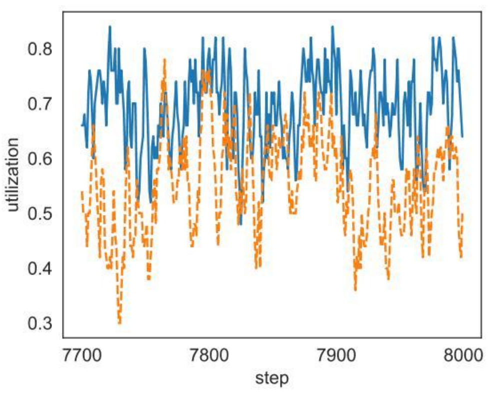
This synthetic setup is specific to the evaluation of the short-term, single-objective algorithm DRACO. This setup covers a wide range of hypothetical scenarios by varying parameters such as system capacity, service/task types and number of rebidding:

- Task types by resource needs in time-resource units: task type F1: 3 units, and task type F2: 30 units.
- Service types by deadline and probability: F1, 300ms: 18.75%; F1, 50ms: 18.75%; F2, 300ms: 6.25%; F2, 50ms: 6.25%; F1-F2, 300ms: 18.75%; F1-F2, 50ms: 18.75%; F2-F1, 300ms: 6.25%; F2-F1, 50ms: 18.75%.
- Service arrival rate per vehicle: requests for services are generated according to a two-state Markov modulated Poisson process (2-MMPP) [148] with transition probabilities $p_{\text{high}} = p_{\text{low}} = 0.6$. The discrete-time MMPP in this study is interchangeable with classic continuous-time MMPPs when modeling arrival processes, as the sampling interval of one time step is small enough [106].
- Capacity: 50–230 resource units.
- Maximum permitted rebidding: 1 or 5 times, respectively.
- Vehicle count: constant at 30.
- Vehicle arrival rate: 0, always in the system; speed: 0.
- Data size: uniformly at random between 2.4-9.6 kbit.
- Uplink and downlink latency: 0 s.

As described in Sec. 3.3, the author sets up environments with homogeneous agents (i.e., agents with the same types of algorithms): 1) benchmark: bidders are passive agents with no learning capability, and the auctioneer uses the load-balancing allocation solution RIAL. The algorithm is therefore denoted by “RIAL”; 2) the proposed solution: bidders are active agents with DRACO algorithm, and the auctioneer uses RIAL. The algorithm is DRACO+RIAL, denoted by D+R. Fig. 4.4a shows a training example



(a) DRACO reduces the overall offloading failure rate.



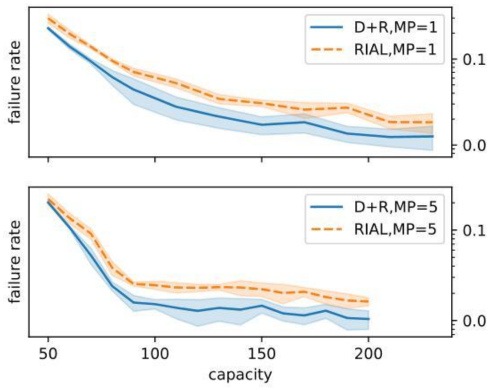
(b) DRACO learns to better utilize resource in remote computing site. To show the details, only the last 300 steps in the simulation are illustrated. More analysis of utilization is in Fig. 4.5d.

Figure 4.4: OFR and resource utilization, capacity=60, MP=1

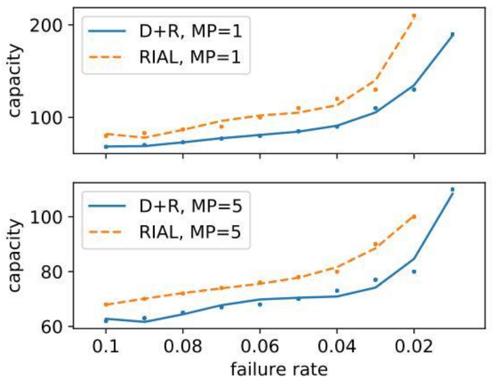
where D+R's OFR is 14% compared to RIAL's 20% at the end of training, or a reduction of 30%. The lines are the mean OFR of several simulation runs, and the shaded area marks the standard deviation. Fig. 4.4b shows where the learning is most useful. We depict the remote site's resource utilization. Since the ACA unit's information of site utilization is delayed, with only RIAL, the site is either over utilized or starved, in distinctive cycles (dotted line). When the vehicles learn with DRACO, they achieve better utilization (solid line).

Overall OFR in all parameter settings is shown in Fig. 4.5a. Evaluation data is collected from additional evaluation runs after the models are trained, with random incoming service requests newly generated by the MMPP. Besides requests that are not admitted by the ACA unit, the failure rate also includes requests that are admitted, but cannot be executed by the operating side before deadline (reliability). We observe that with D+R, reliability is 99% and consistently higher than with RIAL for all results in the paper. We also observe that DRACO significantly reduces OFR (on average 40% reduction), especially when MP is low. In low contention, i.e., capacity$\geq$ 100, D+R achieves the same level of OFR with much less resource (Fig. 4.5b). The improvement becomes more significant as OFR decreases. In particular, D+R reaches 1% OFR with much less resource compared to RIAL regardless of MP.

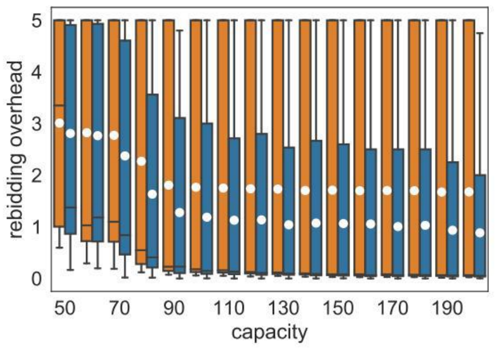
Further, higher MP reduces D+R's advantage over RIAL. This result is to be expected: when more rebidding is permitted, low OFR can be achieved by trial and error, limiting the advantage of DRACO's backoff strategy. However, trial-and-error comes at a cost: Fig. 4.5c compares the rebidding overhead used by both algorithms when MP=5. In



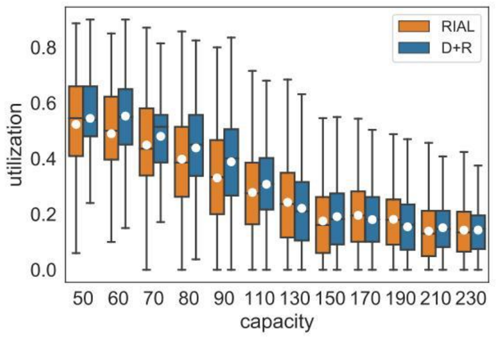
(a) D+R reduces OFR by 40%, achieves 1% OFR in low contention; RIAL OFR only reaches 2%.



(b) D+R needs less resource for same OFR (e.g., 2% failure and MP=1, 38% less resource needed).



(c) Rebidding overhead vs capacity, MP=5. Overhead reduces by 32% on average.



(d) Resource utilization, MP=1. DRACO better utilizes resource by 18% in high contention.

Figure 4.5: (a): OFR vs capacity, (b): required capacity to reach OFR$\leq$ 10%, (c): rebidding overhead, (d): utilization by capacity

high contention, both active and passive agents leverage rebidding, and the difference in rebidding overhead is small. D+R's advantage becomes more significant as capacity increases. The box plot shows the median (line), mean (dot), 1st to 3rd quartiles (box) and data range (whiskers). D+R imposes on average 32% lower rebidding overhead.

To validate the findings in Fig. 4.4b, we can compare resource utilization under different capacities. Fig. 4.5d shows the remote site's utilization when MP=1. In high contention, the increase in utilization is up to 18% — when capacity is limited, D+R achieves lower OFR through more efficient resource usage. In low contention, capacity is less critical; active and passive agents result in similar utilization. Regardless of capacity level, D+R reduces the standard deviation in utilization by up to 21%.

Fig. 4.6 shows the cumulative probability of vehicles' individual OFRs. With DRACO,

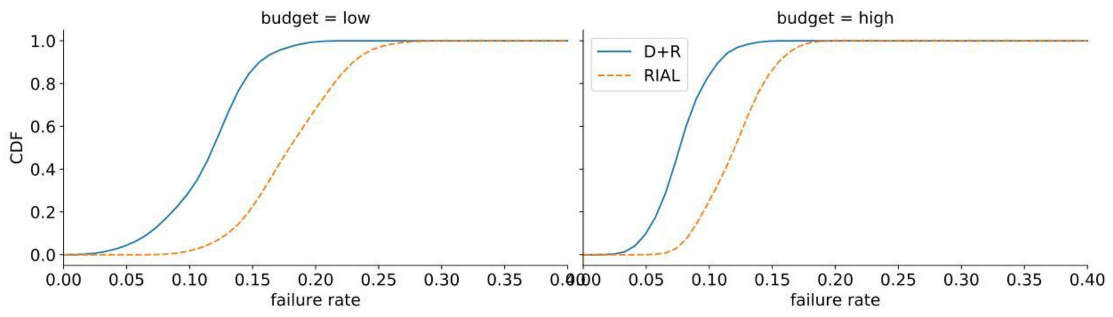
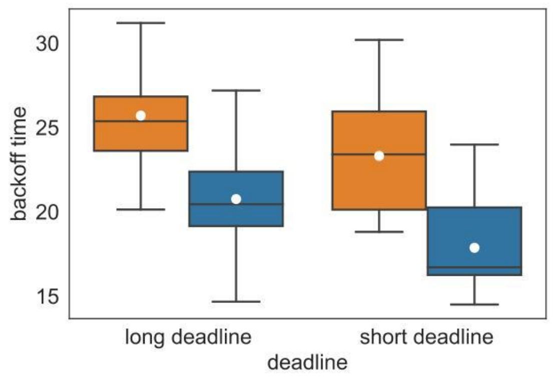
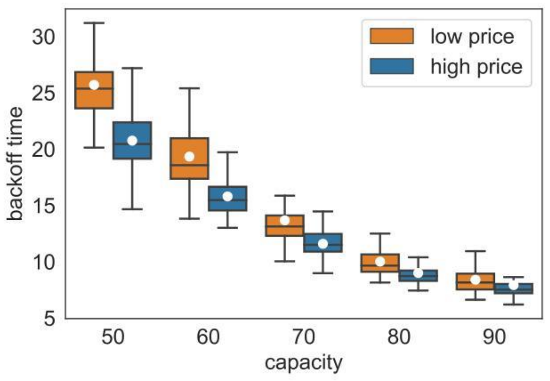


Figure 4.6: CDF of individual OFRs (capacity=70, MP=1): DRACO reduces individual OFRs.



(a) Capacity=50, MP=5, backoff vs. price tradeoff: for all deadlines, vehicles that bid low (high) use long (short) backoff.



(b) MP=5, long deadline, high contention: backoff time decreases with higher capacity, but the tradeoff with price remains.

Figure 4.7: Backoff and price tradeoff

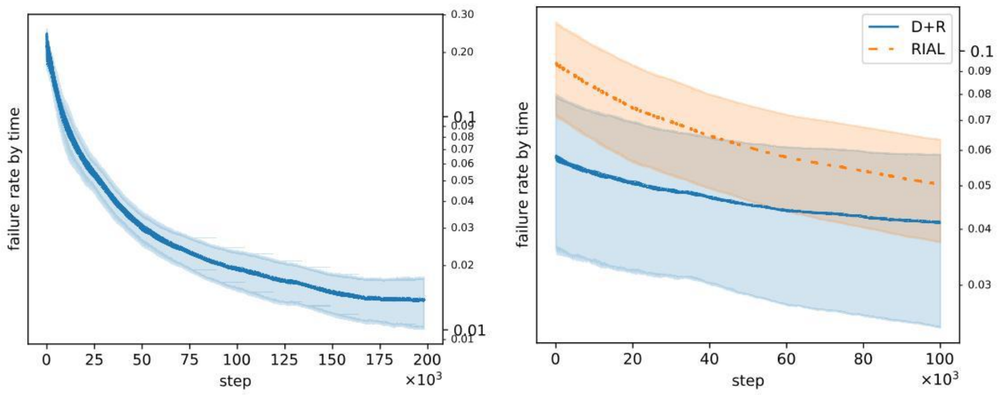
as system overall OFR reduces, the individual OFRs reduce accordingly: the auction does not cause disadvantage to individual vehicles. Moreover, vehicles with lower budget improve by a greater margin: they learn to utilize backoff mechanism to overcome their disadvantage in initial parameterization. This also implies an improvement in system overall fairness. In the following chapters, Sec. 5.5.1 will analyze the algorithm's performance with fairness score as the single objective, and Sec. 6.6 will analyze the algorithm's performance with fairness as one of the multiple objectives. Fig. 4.7a shows how vehicles learn to trade off between bidding price and backoff time. They are separated into two groups: a vehicle is in the "low price" group if it bids on average lower than the average bidding price of all vehicles; otherwise, it is in the "high price" group (here we analyze actual bidding prices instead of the predefined budgets). When service requests have a longer deadline, vehicles in both price groups learn to utilize longer backoff. Regardless of the service request deadline, "low price" vehicles always use longer backoff in their bidding decisions, compared to the “high price” group. Fig. 4.7b demonstrates the tradeoff effect with increasing capacity. As capacity increases, backoff time decreases, but the tradeoff is present in all cases.

To summarize: Fig. 4.4 and 4.5 demonstrate DRACO’s excellent overall system performance. More importantly, Fig. 4.6 shows that proper incentivization aligns the system objective with individual objectives (**C1**), and Fig. 4.7 demonstrates where the proposed approach fundamentally differs from previous approaches: differently initialized agents learn to select the most advantageous strategy based on limited feedback signal (**C2**). The capability to learn and behave accordingly makes the agents highly flexible in a dynamic environment. Finally, Fig. 4.4a shows good convergence speed despite computation and communication complexity of the problem (**C3**).

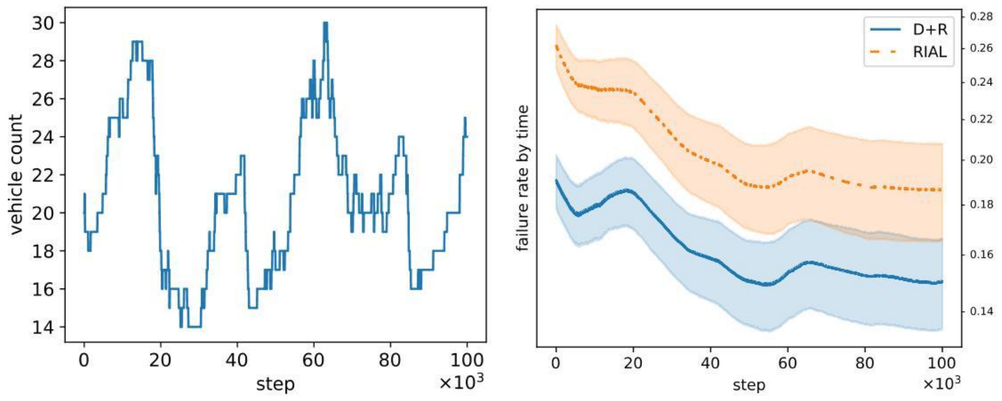
4.6.2 Realistic setup results

In the realistic setup as described in Sec. 3.3.1, we train our active agents with DRACO in low contention. Fig. 4.8a-left shows convergence to OFR of 2%. Then, we evaluate the trained models in the same environment with newly simulated trace data from SUMO. In this setup, the proposed approach still reaches OFR of 4% and outperforms RIAL (Fig. 4.8a-right). All simulations are repeated seven times to take randomness into account.

Finally, we test the trained models in a significantly different environment, changing traffic light phases, vehicle arrival rate and speed to make the environment more volatile and dynamic, and reduces capacity to create a high-contention situation. The resulting vehicle count over time (Fig. 4.8b-left) shows more frequent fluctuation with bigger amplitude compared to the training environment. Note that vehicle count and OFR do not vary synchronously — OFR is determined by vehicle count and numerous other complicating factors such as transmission, queueing and processing time, past utilization, etc. Despite the significant changes to the environment, D+R still outperforms RIAL, reaching low OFRs in high contention without requiring any further training (Fig. 4.8b-right). It shows that DRACO generalizes very well — in fact, in the more volatile and dynamic environment, the superiority of active agents becomes more obvious.



(a) Training env.: low contention with abundant resource, traffic phase=10-40s, low vehicle speed(10 km/h), low arrival rate=(1 every 2.2s), low variation in vehicle count(22-30): OFR in training(left) and evaluation(right).



(b) Test env.: high contention with limited resource, traffic phase=20s, high vehicle speed(30 km/h), high arrival rate(1 every 1s), high variation in vehicle count(14-30): vehicle count over time(left) and OFR(right).

Figure 4.8: Offloading failure rate (OFR) in training and test environments

4.7 Conclusion of the chapter

Compared to only having a centralized load-balancing solution at the MEC, DRACO succeeds in incentivizing each autonomous vehicle to add to the load-balancing effect in a distributed manner, which significantly increases resource utilization in high contention and reduces capacity needed to reach the same service level. More specifically, DRACO lets each vehicle independently decide how to trade off between backoff time and bidding price, at the same time overcoming their initial disadvantages such as a lower budget.

Through incentivization, the interaction mechanism in this thesis aligns private and system goals without sacrificing either user autonomy or system-wide resource efficiency, despite the distributed design with limited information sharing.

Results in the realistic setup show DRACO's excellent generalization property in different realistic environments, making it a potential add-on to any existing centralized solution at the MEC.

One of the shortcomings of this setup is the disregard of the long-term effect of decisions — e.g. if unused budget could be saved for the future, current decision would impact future states. This is essentially a long-term, sparse reward signal with unknown length of delay. In the next chapter, the challenge of such long-term reward signals will be addressed.

Chapter 5

Learning with Sparse and Delayed Reward

5.1 Motivation

The previous chapters introduced single-agent and multi-agent RL algorithms with short-term rewards. RL algorithms are well known for their ability to learn sequential tasks and balance between exploitation and exploration [10, 143]. However, they are typically "short-term" algorithms: in 1961, Minsky in [98] mentioned the necessity and difficulty of long-term temporal credit assignment in RL — it is essential to associate long-term reward to specific behavior or series of behaviors (e.g. strategic behaviors) such that behaviors that contribute to the long-term reward are prioritized. In RL algorithms with no focus on temporal credit assignment, importance of the immediate reward heavily outweighs estimated reward in the distant future, and the estimation has a bias that is related to the length of delay and exponential in the number of possible states [12]. Worse still, if the reward is both delayed and sparse, the reward estimation often has a high variance due to lack of predictable future states, especially with a big state-action space and high variance in the value of next states [96, 129]. When decisions have long-term effects, such "short-term" algorithms would lead to worse performance. It proves to be one of the biggest challenges of applying RL in the real world [44].

One common approach in long-term RL is to extract features from historical records, thus linking the delayed reward to behaviors in the past [62]. Learning with such algorithms is inefficient since learning from past experiences can only happen when the delayed outcomes become available. To address the delay, reference [91] factorizes one state into an intermediate and a final state with independent transition probabilities and predicts each state at different intervals. Reference [65] describes a credit-assignment method that focuses on the most relevant memory records via content-based attention; the algorithm is capable of locating past memory to execute new tasks and generalizes very well. These approaches focus more on the delay in reward signal and less on sparsity. In this thesis, the long-term reward is delayed, sparse and sporadic, making these approaches inapplicable.

To address sparsity of rewards, many model-based methods add intrinsic, intermediate rewards between sparse extrinsic reward signals. Such methods often adopt a supervised learning algorithm to predict next states and use the difference between the predicted and target state-action pair values as intrinsic reward. Due to lack of extrinsic reward signals, these predictions usually have low accuracy, and the inaccuracy is propagated into the future. But such methods significantly improve data efficiency, and as a result, the algorithms learn much faster. For example, reference [62] separately trains many "feature models" to predict each feature of the next state as well as a "reward model" to predict reward. Between sparse extrinsic rewards, the algorithm samples estimated next state and reward from the models. The models are only updated when there is new input available. Their approach assumes that state features are independent and can be learned separately, and the accuracy of the reward model is still related to the sparsity of the reward signal. Reference [31] uses a long-short-term memory (LSTM) to extract features from past memory that are more relevant to the current task, thus improving the model's generalization properties. The algorithm also uses two independent models to predict next state and action. The prediction loss becomes intermediate, intrinsic rewards inserted between sparse extrinsic rewards, and where prediction loss is high, a higher reward encourages exploration. In this approach, the intrinsic reward signal is not related to the extrinsic sparse reward, and the final outcome of the game is not credited to specific agent behaviors. The lack of temporal credit assignment on a long time horizon affects learning efficiency [98], especially with sparse rewards and conflict between the agent's short-term and long-term goals [51, 73], as is often the case in real life, e.g., maximizing short-term gains can hurt long-term strategic goals.

Credit assignment is a method that credits actions over a long time period according to their contribution to a delayed outcome. It is often used in game setups, in which the outcome and reward of the game only becomes available after a long sequence of actions by the player. There are different approaches to credit assignment, for example, the credit assignment in [73] does not directly credit behaviors but credits a population of models. It requires each model to play a full episode in each step to generate experience before the model can be credited based on the final outcome, it is therefore not applicable in the setup of this thesis: a dynamic multi-agent environment with continuing task and no clear episodes. Reference [51] uses an attentional network to assign weights to past behaviors through reward shaping. Their credit assignment algorithm decomposes the long-term reward to densify reward signals. In this thesis, decomposed long-term rewards cannot be added directly to short-term rewards, because the different types of reward can be conflicting, and a scalarized reward loses the information on the contribution of each reward type.

In this chapter, the author expands the short-term, single-objective, stand-alone MARL algorithm in Sec. 4.5 with 1) a feature extraction submodule for generalization, 2) an attention layer for long-term credit assignment and 3) a curiosity module for sparse reward signals. Sec. 5.5 shows that these modules' effect on the agent's performance can be compounded, and the performance is best with all three modules.

In the synthetic setup with two common auction types (Sec. 5.5.1), we can observe that the agents with the proposed algorithm behave more aggressively in the competition against other agents and perform better in the long-term goal to maximize cumulated private payoff; however, the selfish behavior has a negative impact on the overall fairness. To improve system overall fairness, the author also uses fairness score as the long-term goal in a second simulation. Simulation results in the synthetic setup show the improvement in individual payoff and in overall fairness index score; we can also observe an increase in social welfare. In the realistic setup (Sec. 5.5.2), empirical results show that over time, the best-response strategies stabilize and lead to significantly improved individual and overall outcomes. The designed interaction mechanism aligns private and system goals without sacrificing either user autonomy or system-wide resource efficiency, despite the distributed design with limited information sharing. Finally, the algorithm demonstrates capability to generalize to very different, previously unseen environments without the need for retraining. Contributions of this chapter are:

- The author introduces MALFOY, a distributed algorithm that learns to utilize long-term, sparse reward signals with varying delay; it optimizes decision strategy over a long time period.
- The author shows using extensive simulation that agents with MALFOY outperform benchmark agents on overall resource utilization, offloading failure rate, load variation and communication overhead. They also generalize well.
- The author open-sources the code [2] to encourage reproduction and extension of the work.

5.2 Chapter outline

Table 5.1 compares the differences in algorithm type and simulation environment setup between chapters 4 and 5. As the author introduces more algorithms, the table will contain more information for comparison between the algorithms and setups.

Sec. 5.3 reformulates the original problem from Sec. 4.3 into a long-term single-objective optimization problem. Sec. 5.4 proposes the second algorithm of this thesis to solve the problem. Sec. 5.5 evaluates the algorithm in both a synthetic and a realistic setup as described in Sec. 3.3. Sec. 5.6 concludes the chapter.

Table 5.1: Chapter5 outline

	Chapter4: DRACO	Chapter5: MALFOY
Alg. type	short term	long term
Obj. type	single objective	single objective
Synthetic setup	Simulated second-price forward auction in V2X application scenario, with varying resource capacity, service request types, number of rebidding (Sec. 4.6.1). Purpose: analyze algorithm performance with randomized inputs and a wide range of environment parameters.	Simulated two simplified common repeated auctions with no specific application scenario: first-price reversed and second-price forward auction, with fixed resource capacity, one commodity type, no rebidding (Sec. 5.5.1). Purpose: analyze contribution of each module and different system reward signals.
Realistic setup	Same as in Sec.3.3.1. Purpose: analyze and compare performance in simulated traffic.	

5.3 Problem formulation

The author reformulates the single-objective, short-term problem from the previous chapter into a single-objective, long-term reward maximization problem. The biggest difference is the cumulated budget B^t; it carries the effect of current decision into the future. Notation is in Table 5.2.

Table 5.2: Long-term problem symbol definition

Sym	Description	Sym	Description
$k \in K$	service type/commodity	n_k	k's availability
$m \in M$	vehicle/bidder	$i \in I$	service request/bid
B	wealth/budget	v	bid value
β	utilization	Q	service deadline
α	backoff decision	b	bidding price
c	cost to join the auction	q	backoff cost
p	payment	z	bidding outcome
u	immediate utility	U	cumulated utility

Let M be the set of vehicles (bidders) and K the set of commodities (service types), each type with, at time t, a total of n_k^t available service slots in computing sites. Bidder m has an initial wealth of B_m^0. It has at most 1 demand for each service type $k \in K$ at time t, denoted by a bid $i_k^t \in I$.

From its bidding strategy π_m, bidder m draws its actions $\alpha_m^t = \{\alpha_{m,k}^t \in \{0,1\}, \forall k \in K\}$ and $\mathbf{b}_m^t = \{b_{m,k}^t \in \mathbb{R}_+, \forall k \in K\}$ for each service type. α is the vector of backoff decision, $\mathbf{b}$ is the vector of bidding price. More specifically, bidder m's options for each bid i_k^t are: 1) back off ($\alpha_{m,k}^t = 0$) with a backoff cost $q_{m,k}^t$, or 2) bid ($\alpha_{m,k}^t = 1$) with price $b_{m,k}^t$. To avoid overbidding, at any time t, $\sum_k \alpha_{m,k}^t b_{m,k}^t \leq B_m^t$.

From bidder m's perspective, the competing bidders (denoted $-m$) draw their actions from a joint strategy distribution π_{-m}^t that is an unknown function of $(\mathbf{p}^1, \cdots, \mathbf{p}^{t-1})$, where $\mathbf{p}^t \in \mathbb{R}_+^{|K|}$ is the vector of final prices at the end of time t. All bidders get the new $\mathbf{p}^t$ as feedback. If bidder m wins its bid i_k^t indicated by bidding outcome $z_{m,k}^t = 1$, it pays p_k^t to the auctioneer. If it loses ($z_{m,k}^t = 0$), it pays 0 to the auctioneer but has a cost associated with losing the bid, denoted by $c_{m,k}^t$.

The auction repeats for T periods, in every auction round, bidder m's utility (Sec. 2.2.3) is $u_m^t(\alpha_m^t, \mathbf{b}_m^t, \mathbf{p}^t, \mathbf{z}_m^t, \mathbf{c}_m^t, \mathbf{q}_m^t)$, the utility is added to the wealth pool: $B_m^{t+1} = B_m^t + u_m^t$. If $B_m^{t+1} \leq 0$, bidder m loses all the unfinished bids and is reset. Its goal is to maximize the long-term utility:

$$\mathcal{U} = \frac{1}{T} \sum_{t=1}^{T} u_m^t, T \rightarrow \infty.$$

The bidding budget B_m^t is accumulated over time, such that previous decisions on back-off and bidding prices also impact future bidding decisions. In every auction round, the budget reduces through payment and costs, and increases by the commodity's valuation,

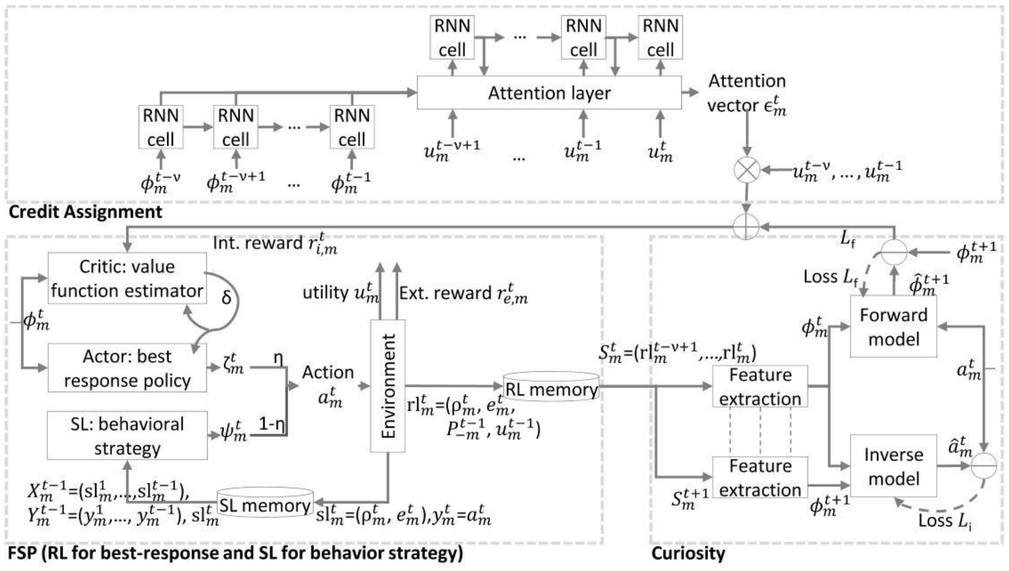

RNN cell
Attention layer
Attention vector ϵ_m^t
$\phi_m^{t-\nu}$ $\phi_m^{t-\nu+1}$ … ϕ_m^{t-1}
$u_m^{t-\nu+1}$ … u_m^{t-1} u_m^t
$u_m^{t-\nu}, \ldots, u_m^{t-1}$
Credit Assignment
Int. reward $r_{i,m}^t$
Critic: value function estimator
δ
ϕ_m^t
Actor: best response policy
ζ_m^t
η
1-η
Action a_m^t
SL: behavioral strategy
ψ_m^t
utility u_m^t
Ext. reward $r_{e,m}^t$
Environment
$\text{rl}_m^t=(\rho_m^t, e_m^t, P_{-m}^{t-1}, u_m^{t-1})$
RL memory
SL memory
$X_m^{t-1}=(\text{sl}_m^1,\ldots,\text{sl}_m^{t-1})$, $Y_m^{t-1}=(y_m^1,\ldots,y_m^{t-1})$, sl_m^t
$\text{sl}_m^t=(\rho_m^t, e_m^t), y_m^t=a_m^t$
FSP (RL for best-response and SL for behavior strategy)
$S_m^t=(\text{rl}_m^{t-\nu+1},\ldots,\text{rl}_m^t)$
S_m^{t+1}
Feature extraction
Forward model
Inverse model
L_f
Loss L_f
ϕ_m^{t+1}
$\hat{\phi}_m^{t+1}$
a_m^t
$\hat{a}_m^t$
Loss L_i
Curiosity

Figure 5.1: Long-term algorithm MALFOY

if the bidder wins the commodity. This is different to the problem definition in Sec. 4.3, where budget is reset to the initial value after every auction round.

5.4 Proposed solution

To solve the long-term reward maximization described in Sec. 5.3, the author proposes MALFOY: **M**ulti-**A**gent reinforcement **L**earning **FO**r sparse and dela**Y**ed reward — an extended version of Sec. 4.5. With this extension, the algorithm is generalized to target a wider range of problems, and the problem tackled in Sec. 4.5 becomes a special case where the long-term reward signals have an interval of 1 (i.e. available at the end of every auction round).

MALFOY's utility function is similar to the original definition in Sec. 4.4.1, Eq. 4.2. We repeat Eq. 4.2 here (the notation t is omitted for simplicity):

$$u_{(m,\text{shortterm})} = W_1 \sum_{k\in K} u_{m,k} + W_2 \cdot (1-\beta) \tag{5.1}$$

where m is the bidder index, β is the system resource utilization, $\mathbf{W}$ are preference

Table 5.3: Long-term solution symbol definition

Sym	Description	Sym	Description	Sym	Description
ζ	best response	ψ	behavioral strategy	$\mathbf{e}_m$	env. variables
ρ_m	private bidder info	$\mathbf{a}$	action, $\mathbf{a} = (\alpha, b)$	P^t_{-m}	other bidders state
sl^t_m	SL present state	rl^t_m	RL present state	S^t_m	RL complete state
λ	$\bar{u}$'s weight factor	θ	actor parameters	$\mathbf{w}$	critic parameters
γ	learning rate	δ	TD error	η	ζ's weight
ν	history length	μ	action mean	Σ	action covariance
ϕ	featurized state	ϵ	credit assign. weight	$r_{i,m}$	intrinsic reward
$r_{e,m}$	extrinsic reward	L_f	forward mdl loss	L_i	inverse mdl loss
ξ	reward weight				

weights given as hyperparameters to scalarize the objectives. For long-term reward, we add an additional reward signal $r_{m,\text{longterm}}$ to it, that is only available at the end of every interval. The agent's preference of the objective is expressed by a constant weight W_3:

$$u_m = u_{(m,\text{shortterm})} + W_3 \cdot r_{m,\text{longterm}} \tag{5.2}$$

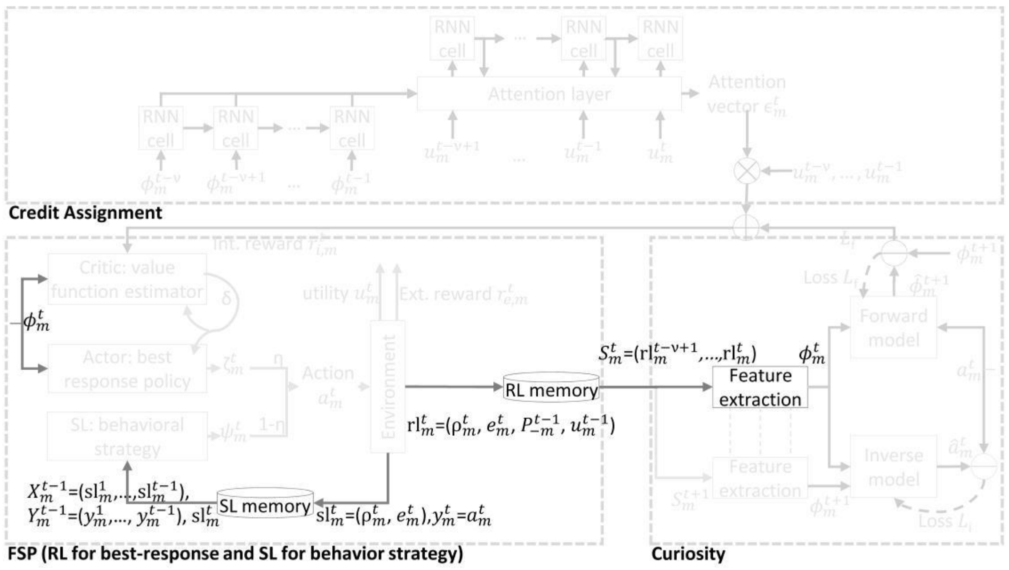
This chapter studies the scalarized single-objective algorithm. Sec. 6.5 will extend the algorithm to learn multiple objectives with a custom and changing preference vector $\mathbf{W}$.

In the synthetic setup of Sec. 5.5.1, the long-term reward signal $r_{m,\text{longterm}}$ is either the cumulated individual payoff or a (normalized) system fairness score. In the realistic setup of Sec. 5.5.2, the longterm reward signal is the normalized cumulated individual payoff. In all training and evaluation of this chapter, W_1 is fixed at 1, W_2 at 0.1 and W_3 at 0.5.

In algorithm design, alongside the original FSP (with the RL and SL modules), we add a curiosity module and a long-term credit assignment module (Fig. 5.1). The following section explains the details.

5.4.1 Feature extraction

In general, input to and output from the FSP's SL and RL modules are the same as in Sec. 4.5, only instead of the original state vector, the input to RL is now a featurized state vector from a feature extraction submodule. The feature extraction submodule is part of a curiosity module (Sec. 5.4.2); it extracts features that are most relevant to the agent's actions and disregards influence from the environment (Fig. 5.2). By doing so, the model's generalization property improves significantly in new environments. This is also shown in the empirical results in Sec. 5.5.2. We reiterate the inputs to RL and SL below.



Credit Assignment
RNN cell
Attention layer
Attention vector e^t_m
Int. reward $r^t_{i,m}$
Critic: value function estimator
Actor: best response policy
SL: behavioral strategy
utility u^t_m
Ext. reward $r^t_{e,m}$
Action a^t_m
Environment
RL memory
S^t_m=(rl$^{t-\nu+1}_m$,...,rl^t_m)
rl^t_m=(ρ^t_m, e^t_m, P^{t-1}_{-m}, u^{t-1}_m)
Feature extraction
Forward model
Inverse model
Loss L_f
Loss L_i
X^{t-1}_m=(sl^1_m,...,sl$^{t-1}_m$),
Y^{t-1}_m=(y^1_m,..., y^{t-1}_m), sl^t_m
SL memory
sl^t_m=(ρ^t_m, e^t_m),y^t_m=a^t_m
FSP (RL for best-response and SL for behavior strategy)
Curiosity

Figure 5.2: Feature extraction and replay memory

Input to RL: present state rl^t_m includes 1) private bidder information ρ^t_m, including service type in the bid, resource amount required, deadline, initial and current bidding budget, etc.; 2) m's observation of the environment e^t_m, including number of bidders in the vicinity (provided by either vehicle sensors or the RSU) and system utilization (as feedback from the RSU); 3) previous other bidders' state P^{t-1}_{-m}, represented by the final prices, or $P^t_{-m} = \mathbf{p}^t = \{p^t_k | k \in K\}$.

Next state depends not only on the current state, but on a number of historical states. We take ν most recent states to form the complete state vector: $S^t_m = \{\mathrm{rl}^\tau_m | \tau = t - \nu + 1, \cdots, t\}$. Thus, input data consists mostly of information private to the user m as well as environment data and past prices, which are easily obtainable public information. Each state S^t_m corresponds to a reward u^t_m, both are saved in the RL replay memory. The input to RL, ϕ^t_m, is a featurized state vector from the original S^t_m. RL outputs best response ζ_m.

Input to SL: compared to RL, inputs to SL do not include previous other bidders' states (past final prices) or m's utility. The input $\mathrm{sl}^t_m = (\rho^t_m, e^t_m)$ and actual actions $\mathbf{a}^t_m$ are stored in SL memory to train the regression model. SL infers behavioral strategy ψ^t_m purely based on m's knowledge of its private bidder information and observation of the environment.

The modified FSP and RL algorithms (as an extension to Fig. 4.2) are in Algorithms 6 and 7.

Algorithm 6 FSP with featurized state vector

```
1: Initialize $\psi_m, \zeta_m$ arbitrarily,$v, t = 1, \eta = 1/t, P_{-m}^{t-1} = \mathbf{0}, u_m^{t-v+1}, \cdots, u_m^{t-1} = 0$, observe $e_m^t$, create $\mathrm{rl}_m^t, \mathrm{sl}_m^t$ and add to memory
2: while true do
3:     Take action $\mathbf{a}_m^t = (1-\eta)\psi_m^t + \eta\zeta_m^t$
4:     Receive $P_{-m}^t$, calculate $u_m^t$, observe $\rho_m^{t+1}, \mathbf{e}_m^{t+1}$
5:     Create and add state to RL memory: $\mathrm{rl}_m^{t+1}$
6:     Create and add state to SL memory: $(\mathrm{sl}_m^{t+1}, \mathbf{a}_m^t)$
7:     Construct $S_m^t, S_m^{t+1}$
8:     Get $\phi_m^t, \phi_m^{t+1}, r_{i,m}^t = \mathrm{Curiosity}(S_m^t, S_m^{t+1}, \mathbf{a}_m^t)$
9:     Get $\zeta_m^{t+1} = \mathrm{RL}(\phi_m^t, \phi_m^{t+1}, r_{i,m}^t)$
10:    Get $\psi_m^{t+1} = \mathrm{SL}(\mathrm{sl}_m^{t+1})$
11:    $t \leftarrow t+1, \eta \leftarrow 1/t, \zeta_m^t \leftarrow \zeta_m^{t+1}, \psi_m^t \leftarrow \psi_m^{t+1}$
12: end while
```

Algorithm 7 RL with featurized state vector

```
1: Initialize $\theta, w$ arbitrarily. Initialize $\lambda$
2: while true do
3:     Input $t$ and $\phi_m^t, \phi_m^{t+1}$
4:     Run critic and get $\hat{V}(\phi_m^t, \mathbf{w}), \hat{V}(\phi_m^{t+1}, \mathbf{w})$
5:     Calculate $\bar{r}_{i,m} = \lambda \bar{r}_{i,m}$ and $\delta$
6:     Run actor and get $\mu(\theta), \Sigma(\theta)$
7:     Sample $\zeta_m^{t+1}$ from $F(\mu, \Sigma)$, update $\mathbf{w}$ and $\theta$
8: end while
```

Next, we describe the curiosity learning and credit assignment modules in detail, which are key to the long-term algorithm.

5.4.2 Curiosity module

The curiosity module used in this thesis is based on the vanilla model from [113]. The curiosity learning algorithm from [113] predicts future states and actions with two

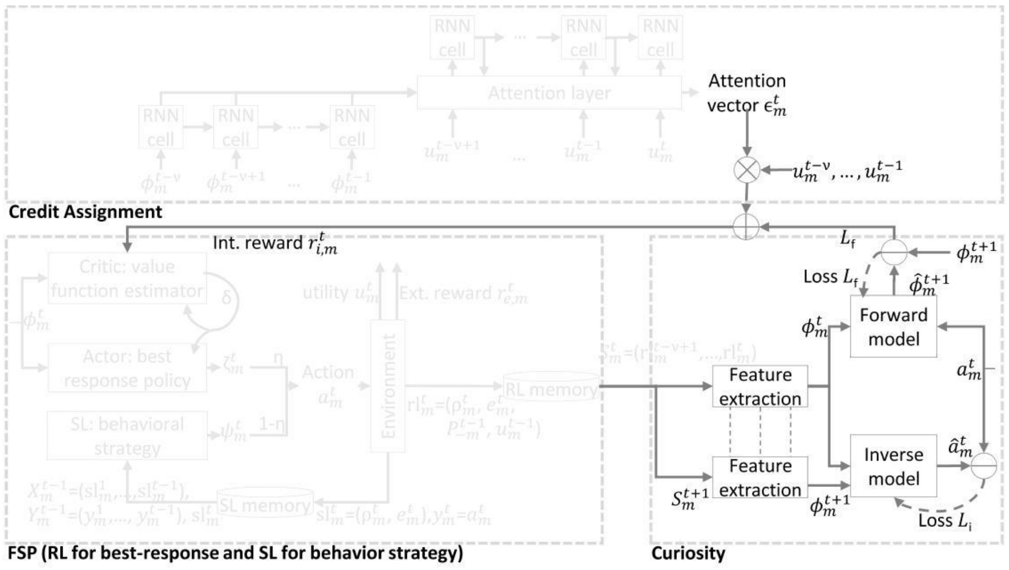


Figure 5.3: Curiosity module

supervised learning parts: the forward and the inverse submodules. The objective of these submodules is to predict the consequence of each action with minimal prediction losses $L_f^t = \|\phi_m^t - \hat{\phi}_m^t\|_2^2$ and $L_i^t = \|\mathbf{a}_m^t - \hat{\mathbf{a}}_m^t\|_2^2$, even without any reward signal. In this thesis, the setup also has short-term reward signals (not aligned and potentially conflicting with the extrinsic rewards); therefore, the author adapts the input to include past short-term reward values, and the forward submodule's objective is to improve prediction accuracy of both future states and future short-term rewards.

In [113], the intrinsic reward is the weighted loss of the forward submodule: $r_{i,m}^t = \xi L_f$, and the bigger the forward loss, the higher the intrinsic reward. Through the adversarial design, the curiosity learning algorithm is encouraged to explore state-action pairs where the agent has less experience, and prediction accuracy is low. The intrinsic rewards are inserted between sparse extrinsic rewards to improve learning efficiency despite the sparseness — the authors of [113] call this internal motivation "curiosity-driven exploration". In this thesis, the proposed approach (Fig. 5.3) applies the same method with a modified intrinsic reward definition: $r_{i,m}^t = \xi L_f^t + (1-\xi)\epsilon_m^t u_m^t$, where ξ is a predefined weight factor to balance between the two short-term objectives, and ϵ_m^t is a weight factor from the credit assignment module (see below). The objective is to maximize $\mathbb{E}_\pi[\sum_t r_{i,m}^t] - L_i^t - L_f^t$. Pseudo code is in Alg. 8.

Algorithm 8 Curiosity learning module

```
1: Initialize model parameters, $\epsilon_m^t$ arbitrarily. Initialize $\xi$
2: while true do
3:     Input $a_m^t$ and $S_m^t$, $S_m^{t+1}$ constructed from RL memory
4:     Run feature extraction, get $\phi_m^t$ and $\phi_m^{t+1}$
5:     Run forward submodule, get $\hat{\phi}_m^{t+1}$, calculate $L_f$
6:     Run inverse submodule, get $\hat{a}_m^t$, calculate $L_i$
7:     Update model parameters
8:     Infer from credit assignment, extract $\epsilon_m^t$ from attention layer
9:     Calculate and output $r_{i,m}^t$
10: end while
```

5.4.3 Credit assignment module

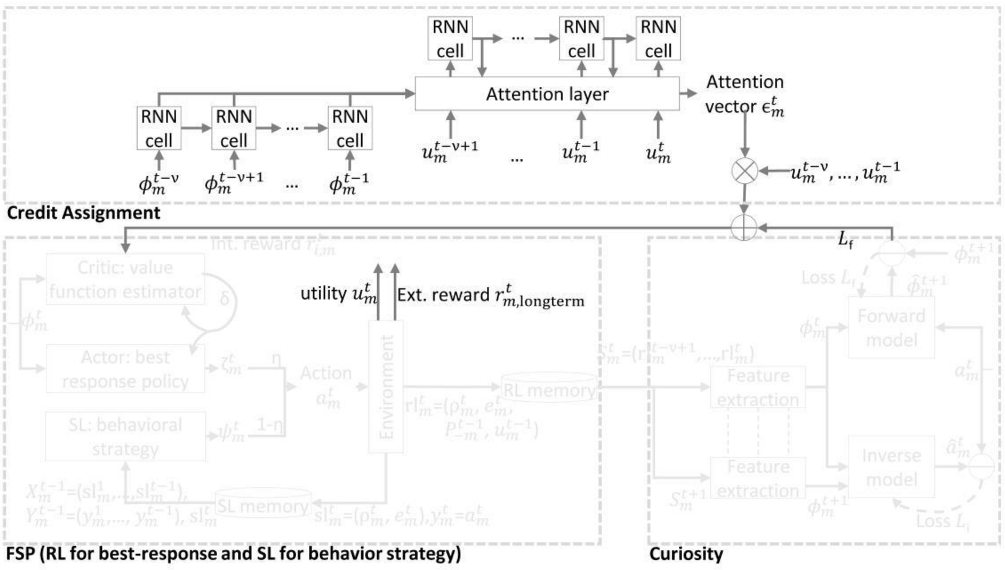


Figure 5.4: Credit assignment module

The proposed credit assignment module in this thesis (Fig. 5.4) uses a sequential network (recurrent neural network as encoder and decoder) with an attention layer. Typically, such a sequential network is used to identify correlation between sequenced input elements enc_i and predict a corresponding sequence of output elements $\hat{\text{dec}}_o$. The sequential network is enhanced with an attention layer, which establishes relationship between any elements in the sequence, regardless of the distance between them. The proposed credit assignment module is inspired by [51], it is different from [51] in that it does not decompose the extrinsic reward.

In this credit assignment module, we are not interested in predicting $\hat{\text{dec}}_o$. Instead, we want to determine the contribution of each state-action pair towards each long-term reward $r^t_{m,\text{longterm}}$. Therefore, the training of the credit assignment module is triggered only when there is a new signal $r^t_{m,\text{longterm}}$ at time t, the corresponding u^t_m becomes the last element of the target vector. We train the module on the batch of ν featurized state vectors $\text{enc}_i = \{\phi^{t-\nu}_m, \cdots, \phi^{t-1}_m\}$ with both short- and long-term rewards as target vector, $\text{dec}_o = \{u^{t-\nu+1}_m, \cdots, u^t_m\}$. In time step $\tau \in [t-\nu, t-1]$, the attention layer generates a weight vector corresponding to input vector enc_i, marking its relevance to the current output prediction $\hat{\text{dec}}^\tau_o$, until in the last time step t, the attention layer outputs a weight vector $\epsilon^t_m = \{\epsilon_1, \cdots, \epsilon_\nu | \sum^n_{i=1} \epsilon_i = 1\}$ corresponding to enc_i that marks their relevance to the last output u^t_m. Model parameters are updated with the mean square error between the generated output $\hat{\text{dec}}_o$ and target vector dec_o.

The weight vector ϵ^t_m is then multiplied with the original utilities u^t_m. Through ϵ^t_m, short- and long-term rewards are aligned, even if they are conflicting in nature. Between sparse extrinsic rewards, only the forward network of credit assignment module is run to infer a weight vector. Pseudo code is in Alg. 9.

Algorithm 9 Credit assignment module

```
1: Initialize module parameters arbitrarily, initialize batch size $\nu$
2: Input $S^{t-1}_m, \cdots, S^{t-\nu}_m, u^t_m, \cdots, u^{t-\nu+1}_m$ from RL memory
3: Run feature extraction and get $\phi^{t-1}_m, \cdots, \phi^{t-\nu}_m$
4: for $\tau \leftarrow t-\nu$ to $t-1$ do
5:     Input $\phi^\tau_m$ to encoder, get encoder output $\text{enc}_o$
6:     Input $\text{enc}_o, u^{\tau+1}_m$ to decoder, get output $\text{dec}^{\tau+1}_o$
7: end for
8: Calculate prediction loss $||\hat{\text{dec}}_o - u_m||^2_2$
9: Update model parameters, output $\epsilon^t_m$ from attention layer
```

To summarize: the features that make the proposed algorithm truly long term are: 1) reward prediction, 2) more exploration in the early stages of learning, and 3) short- and long-term reward alignment through credit assignment. Points 1) and 2) are achieved through an adapted curiosity module (Sec. 5.4.2). Point 3) is achieved by a hierarchical structure that uses an attentional network to learn and assign weights to short-term rewards based on their relevance to the long-term, sparse extrinsic reward; the learning process is only triggered when a new extrinsic reward becomes available. Between the extrinsic reward signals, the FSP+curiosity learns to better predict next states, actions and intrinsic rewards.

In this chapter, only the extrinsic reward is delayed; for the intrinsic reward, we measure offloading failure at the time of task admission. However, the proposed algorithm can also learn with delayed intrinsic rewards, e.g., if the measurement of offloading failure is after task execution. For simplicity, the author assumes that failure rate measured before and after actual task execution is the same. The author verifies this assumption in the next section, where it shows that applying the proposed solution, a system responsiveness [14] of 99% can be reached (i.e. 99% of the admitted jobs at MEC are successfully processed before their deadlines).

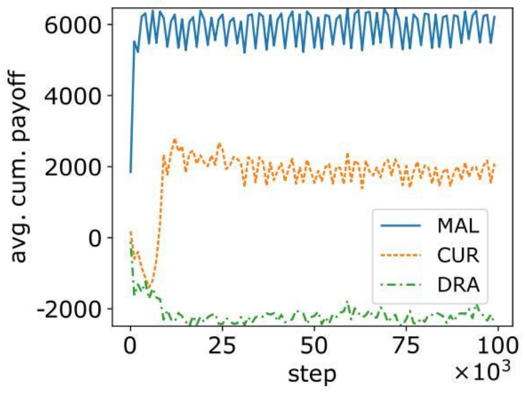
5.5 Evaluation

5.5.1 Synthetic setup results

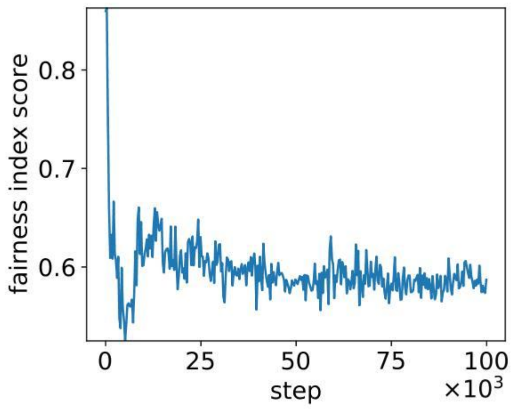
The synthetic setup consists of two common repeated auction games: a first-price reverse auction, and a second-price forward auction. Both games have six bidder agents and one auctioneer agent. The auctioneer is a passive agent without learning capabilities. In every time step, the auctioneer offers one commodity (e.g. object or service) for bidding, all bidders can join the auction simultaneously. The auctioneer grants the commodity to the bidder with the highest score; ties are broken randomly. An immediate payoff is given to the winner; the value of the payoff is specific to the type of game. Except for the winner, all other participating bidders pay a fixed cost for joining the auction.

All bidders start with an initial wealth; the wealth pool is updated every time step with payoffs and costs. Regardless of the bidder's behavior, there is a constant cost each time step (carrying cost). If the pool is depleted, the game is over for the bidder, it receives a penalty and rejoins the game with the same initial wealth. Otherwise, the game continues for a certain number of time steps (in our simulation we take $T = 150$ time steps). When the game ends, all bidders restart the game with the same initial wealth. In the case of long-term learning algorithms, bidder m receives a long-term extrinsic reward signal at the end of each game. It can be m's own cumulated payoff in the wealth pool: $r^t_{m,\text{longterm}} = B^t_m$, or overall fairness, defined as the J-index [68] of payments from the auctioneer to the bidder agents over time: $r^t_{m,\text{longterm}} = \frac{(\sum\limits_{m \in M} \sum\limits_{\tau=t-T}^{t} p^\tau_m)^2}{|M| \sum\limits_m (\sum\limits_{\tau=t-T} t p^\tau_m)^2}, \forall m \in M$. To preserve privacy, extrinsic reward signals do not contain private agent information.

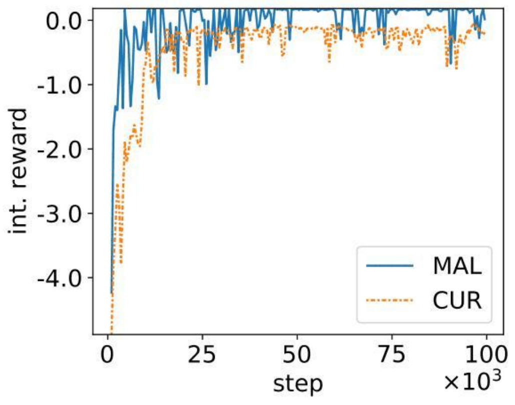
There are free and occupied bidders: if a bidder wins a bid, its resources are occupied for



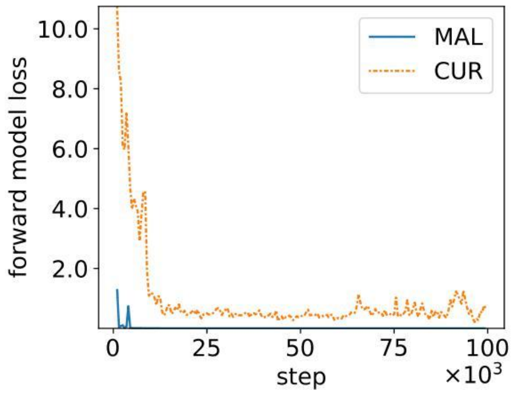
(a) Payoff performance per agent, by algorithm type. Three types of agents are pitched against each other in the same environment.



(b) Overall fairness performance.



(c) Intrinsic reward, comparison of MAL and CUR agents.



(d) Forward submodule loss, comparison of MAL and CUR agents.

Figure 5.5: First-price auction with HETERO agents and payoff-signal

a period of time, i.e. service duration, during which the occupied bidder cannot submit new bids. Each free bidder decides 1) whether to join the auction for the commodity in the current time step, 2) if so, a bidding price b that is lower than or equal to the amount in the wealth pool, and 3) other decision factors required by the specific setup, for example, service duration d.

The author assumes correlation between service duration d and bidding price b. The auctioneer gives each bidder a balanced score p_m. There are many ways for the auctioneer to rank the bids and determine the winner. In this thesis, the *winner determination problem* is not the focus, therefore the author greatly simplifies the calculation of p to multiplication of price and service duration. Winner m of the auction is the one with the highest score p_m. In the first-price reverse auction, it gets an immediate payoff of $b_m \cdot d_m$. In the second-price forward auction, it pays the auctioneer the second-highest score p^* among all bidders and gets an immediate payoff of $(b_m \cdot d_m - p^*)$. In both games, during the service duration d_m, the winner cannot join any new auctions.

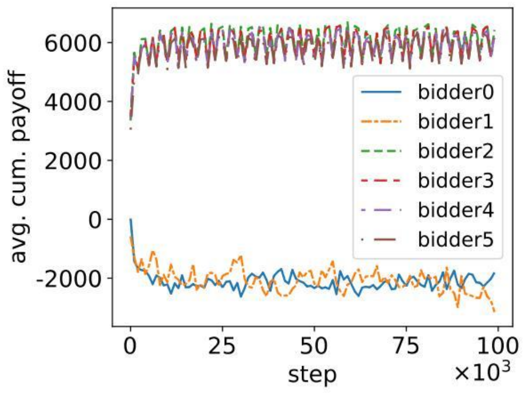
The bidders may use one of three learning algorithms: the short-term DRACO algorithm from the previous chapters (DRA), the long-term algorithm based on curiosity learning (CUR), or MALFOY (MAL), the long-term algorithm with an attention layer for credit assignment. In the setup with homogeneous agents, all six agents have an algorithm of the same type. In the setup with heterogeneous agents, each algorithm is given to two out of six bidders, and all algorithms compete in the same auction game.

To summarize, the author simulates first-price reverse (FP) or second-price forward (SP) auction, with homogeneous (MAL, CUR or DRA) or heterogeneous agents (HETERO) and use either average cumulated payoff per agent (payoff signal) or fairness index score (fairness signal) as extrinsic reward signals. As performance metric (Sec. 3.4), the author uses the average cumulated payoff per agent (payoff performance) and the overall fairness index score (fairness performance). All results come from continuous training.

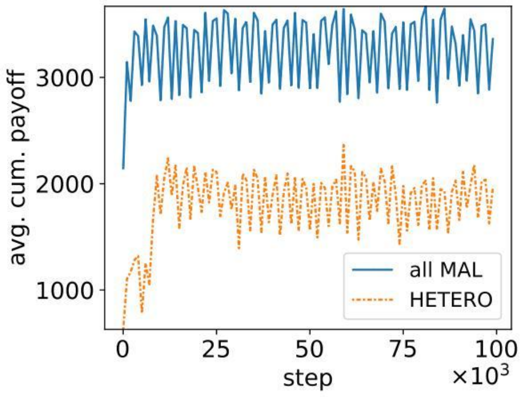
First-Price Reverse Auction (FP)

First-price reverse auction (lowest-bid-wins) is common e.g. in long-term energy contracts [87] or network resource allocation [154] where multiple resource owners bid to sell to one buyer that prefers low price for a long duration.

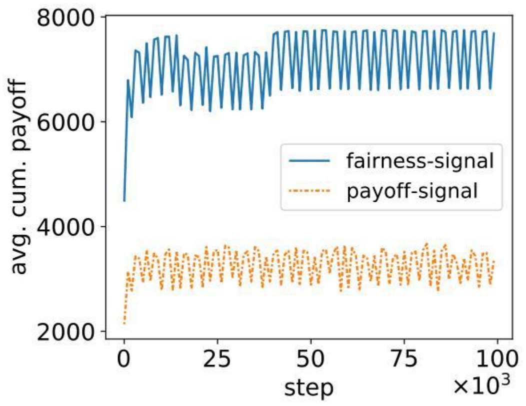
Each curve in Fig. 5.5a represents the average performance of two agents with the same type of algorithm, in a heterogeneous setting. Both MAL and CUR agents outperform DRA: through the reserve pool of wealth, current behavior influences bidding decisions



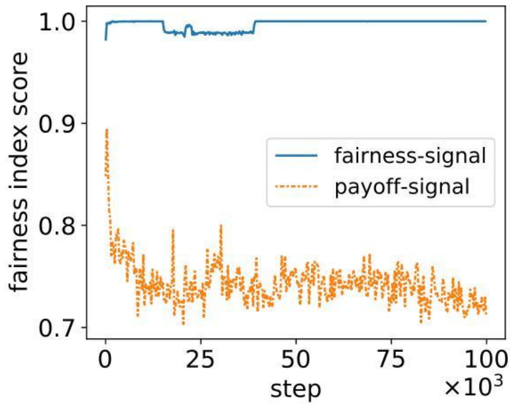
(a) Individual payoff performance per agent when only MALFOY agents exist in the environment: 4/6 agents receive max. reward.



(b) Average payoff performance comparison of the all-MAL auction, vs. the HETERO auction previously (average of Fig. 5.5a)



(c) Average payoff performance in all-MAL auction: if the long-term signal is a payoff signal to incentivize max. individual payoff, or fairness signal to incentivize system fairness. Counter intuitively, with fairness signal, agents become less aggressive and achieve higher payoff.



(d) Overall fairness performance comparison in all-MAL auction: if the long-term signal is a payoff signal to incentivize maximization of individual payoff, or fairness signal to incentivize system overall fairness.

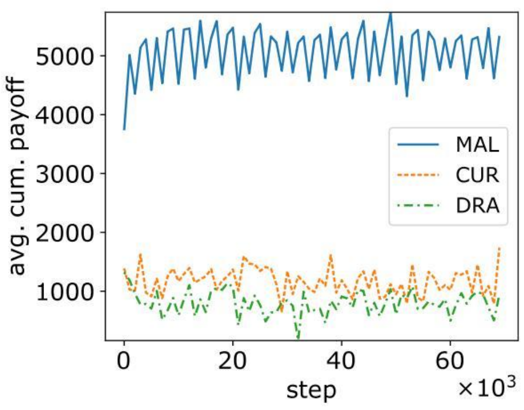
Figure 5.6: First-price auction with only MALFOY agents: payoff and fairness signal

in the future and has direct impact on the delayed extrinsic reward. However, the short-term algorithm values the immediate intrinsic reward much higher than the extrinsic reward in the distant future, therefore failing to succeed in the game. On the other hand, the MALFOY agents clearly perform the best but at the cost of other agents with less aggressive algorithms. This is shown by the low fairness index in Fig. 5.5b. Fig. 5.5c and 5.5d compare training performance of MAL and CUR agents in the game. The MALFOY agent not only converges faster, it also converges to lower loss and higher intrinsic reward.

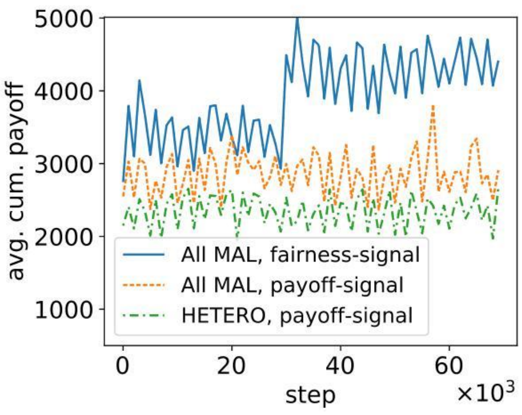
If we pitch the aggressive MALFOY agents against each other, i.e. all six agents are MALFOY agents, we have a similar result (Fig. 5.6a): only four MALFOY agents can maximize their cumulated payoff over time, yet the game still has a higher social welfare compared to the HETERO case (Fig. 5.6b). The difference in individual performance is caused by MALFOY agents' aggressive, selfish (i.e. with private individual goals), rational (i.e. act to maximize reward) behavior, and agents fall into different behavioral patterns over time, where winners profit from an unregulated system at the cost of social welfare. In fact, it is possible for all six agents to maximize their reward: to motivate cooperation, we replace the cumulated payoff with fairness index score as long-term extrinsic reward signal. The negative impact on social welfare can thus be prevented.

Fig. 5.6c and 5.6d compare two independent simulation results. The dotted orange curve is the average cumulative payoff of six MALFOY agents when the extrinsic reward is also the cumulative payoff. The solid blue curve is when the extrinsic reward is fairness index score. With fairness as incentive, all agents receive better cumulated payoffs, while achieving a much higher fairness index score. Hence, with the proposed solution, it is possible to increase both individual gain and social welfare. In real life, independent and selfish agents do not have to respect system objectives such as fairness or utilization. The effect of the agent's custom preference of objectives is explored in Chapter 6. In this chapter, we assume that the agents' preferences for objectives are given. However, agents with custom preferences are not necessarily malicious agents — they may ignore system objectives, but they do not have the sole individual objective of worsening system objective scores. The discussion on malicious agents is out of the scope of this study.

To wrap up, the first simulation setup (Fig. 5.5) demonstrates how the MALFOY algorithm learns quickly and aggressively in a multi-agent, dynamic environment with partial information, a big state-action space and sparse / delayed extrinsic reward. The second setup (Fig. 5.6c and 5.6d) demonstrates how MALFOY can be easily optimized to integrate a system goal while preserving privacy and individual goals.



(a) Second-price auction, payoff performance with all three algorithms in one environment (HETERO) and payoff signal: the result is similar to the previous first-price auction, MALFOY agents achieve the most individual payoff at the cost of other agents.



(b) Payoff-performance comparison with different combinations of algorithms and long-term incentivation signals. Result is similar to first auction: 1. average payoff in all-MAL auction is better than average payoff of HETERO; 2. fairness signal also leads to better individual payoff.

Figure 5.7: Second-price auction, all MALFOY vs. HETERO, payoff vs. fairness signal

Second-Price Forward Auction (SP)

In a second-price forward auction, the auctioneer is a seller that grants the commodity to the bidder with highest bidding price, but the payment for the commodity is the second-highest price of all bidding prices. This type of auction is common for selling public goods, maximizing welfare rather than seller profit, e.g. in networking resource allocation [155] and e-commerce [64], where multiple end users bid for resources from one service provider.

Fig. 5.7 shows similar results in SP as in FP. When three types of agents co-exist in a profit-oriented setup, the two MALFOY agents win at the cost of social welfare (Fig. 5.7a). Social welfare increases when all agents are MALFOY agents. Finally, if the MALFOY agents are instead given a fairness index as incentive, social welfare reaches is much higher. This can be seen from Fig. 5.7b: with fairness index score as extrinsic reward signal, social welfare increases.

The results indicate that in real life, as service providers on the operating side, to incentivize users to consider system objectives, there is need to provide system information as feedback, e.g. fairness and resource utilization. Although the users are free to ignore these system reward signals, the signals can provide extra information that is beneficial for the users to achieve their individual objectives. A smart design of system reward signals can help to align all objectives without sacrificing individual goals. Further analysis of system reward signals and multiple objectives is in Chapter 6.

5.5.2 Realistic setup results

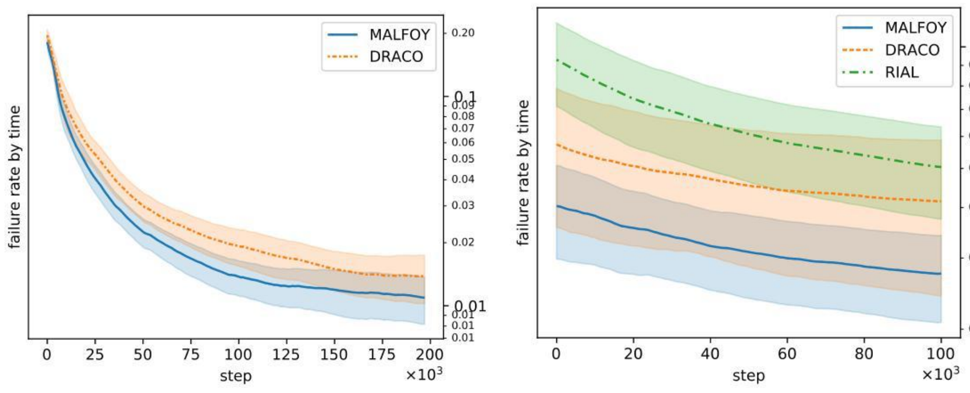
Again, we bring the MALFOY algorithm into the realistic V2X environment simulated with mobility data as in Table 3.1 of Chapter 3. The only difference in the simulation of MALFOY is the addition of a long-term reward signal from the auctioneer that is available after 2000 time steps. Performance measurements are defined in Sec. 3.4.

As mentioned in Sec. 3.3.1, the uplink and downlink time, service request arrival rate and service deadlines are based on the requirements of semantic segmentation and motion planning applications. If the vehicle expects its position before service deadline to be out of range of the MEC, the service request is dropped without any performance measurement.

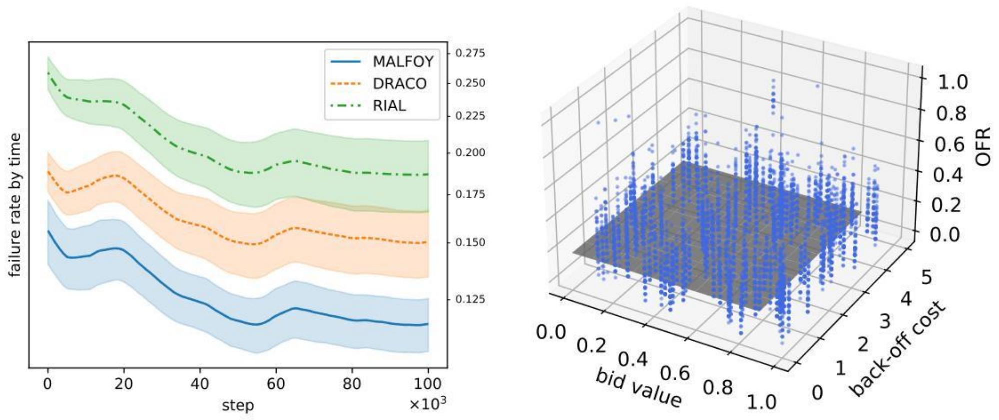
Higher vehicle arrival rate and slower driving speed typically lead to high contention. By changing the arrival rate and speed in the simulation, we create high and low-contention scenarios.

The author trains and tests the active agents with MALFOY in low contention, with long-term reward signal every 2000 time steps. MALFOY is tested against the short-term DRACO algorithm in the same setup. DRACO only considers the immediate effect of each action — for the long-term reward signal, the effect of only one previous action is considered. Fig. 4.8a-left shows that DRACO converges to OFR of 1.4%, and MALFOY converges much faster to an even lower failure rate. Then, the author evaluates the trained models in the same environment with newly simulated trace data from SUMO (Fig. 5.8a-right). DRACO still achieves an OFR of 4%, a reduction of 18% compared to the passive algorithm RIAL; MALFOY further reduces failure rate by 34%, compared to DRACO.

Then, the author tests (i.e. without retraining) the trained MALFOY models in a significantly different environment, changing traffic light phases, vehicle arrival rate and speed to make the environment more volatile and dynamic and reducing capacity to create a high-contention situation. The resulting vehicle count over time (same as in Fig. 4.8b-left) shows a much heavier and more frequent fluctuation compared to the original training environment. Despite the significant changes to the environment, and without requiring any further training, DRACO reduces failure rate by 20% compared to RIAL (passive agents with no learning capability, see Sec. 3.3), and MALFOY further reduces failure rate by 23% (Fig. 5.8b-left).



(a) Training environment: low contention with abundant resource, traffic phase=10-40s, low vehicle speed(10 km/h), low arrival rate(1 every 2.2s), low variation in vehicle count(22-30): OFR in training(left) and evaluation(right). MALFOY learns faster in training, and outperforms both DRACO and RIAL.



(b) Test environment: high contention with limited resource, traffic phase=20s, high vehicle speed(30 km/h), high arrival rate(1 every 1s), high variation in vehicle count(14-30): OFR (left) and sensitivity analysis (right). Left: OFR performance of MALFOY is even more distinguishable from DRACO and RIAL. Right: Individual OFR is not sensitive to private bid values v and backoff cost q (normalized).

Figure 5.8: Comparison of offloading failure rate (OFR) in training and test environments, between: MALFOY, DRACO, and RIAL only.

Fig. 5.8a shows good convergence speed despite computation and communication complexity of the problem. Fig. 5.8b shows that with the capability to predict long-term impacts of each action, MALFOY has even better performance and generalization properties. With little need for retraining in a new environment, the computation delay is only the time for model inference.

Additionally, the author randomizes each vehicle's private bid valuation $v_{m,k}$ and the backoff cost $q^t_{m,k}$, to analyze how sensitive the individual offloading failure rate (OFR) is to changes in v and q. Results from this sensitivity analysis show that the changes in v and q have almost no impact on the individual OFR; the Pearson coefficient values are 0.008 (p-value=0.5) between OFR performance and v values, and 0.007 (p-value=0.5) between OFR performance and q values. Fig. 5.8b-right visualizes this result. This and the results in Fig. 4.7 demonstrate the robustness of the proposed auction mechanism: vehicles learn to compensate for differences in initial parameterization through trade-off in bidding price and backoff time, without impact on individual OFR.

5.6 Conclusion of the chapter

The extended MALFOY algorithm can utilize long-term, sparse reward signals and has enhanced predictive power, as well as better alignment between short-term and long-term goals. When behaving long-term, it shows further performance improvements. A sensitivity analysis of OFR with regard to bid valuation and backoff cost shows the robustness of the proposed solution.

The ablation study with the synthetic setup shows that each module of the MALFOY algorithm increases overall performance, and the effect is compounded. Therefore, this chapter and the following chapters use all three modules in the proposed algorithms. Despite the complexity, the next chapter shows that the modules can be trained in a distributed and asynchronous manner, greatly improving computation performance.

Until now, this thesis has only dealt with single-agent and multi-agent RL algorithms that learn a single objective. Although both short-term and long-term objectives are considered in this chapter, they are scalarized with a fixed preference vector into a single objective. However, over time, the agent's preferences to its private objectives as well as the system's incentives may change, and the single-objective model will not adapt to the new objectives. The next chapter discusses multi-objective optimization problems and proposes an extension to the MALFOY algorithm to learn multiple objectives.

Chapter 6

Multi-objective Optimization

6.1 Motivation

The single-agent and multi-agent RL algorithms from the previous chapters focus on single objectives, although many real-life decision-making problems are multi-objective in nature [39]. In the previous single-objective algorithms, the multiple objectives are scalarized into a single objective through a constant preference weight vector.

A multi-objective problem (MOP) is different from a single-objective problem (SOP) where the objective is scalar and totally ordered: in an MOP, the set of objectives are only partially ordered [131]. An MOP is formulated as finding values for decision variables that leads to non-dominated $f(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_l(\mathbf{x}))$ s.t. $\mathbf{x} \in \mathbf{K} \subseteq \mathbb{R}^n$, where f is a vector of l objective functions, $\mathbf{x}$ is the decision variable, and $\mathbf{K}$ is the feasible region in an n-dimensional decision variable space. Since f can only be partially ordered, we use *Pareto frontier* to represent a set of equivalent solutions, where one objective cannot be improved without at least one other objective being worsened. Put differently, no two members of the Pareto frontier are better than each other, and there are no better solutions outside of the Pareto frontier.

Converting an MOP into an SOP through scalarization is the most common way to address an MOP [144], such as in [54, 56, 161]. But in some scenarios, the conversion 1) is impossible when utility or user preference over each objective is unknown *a priori*, changing fast or incommensurate, or 2) is intractable with high dimensionality or non-convexity, or 3) performs bad because a single-objective learning algorithm cannot track the development of reward on multiple objectives [11, 59]. But these aspects describe typical networking scenarios, necessitating a new approach for this thesis.

This chapter proposes a MARL algorithm to address MOP in a dynamic and adversarial environment. The contributions are:

- To the best of the author's knowledge, this thesis is the first one to address the multi-objective nature of V2X applications in its distributed, non-stationary and adversarial environment. The proposed multi-agent, multi-objective algorithm can optimize frequently changing combinations of objectives and preferences.
- The author trains one optimal initial model offline, then deploys the model to each independent agent representing a vehicle user, who is able to change its private objectives and update its offloading strategy through online few-shot learning, needing low retraining cost and no prior knowledge for reward shaping. This approach outperforms the benchmarking state-of-the-art algorithms on all individual and system metrics. Also, in a heterogeneous environment with different competing algorithms, this approach increases bottom-line resource efficiency, such that other algorithms in the environment also benefit from improved offloading rate and fairness.
- The proposed algorithm can be modularized and trained asynchronously. The author tests the runtime inference performance of the proposed algorithm on a single-board computer with a GPU and shows that inference in 6 milliseconds is feasible.
- The author provides public access to the code and data at [3].

6.2 Chapter outline

Table 6.1 lists the difference between three algorithms and simulation setups introduced in chapters 4 to 6.

Sec. 6.3 introduces the background and related work in multi-objective optimization problems, including the single-model method used as basis for the proposed algorithm as well as a recount of its first-order estimate. Sec. 6.4 reformulates the original problem into a long-term multi-objective one. Sec. 6.5 proposes the third and final algorithm, that is the same structure as the previous chapter (Fig. 5.1), but trained in a two-phase loop, including an inner loop to train independently for different objectives and an outer loop to consolidate local single models' gradients. The training result is an initial generic model that is deployed to each local agent that can quickly adapt to very different objectives and environments. Sec. 6.6 evaluates the algorithm. Sec. 6.7 addresses some practicality concerns and tests the algorithm on a single-board computer. Sec. 6.8 concludes the section.

Table 6.1: Chapter6 outline

	Chapter4: DRACO	Chapter5: MALFOY	Chapter6: MOODY
Alg. type	short term	long term	long term
Obj. type	single objective	single objective	multiple objectives
Synthetic seup	Simulated second-price forward auction in V2X application scenario, with varying resource capacity, service request types, number of rebidding (Sec. 4.6.1). Purpose: analyze algorithm performance with randomized inputs and a wide range of environment parameters.	Simulated two simplified common repeated auctions with no specific application scenario: first-price reversed and second-price forward auction, with fixed resource capacity, one commodity type, no rebidding (Sec. 5.5.1). Purpose: analyze contribution of each module and different system reward signals.	No synthetic setup
Realistic setup	Same as in Sec.3.3.1. Purpose: analyze and compare performance in simulated traffic.		

6.3 Preliminaries and related work

A multi-objective (or multi-criteria, multi-task) optimization problem is formulated as:

Find non-dominated values of: $f(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_l(\mathbf{x}))$ subject to $\mathbf{x} \in \mathbf{K}$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}^l$ is the objective function, $\mathbf{x}$ is an n-dimensional decision variable vector, $\mathbf{K} \subseteq \mathbb{R}^n$ is the feasible region in a n-dimensional decision variable space. $f(\mathbf{K})$ represents the feasible objective region, which is also a subset of the objective space $\mathbb{R}^l$.

Unlike in single-objective optimization problems where the fitness of the solutions can be fully ordered, in multi-objective optimization problems, the fitness of a solution is defined by its *dominance* over other solutions. Formally, $f(\mathbf{x}_1)$ *dominates* $f(\mathbf{x}_2)$, or $f(\mathbf{x}_1)$ is a better solution than $f(\mathbf{x}_2)$, if 1) no objective in $f(\mathbf{x}_1)$ is worse than in $f(\mathbf{x}_2)$, and 2) at least one objective in $f(\mathbf{x}_1)$ is strictly better than in $f(\mathbf{x}_2)$.

If a solution $f(\mathbf{x})$ is not dominated by any other solution in the solution set $f(\mathbf{K})$, it is a *non-dominated solution*. The subset of all non-dominated solutions in the solution set is called the *Pareto-optimal set*. The *Pareto frontier* symbolizes the boundary defined by the Pareto-optimal set, where any objective cannot be optimized without other objectives being worsened. Due to conflicts between objectives, it is typically not possible to find a single solution that is optimal in all objectives.

6.3.1 Single-objective RL for solving MOP

Most RL algorithms only solve SOP [138]. The goal of a single-objective RL algorithm is to maximize the return $J = \mathbb{E}[\sum_{t=0}^{T} r_t]$, where T is the time horizon and r_t is a scalar-valued reward at time t. In a multi-objective RL algorithm, we have a vector-valued return for $|O|$ objectives: $\mathbf{J} = \{J_{(o)}|o = \{1, \ldots, O\}\} \in \mathbb{R}^{|O|}$, the return for each objective o is $J_{(o)} = \mathbb{E}[\sum_t r_{o,t}]$. Owing to partial order of rewards, such a situation is not directly amenable to standard RL techniques. One option is, hence, to try to cast this problem back into a conventional SOP setting.

Weighted sum is the most widely used classical approach in scalarization [54, 161]. An MOP can be simplified to an SOP through a constant weight vector $\mathbf{W} \in \mathbb{R}^{|O|}$ to form a single reward, such that classic single-objective RL algorithms can be applied to solving MOP. Using a constant weight vector is equivalent to finding one optimal point on the Pareto frontier. Even if the simplification is acceptable in some cases, reference [59] points out that such simplification requires much *a priori* knowledge for reward engineering and manual tuning when preferences of objectives change over time; a scalarized reward is also inexplainable, i.e., the scalarization does not always reflect the real relationship between decision variables and objectives. Such methods are therefore sensitive to preference changes.

Instead, multi-objective RL algorithms aim at explicitly finding the shape of the Pareto frontier. This befits the studied V2X setup where each vehicle is free to change their preferences of multiple objectives over time, i.e. the preference weight vector $\mathbf{W}^t$ is time-variant and unknown *a priori*. Such multi-objective RL algorithms can be roughly categorized into multi-model and single-model methods, described next.

6.3.2 Multi-model, multi-objective RL

Multi-model methods aim at finding a finite set of non-dominated points on the Pareto frontier; each point is learned through one model. The parameters of each model need to be stored. The most straightforward extension from SOP to an MOP with the multiple-model method is to sample different preference weight vectors (i.e., preference for each objective), and for each such vector, train a separate SOP model. During runtime, if a previously unknown weight vector is given, All trained models have to be evaluated in order to select one out of all models for further training. We give two examples of the multi-model method below.

The *radiant algorithm (RA)* scalarizes the gradient from each objective $m \in \{1, \dots, M\}$ to efficiently find one non-dominated point $i \in \{1, \dots, N\}$ for a sampled weight vector $\mathbf{w}_i$ on the Pareto frontier. Gradient for the ith model is

$$S_i(\mathbf{w}_i, \theta_i) = \mathbf{w}_i^T \cdot \nabla \mathbf{J}_i, \text{ where } \nabla \mathbf{J}_i = \begin{bmatrix} \nabla_{\theta_i} J_{(i,1)} \\ \vdots \\ \nabla_{\theta_i} J_{(i,M)} \end{bmatrix}.$$

Update in each model is $\theta_i = \theta_i + \alpha S_i$, where α is the learning rate.

The *Pareto following algorithm (PFA)* tries to find one non-dominated point first with one objective, then moves along the Pareto frontier to find other non-dominated points, adding objective dimensions incrementally. The update rule is:

$$\theta_i = \mathrm{OPT}(\theta_{i-1} + \alpha w_i \nabla_{\theta_{i-1}} J_i(\theta_{i-1}))$$

where OPT is any optimization method that finds a set of θ to optimize J_i, w is a weight factor of the gradient, and α is the learning rate.

Multiple-model methods pose several problems: 1) choice of weight vector samples may not represent the shape of the Pareto frontier; 2) since each model is independently trained, they do not share learning information, making learning inefficient; 3) each time a previously unknown weight vector is given, all the models need to be evaluated, making the method inflexible in a dynamic environment where user preferences change frequently; 4) it is not obvious how to choose one model or combine several models as initial points for additional training. The two multi-model methods proposed in [111] either have many individual models to evaluate whenever user preferences change or have high computational cost in high-dimensional objective spaces.

6.3.3 Single-model, multi-objective RL

Single-model methods train only one model for all non-dominated solutions to the MOP. Reference [117] proposes an algorithm that learns parameters that best estimate a continuous Pareto frontier. Once trained, the model does not need any retraining when user preferences change. However, it is computationally expensive, requiring estimating a Hessian matrix of the expected return w.r.t. the model parameters that is of size $qd \times d$ where q is the number of objectives and d is the number of parameters.

MAML [52] is an approach that combines multi-model and single-model methods: multiple models are trained for their specific objectives in an inner loop and consolidated into a single model in an outer loop, which can be quickly retrained for any new objective with only a few sample data points (*"shots"*). Although theoretically the method requires estimating a Hessian, the authors propose a first-order estimation and prove its sufficiency under the assumption that the inner loop uses a small learning rate and n-shot learning with very small n. The original MAML algorithm from [52] addresses the problem of a single agent learning to do multiple tasks. Both references [72] and [9] study MAML for multiple agents, but the former considers a stationary environment, and the latter formulates a non-stationary SOP as a stationary MOP. To the best of the author's knowledge, single-model, multi-objective RL approaches similar to MAML have not been studied in a multi-agent, non-stationary environment with multiple objectives in V2X applications.

The proposed approach is based on the MAML inner/outer loop training concept. The author uses the long-term algorithm from the previous chapter that is specifically designed for a dynamic environment with sparse and delayed rewards and capable of automatically triggering adaptive few-shot retraining (Sec. 6.5). The author also applies various performance improvement measures (Sec. 6.7). As a basis for the proposed algorithm, the author describes MAML in detail below.

6.3.4 MAML

MAML tries to find a model initialization that produces close-to-good results for *any* preference vector. With MAML, the initial model needs to be trained extensively; it still requires a very small amount of data samples to quickly learn any new preference vector afterwards (few-shot learning as a type of transfer learning). Although implicit, at training time, N local models are run in memory before gradients are consolidated in a generic model (parallel stochastic gradient ascent).

MAML's *inner loop* optimizes local models for a single task or any scalarized objective from the objective space of the original MOP.

$\mathcal{J}_i$ is the expected scalarized return through any scalarization method for the ith model, $i \in \{1, \ldots, N\}$. The initial model is denoted f_θ with parameters θ. Thus, $\mathcal{J}_i$ depends on θ. Formally: $\mathcal{J}_i = \mathbb{E}[\sum_{t=1}^{\infty} r_i^t(x, f_\theta(x))]$; for policy gradient methods, the model f_θ is the parameterized policy π_θ. r_i^t is the scalarized reward for model i at time t, which is a weighted sum of all rewards on all objectives. The weight vector w_i is different for each model i.

The update for each local model i is the same as for a single-objective model:

$$\theta_i^{(\tau)} = \theta_i^{(\tau-1)} + \Gamma \delta_i^{(\tau-1)} \nabla_{\theta_i^{(\tau-1)}} \ln \pi_{\theta_i^{(\tau-1)}}, \tau \in \{1, \cdots, \mathcal{T}\}, \theta_i^{(0)} = \theta \tag{6.1}$$

where τ is the index of training data, δ can be TD error or advantage, depending on the algorithm used, and Γ is the learning rate.

Outer loop: optimize model for all the differently weighted objectives

In the *outer loop*, the generic RL model's objective is to maximize the sum of all returns (i.e., expected future reward) from all local models. Formally, it is to find model parameters θ to maximize $\mathcal{J} = \sum_i \mathcal{J}_i$.

Because the meta-learning step in the outer loop has a different objective function from the inner loop algorithm, there is a second pass of backpropagation from the objective function in the outer loop to $\theta^{(0)}$. Since every intermediate set of parameters $\theta_i^{(\tau)}$ is based on $\theta_i^{(\tau-1)}$ and in turn based on the initial parameters $\theta_i^{(0)}$, the gradient of the outer loop objective function is backpropagated to $\theta_i^{(t)}, \forall t = \{0, \cdots, \tau\}$ again. In the proposed algorithm, initial model parameters $\theta_i^(0)$ are randomly sampled and the same for every model i.

If $\mathcal{T} = 1$, we can simplify the update rule to $\theta = \theta + \Gamma_{\text{outer}} \nabla_\theta \sum_i \mathcal{J}_i\big(\pi_{\theta+\Gamma\nabla_\theta \mathcal{J}_i(\pi_\theta)}\big) = \theta + \beta S$, where S is the collected gradient for the outer loop, and Γ_{outer} is the learning rate in the outer loop that may be different from Γ, the learning rate in the inner loop.

For $\mathcal{T} \geq 2$, we need to calculate the inner product of all previous updates. To simplify the notation, we define the following RL operator:

$$J_{(i,\tau)} : \pi_{\theta_i^{(\tau)}} \rightarrow \mathcal{J}_i \tag{6.2}$$

where $i \in \{1, \cdots, N\}, \tau \in \{1, \cdots, \mathcal{T}\}, \theta_i^{(0)} = \theta$, and

$$U_{(i,\tau)}(\theta) = \theta_i^{(\tau)} + \Gamma \nabla_{\theta_i^{(\tau)}} J_{(i,\tau)}(\theta_i^{(\tau)}) \tag{6.3}$$

We rewrite the outer-loop gradient S as follows:

$$\begin{aligned} S &= \sum_i \nabla_\theta J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \\ &= \sum_i \nabla_\theta J_{(i,\mathcal{T})}\Big(U_{(i,\mathcal{T}-1)}\big(U_{(i,\mathcal{T}-2)}\big(\ldots\big(U_{(i,1)}(\theta)\big)\big)\big)\Big) \\ &= \sum_i \nabla_\theta J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \cdot \nabla_\theta U_{(i,\mathcal{T}-1)}(\theta_i^{(\mathcal{T}-1)}) \cdots \nabla_\theta U_{(i,1)}(\theta) \\ &= \sum_i \nabla_\theta J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \cdot \big(I + \Gamma \nabla_\theta^2 J_{(i,\mathcal{T}-1)}(\theta_i^{(\mathcal{T}-1)})\big) \cdots \big(I + \Gamma \nabla_\theta^2 J_{(i,1)}(\theta)\big) \\ &= \sum_i \nabla_\theta J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \cdot \Big(\prod_{\tau=1}^{\mathcal{T}-1} \big(I + \Gamma \nabla_{\theta_i^{(\tau)}}^2 J_{(i,\tau)}(\theta_i^{(\tau)})\big)\Big), \forall i \in \{1, \ldots, N\} \end{aligned} \tag{6.4}$$

In the original MAML algorithm, we need to calculate for each sample τ in each model i, the inner product of every $\nabla_{\theta_i^{(\tau)}}^2 J$.

If Γ and $\mathcal{T}$ are small enough, we can omit the product of multiple Hessian matrices, Eq. 6.4 can be further simplified into:

$$S_{\text{MAML}} \approx \sum_i \nabla_\theta J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \cdot \Big(I + \Gamma \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta^2 J_{(i,\tau)}(\theta_i^{(\tau)})\Big) \tag{6.5}$$

We can also write U as the sum of updates on θ with each sample:

$$\begin{cases} U_{(i,\mathcal{T}-1)} = \theta_i^{(\mathcal{T})} & = \theta_i^{(\mathcal{T}-1)} + \Gamma \nabla_{\theta_i^{(\mathcal{T}-1)}} J_{(i,\mathcal{T}-1)}(\theta_i^{(\mathcal{T}-1)}) \\ & \vdots \\ \theta_i^{(1)} & = \theta + \Gamma \nabla_\theta J_{(i,0)}(\theta) \end{cases}$$

$$\implies U_{(i,\mathcal{T}-1)} = \theta_i^{(\mathcal{T})} = \theta + \Gamma \sum_{\tau=1}^{\mathcal{T}-1} \nabla_{\theta_i^{(\tau)}} J_{(i,\tau)}(\theta_i^{(\tau)}), \forall i \in \{1, \ldots, N\} \tag{6.6}$$

We use these formulations to understand the core difference between MAML and its first-order estimate.

6.3.5 Simplification of MAML's second-order derivative

For performance, the authors of [52] and [103] provide an estimation of the Hessian (first-order MAML, or FOMAML for short) in the case that we update the inner gradient for each individual objective with $\mathcal{T} \geq 2$ shots, written as:

$$S_{\text{FOMAML}} = \sum_i \nabla_{\theta_i^{(\mathcal{T})}} J_{(i,\mathcal{T})}(\theta_i^{(\mathcal{T})}) \tag{6.7}$$

FOMAML ignores the second-order derivatives, with the assumption that learning rate Γ and inner loop sample size $\mathcal{T}$ are both small. The authors conclude that the second-order derivative is not the most contributive to the learning but the meta-gradient is. By replacing the second-order derivative with a first-order estimation, the authors claim a 33% increase in learning speed.

To demonstrate the difference between MAML and FOMAML, MAML is analyzed through Taylor expansion [103]. We repeat the analysis in this section. To simplify the notation, we omit the model index $i \in \{1, \ldots, N\}$ and focus on the gradient for one model i. The final gradient is the sum of gradients of the inner loop return $J_{(i,\mathcal{T})}$ with regard to original parameters $\theta^{(0)} = \theta$.

First, we rewrite $S_{(\text{FOMAML,i})}$ with Taylor expansion:

$$S_{(\text{FOMAML,i})} = \nabla_\theta J_{(\mathcal{T})}(\theta^{(\mathcal{T})}) \approx \nabla_\theta J_{(\mathcal{T})}(\theta) + \nabla_\theta^2 J_{(\mathcal{T})}(\theta) \cdot (\theta^{(\mathcal{T})} - \theta) \tag{6.8}$$

From the formulation of Eq. 6.6, we can derive $\theta^{(\mathcal{T})} - \theta$ and get:

$$\begin{aligned} S_{\text{(FOMAML,i)}} &= \nabla_\theta J_{(\mathcal{T})}(\theta^{(\mathcal{T})}) \\ &\approx \nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta J_{(\tau)}(\theta^{(\tau)}) \end{aligned} \tag{6.9}$$

From the Taylor expansion of $\nabla_\theta J_{(\tau)}(\theta^{(\tau)}) \approx \nabla_\theta J_{(\tau)}(\theta)$, we get:

$$\begin{aligned} S_{\text{(FOMAML,i)}} &= \nabla_\theta J_{(\mathcal{T})}(\theta^{(\mathcal{T})}) \\ &\approx \nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta J_{(\tau)}(\theta) \end{aligned} \tag{6.10}$$

In the same way we expand $\nabla^2_\theta J_{(\tau)}$:

$$\nabla^2_\theta J_{(\tau)}(\theta^{(\tau)}) \approx \nabla^2_\theta J_{(\tau)}(\theta) \tag{6.11}$$

Next, we revisit the gradient of the meta-learning in Eq. 6.5 and rewrite $S_{\text{(MAML,i)}}$ into:

$$\begin{aligned} S_{\text{(MAML,i)}} &= \nabla_\theta J_{(\mathcal{T})}(\theta^{(\mathcal{T})}) \cdot \Big(I + \Gamma \sum_{\tau=1}^{\mathcal{T}-1} \nabla^2_\theta J_{(\tau)}(\theta^{(\tau)}) \Big) \\ &\approx \Big(\nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta J_{(\tau)}(\theta) \Big) \cdot \Big(I + \Gamma \sum_{\tau=1}^{\mathcal{T}-1} \nabla^2_\theta J_{(\tau)}(\theta^{(\tau)}) \Big) \\ &\approx \nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta J_{(\tau)}(\theta) + \Gamma \cdot \nabla_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla^2_\theta J_{(\tau)}(\theta^{(\tau)}) \end{aligned} \tag{6.12}$$

And according to Eq. 6.11:

$$\begin{aligned}S_{(\text{MAML,i})} &\approx \nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla_\theta J_{(\tau)}(\theta) + \Gamma \cdot \nabla_\theta J_{(\mathcal{T})}(\theta) \cdot \sum_{\tau=1}^{\mathcal{T}-1} \nabla^2_\theta J_{(\tau)}(\theta) \\ &= \nabla_\theta J_{(\mathcal{T})}(\theta) + \Gamma \cdot \sum_{\tau=1}^{\mathcal{T}-1} \left(\nabla^2_\theta J_{(\mathcal{T})}(\theta) \cdot \nabla_\theta J_{(\tau)}(\theta) + \nabla_\theta J_{(\mathcal{T})}(\theta) \cdot \nabla^2_\theta J_{(\tau)}(\theta) \right)\end{aligned} \tag{6.13}$$

This the complete gradient for model i at the end of the inner loop, expanded at the point θ, according to the original MAML.

For the average of $\mathcal{T}$ samples, we rewrite Eq. 6.13 into:

$$\mathbb{E}[\nabla_\theta J_{(\mathcal{T})}(\theta)] + (\mathcal{T} - 1) \cdot \Gamma \cdot \mathbb{E}[\nabla^2_\theta J_{(\mathcal{T})} \cdot \nabla_\theta J_{(\tau)} + \nabla_\theta J_{(\mathcal{T})} \cdot \nabla^2_\theta J_{(\tau)}] \tag{6.14}$$

Assuming we randomize data in the experience replay to de-correlate input data and stabilize learning (which is the case in the proposed algorithm), the index is interchangeable. Therefore, the second expected value is equivalent to $2 \cdot \mathbb{E}[\nabla_\theta(\nabla_\theta J_{(j)} \cdot \nabla_\theta J_{(\tau)})], j, \tau \in \{1, \ldots, \mathcal{T}\}$

We can now define the two expected values as:

$$\begin{cases}\text{AvgGradOuter} & = \mathbb{E}[\nabla_\theta J_{(\mathcal{T})}] \\ \text{AvgGradInner} & = \mathbb{E}[\nabla_\theta(\nabla_\theta J_{(j)} \cdot \nabla_\theta J_{(\tau)})]\end{cases} \tag{6.15}$$

And the expected MAML gradient based on Eq. 6.13 can be rewritten into the simplified form:

$$\mathbb{E}[S_{\text{MAML}}] \approx \text{AvgGradOuter} + 2(\mathcal{T} - 1)\Gamma \cdot \text{AvgGradInner} \tag{6.16}$$

Similarly, the expected FOMAML gradient based on Eq. 6.10 can be rewritten as:

$$\mathbb{E}[S_{\text{FOMAML}}] \approx \text{AvgGradOuter} + (\mathcal{T} - 1)\Gamma \cdot \text{AvgGradInner} \tag{6.17}$$

Comparing Eq. 6.16 and 6.17 shows that FOMAML only sacrifices learning rate. Theoretically, FOMAML may be converging slower than the original MAML, but if $\mathcal{T}$ and Γ are sufficiently small, it does not have a big impact on the final result.

6.3.6 Related work in V2X applications

RL algorithms are increasingly used to optimize performance in V2X applications. Some studies such as [156], [158] and [70] consider the multi-objective nature of V2X applications, they either decompose the objectives into subproblems [156] or convert the problem into a single-objective one through scalarization [158][70]. All of these studies simplify the V2X environment in some regards: [89] and [85] consider only one single objective; [162], [158] and [85] require complete information to centrally solve the optimization problem, but in a dynamic environment, acquiring enough information for a centralized approach is often not feasible or violating user's privacy; [104] and [156] consider only system objectives, not user objectives, and it is assumed that users do not make offloading and resource allocation decisions – this assumption may not apply to the highly individual and customized environment of ITS, where even today, vehicle and mobile users are participating in offloading and resource allocation decisions in the network, motivated only by their individual objectives. References [17], [153], [150] and [70] assume all users are cooperative with common objectives, and if users have multiple objectives, the modeling complexity will increase significantly.

Other approaches to solving MOPs extend equilibria concepts to multi-objective settings [69, 100, 114, 116]. They all assume some degree of cooperation and communication between agents; they therefore differ from the competitive environment setup of this thesis where agents do not share private information. Reference [120] assumes a stationary environment, which is different from the multi-state MDP and dynamic environment setup of this thesis. Reference [26] assumes complete information that is different from the partial information assumption. Other approaches try to find discrete solutions on a Pareto frontier in stationary environments through objective selection [55] or decomposition [83, 137, 149]. However, in a dynamic environment, they do not meet the challenge of huge state and action space, unknown state distributions, and MDP with continuing tasks.

6.4 Problem formulation

The auction formulation below is the same as in Sec.5.3. Notations are in Table 6.2. The difference to the original formulation is in the objectives definition, which will be introduced afterwards.

Let M be the set of vehicles (bidders) and K the set of commodities (service types), each type with, at time t, a total of n_k^t available service slots in computing sites. Bidder m has an initial wealth of B_m^0. Its demand for each service type $k \in K$ at time t is represented by a bid denoted $i_k^t \in I$.

From its bidding strategy π_m, bidder m draws its actions $\alpha_m^t = \{\alpha_{m,k}^t \in \{0, 1\}\}$ and $\mathbf{b}_m^t = \{b_{m,k}^t \in \mathbb{R}_+\}$ for each service type. α is the vector of backoff decisions, $\mathbf{b}$ is the vector of bidding prices. More specifically, bidder m's options for each bid are: 1) back off ($\alpha_{m,k}^t = 0$) with a backoff cost $q_{m,k}^t$, or 2) bid ($\alpha_{m,k}^t = 1$) with price $b_{m,k}^t$. To avoid overbidding, at any time t, $\sum_k \alpha_{m,k}^t b_{m,k}^t \leq B_m^t$.

From bidder m's perspective, the competing bidders (denoted $-m$) draw their actions from a joint strategy distribution π_{-m}^t that is an unknown function of $(\mathbf{p}^1, \cdots, \mathbf{p}^{t-1})$, where $\mathbf{p}^t \in \mathbb{R}_+^{|K|}$ is the vector of final prices at the end of time t. All bidders get the new $\mathbf{p}^t$ on all commodities as feedback from the auctioneer. If bidder m wins its bid i_k^t indicated by bidding outcome $z_{m,k}^t = 1$, it pays p_k^t to the auctioneer. If it loses ($z_{m,k}^t = 0$), it pays 0 to the auctioneer but has a cost associated with losing the bid, denoted by $c_{m,k}^t$. If rebidding is permitted and i_k^t has not reached its deadline, m repeats the decision-making process in $t + 1$. If i_k^t passes the deadline before it is admitted, it is viewed as a lost bid with cost $c_{m,k}^t$.

The auction repeats for $\mathbb{T}$ rounds, in every auction round, bidder m's utility (Sec. 2.2.3) is $u_m^t(\alpha_m^t, \mathbf{b}_m^t, \mathbf{p}^t, \mathbf{z}_m^t, \mathbf{c}_m^t, \mathbf{q}_m^t)$, the utility is added to the wealth pool: $B_m^{t+1} = B_m^t + u_m^t$. If $B_m^{t+1} \leq 0$, bidder m loses all the unfinished bids with cost $c_{m,k}^t$ and is reset.

Next, we formulate the problem with multiple objectives $o \in O$. Bidder m receives a reward vector $\mathbf{r}_m^t \in \mathbb{R}^{|O|}$ in random intervals from its own observation and the feedback signals from the auctioneer for its achievement of these objectives. More details will be provided in the next section. Each bidder's preference vector over the objectives is $\mathbf{W}_m^t \in \mathbb{R}^{|O|}$. Bidder preferences can change over time. In real life, changes in preference can be driven by long-term shifts in societal, legal and personal attitudes, or short-term private prioritization, etc. The bidder's goal is to maximize expected return $\mathcal{J}_m = \frac{1}{\mathbb{T}} \sum_{t=1}^{\mathbb{T}} (\mathbf{W}_m^t)^T \cdot \mathbf{r}_m^t, \mathbb{T} \to \infty$, where $\mathbf{W}_m^t$ is time-variant and unknown *a priori*.

Table 6.2: MOP problem symbol definition

Sym	Description	Sym	Description
$k \in K$	commodity type	n_k	k's availability
$i \in I$	bid/request	$m \in M$	bidder
B	budget	v	valuation
α	backoff decision	b	bidding price
c	cost of losing the bid	q	backoff cost
p	payment	z	bidding outcome
u	immediate utility	U	cumulated utility
r	reward	$\mathbf{W}$	preference vector
$o \in O$	objective	π	bidding strategy

Typical RL techniques learn to maximize reward with a constant preference vector over the multiple objectives. This is essentially one single point on the Pareto frontier of the MOP. The proposed approach in Sec. 6.5 finds the shape of the Pareto frontier, befitting the V2X environment where vehicles have time-variant preference vectors $\mathbf{W}_m^t$ that is unknown *a priori*.

In the following Sec. 6.4.1, the author builds a multi-agent system to simulate the auction mechanism and bidders with multiple objectives.

6.4.1 Modified second-price auction mechanism

Utility function is a generalization of the original definitions in Sec. 4.4.1 and 5.4, specifically of Eq. 5.1. For simplicity, the notation t is omitted in this section.

In each auction round, bidder m has **objective** o_1: maximize immediate auction utility $r_m^{o_1} = u_m$. Objective o_1 is broken down into three sub-objectives. 1) o_{1-1}: maximize payoff $r_m^{(k,o_{1-1})} = \alpha_{m,k} \cdot z_{m,k} \cdot (v_{m,k} - b_k^*)$. 2) o_{1-2}: minimize the chance of being rejected by the auctioneer. The cost of bidding and then losing the bid is $r_m^{(k,o_{1-2})} = -\alpha_{m,k} \cdot (1 - z_{m,k}) \cdot c_{m,k}$. 3) o_{1-3}: minimize backoff time. If backed off, m has cost $r_m^{(k,o_3)} = -(1 - \alpha_{m,k}) \cdot q_{m,k}$.

$$\begin{aligned} u_{m,k} &= r_m^{(k,o_{1-1})} + W_m^{o_{1-2}} r_m^{(k,o_{1-2})} + W_m^{o_{1-3}} r_m^{(k,o_{1-3})} \\ r_m^{o_1} &= u_m = \sum_{k \in I} u_{m,k} \end{aligned} \tag{6.18}$$

The cost terms $c_{m,k}$ and $q_{m,k}$ with preferences $W_m^{o_1-2}$ and $W_m^{o_1-3}$ quantify tradeoff between long backoff time and risky bidding. In this implementation (Sec. 6.6), $c_{m,k} = v_{m,k}$, $q_{m,k}$ is reciprocal to the time-to-deadline, and non-negative weights $W_m^{o_1-2}+W_m^{o_1-3} = 1$. Sec. 6.6.3 shows the proposed algorithm is not sensitive to changes in the hyperparameters $v_{m,k}$ and $q_{m,k}$.

Bidder m's long-term individual **objective** o_2 is to minimize $r_m^{o_2} = \text{OFR}_m \in (0, 1)$ at long intervals with preference $W_m^{o_2}$. Long-term objectives are only available to m at the end of each interval. The short-term system **objective** o_3 of maximizing resource utilization $r_m^{o_3} = \beta$ and the long-term system **objective** o_4 of maximizing fairness $r_m^{o_4} =$ Fairness are the same for all bidders and broadcasted to all. Bidders do not have to respect the system objectives; their preferences are reflected in the values $W_m^{o_3}$ and $W_m^{o_4}$. A preference of 0 means the bidder does not consider system objectives at all. The definitions of OFR, β and Fairness for this implementation are in Sec. 6.6.1.

The reward for objective achievement in each time step is:

$$r_{e,m} = W_m^{o_1} \cdot r_m^{o_1} + W_m^{o_2} \cdot r_m^{o_2} + W_m^{o_3} \cdot r_m^{o_3} + W_m^{o_4} \cdot r_m^{o_4} \tag{6.19}$$

The notation $r_{e,m}$ is for the scalarized *extrinsic reward* (Sec. 6.5.1) specific for the bidder m with preference vector $\mathbf{W}_m^t$ at time t. We let $W_m^{o_1} + W_m^{o_3} = 1$ to balance the short-term objectives, and $W_m^{o_2} + W_m^{o_4} = 1$ to balance the long-term objectives.

Next, the author proposes an algorithm that learns to maximize $r_{e,m}$ over time, with changing $\mathbf{W}_m^t$.

6.5 Proposed solution

To solve the long-term reward maximization problem, the author proposes MOODY: **M**ulti-**O**bjective **O**ptimization through **D**istributed reinforcement learning with dela**Y**ed reward.

In this chapter, the author defines six different short and long term, individual and system objectives, and periodically samples objective weights for each agent to simulate their changing choice and preference of objectives. If the sampled preference weight is 0, the agent does not consider that objective. The technique comprises two parts: the offline training cycle and the online inference/retrain cycles. In the offline training

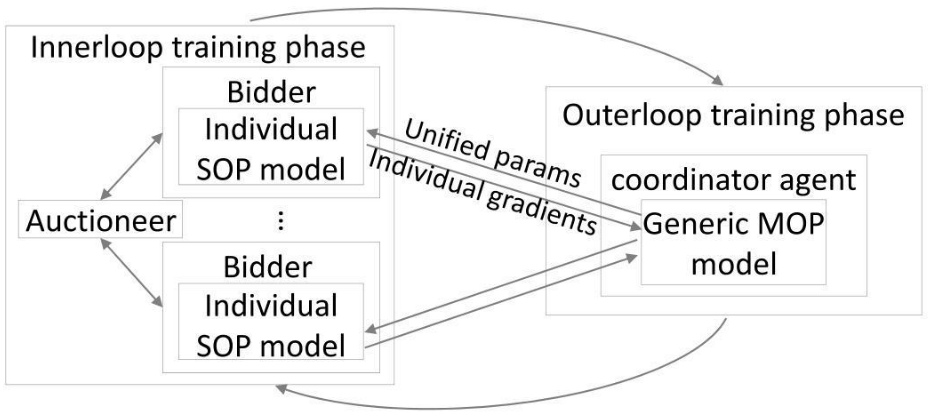


Figure 6.1: Two-phase training

cycle, the approach is further split into two phases: the inner-loop training phase and the outer-loop training phase. In the inner-loop training phase, a local agent trains with a uniform randomly sampled, constant preference vector and tries to find an optimal solution on its Pareto frontier for the given preference vector — with $|M|$ number of agents, the inner-loop phase finds $|M|$ non-dominated solutions on the Pareto frontier. In the outer-loop training phase, one coordinator agent combines all results from inner-loop trainings. The inner-loop and outer-loop training happens alternatively (Fig. 6.1). At the end of the training cycle, we have an initial generic model that, given any new preference vector, can infer an action which leads to a set of rewards that is close to the Pareto frontier. Authors of [52] demonstrate that such a two-phase training method creates a initial generic MOP model that can be easily retrained online for a different task (few-shot learning).

Although reference [52] provides a framework for two-phase multi-task training, it does not suggest a choice of the model, as it does not consider any specific problem or application; due to the complexity of the approach, real-life implementation of their method in a dynamic environment such as V2X is problematic.

In this thesis, the author makes several improvements to the framework: 1) the author designs a specific inner-loop algorithm for the multi-agent application scenario, it outperforms classic RL algorithms such as actor-critic in dynamic environments with sparse and delayed rewards. 2) In the outer loop, the author implements the parallel stochastic gradient ascent method [84] using fully distributed, asynchronous federated learning to increase learning efficiency. 3) The author proposes an adaptive online retraining mechanism that continuously predicts long-term reward; a decreasing prediction accuracy triggers a short, few-shot online retraining cycle. The author therefore extends the framework proposed in [52] that only retrains the model at the beginning of test environment deployment. The proposed approach in this thesis is more adaptive to changing environment and objectives.

Table 6.3: MOP solution symbol definition

Sym	Description	Sym	Description
θ	model parameters	Γ	learning rate
γ	discount rate	δ	TD error
A	action	S	state
V	state value	π	target policy
$\mathcal{J}$	scalarized return	τ	inner-loop training shots
r_e	extrinsic reward	r_i	intrinsic reward
ζ	best response strategy	ψ	behavior strategy
η	best response weight	L_f	state prediction loss
ϕ	state features	ϵ	credit weight for actions

The two-phase training cycle takes place offline with gradient-sharing between the generic model and the local, single models. Otherwise, observation data, hyperparameters for initialization and objective preferences remain private to the local agents with the single models. Once the training cycle is over, the simulation environment is reset to have all local agents initialized with the extensively trained generic model. Then we test them for online inference and retraining without further parameter sharing, in a realistic test environment.

Section 6.5.1 introduces the RL algorithm for the dynamic environment in the inner loop. Section 6.5.2 describes the asynchronous federated learning approach in the outer-loop offline training phase. Section 6.5.3 introduces the proposed adaptive retraining method. Notations are in Table 6.3.

6.5.1 RL in the inner loop

In the inner loop, each bidder (local agent) learns autonomously to maximize its reward. The inner-loop algorithm is based on MALFOY (Sec. 5.4), the author changes it to suit the multi-objective problem, such that it can now learn to optimize multiple short and long-term objectives with a preference vector $\mathbf{W}_m^t \in (0,1)^{|O|}$ that changes over time (in the following simulation, it is drawn from a uniform distribution).

The local, single models have the same structure as in Sec. 5.4 (see Fig. 6.2): 1) a fictitious self-play (FSP) module [60], including an RL with actor-critic and an SL, 2) a curiosity-learning [113] module, and 3) a credit-assignment module.

In the beginning of every inner-loop offline training phase, all local agents are initialized with the same generic model that is the outcome of the previous outer-loop

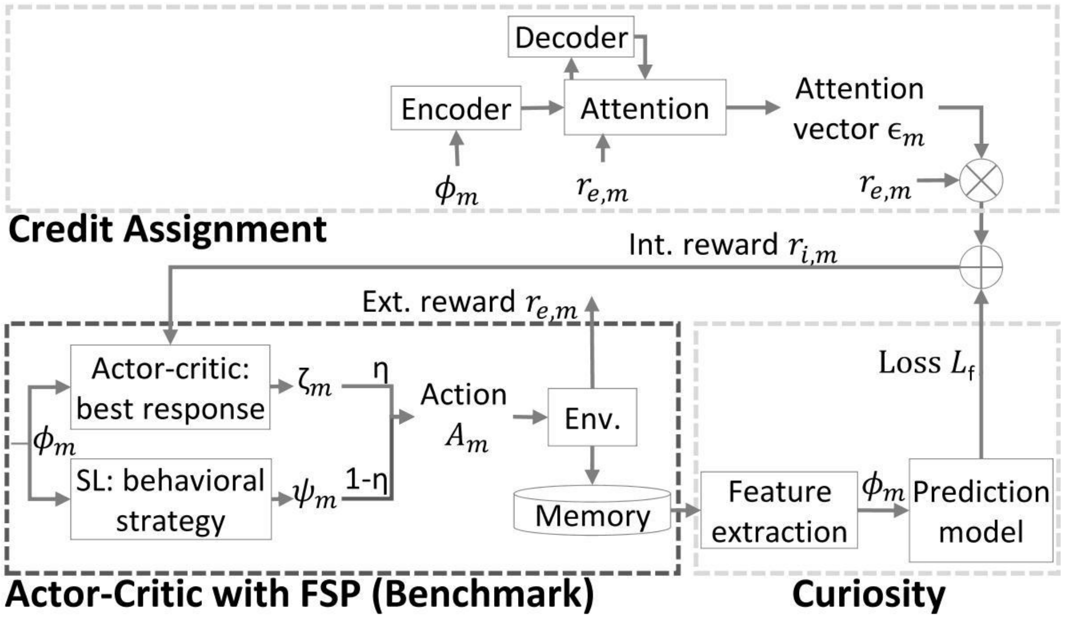


Figure 6.2: Inner loop RL

phase, with the parameters θ^0; during inner-loop training, local agent m trains its own local, single model and does not share parameters or private observations with other agents. At each time step t, it receives extrinsic reward $r^t_{e,m}$, including the long-term rewards if they are available. m's curiosity module predicts next state with prediction loss L^t_{fm}, and the credit assignment module outputs attention vector ϵ^t_m. The resulting intrinsic reward is $r^t_{i,m} = \epsilon^t_m r^t_{e,m} + L^t_{fm}, t \in \{1, \cdots, \tau\}$. The expected return is now $\mathcal{J}_m = \frac{1}{\mathbb{T}} \sum_{t=1}^{\mathbb{T}} r^t_{i,m}, \mathbb{T} \to \infty$. In trying to maximize $\mathcal{J}_m$, the local agent encourages 1) actions that bring higher extrinsic reward, 2) exploration in less visited states with poor prediction accuracy (high L^t_{fm}), and 3) actions that contribute more to the accurate prediction of long-term rewards (high ϵ^t_m) . The update rule for m's individual parameters in the inner-loop offline training phase is [138]:

$$\begin{cases} \theta^t_m \leftarrow \theta^{t-1}_m + \Gamma \delta^{t-1}_m \nabla_{\theta^{t-1}_m} \ln \pi(A|\phi^{t-1}_m, \theta^{t-1}_m) \\ \theta^0_m = \theta^0, \forall t \in \{1, \ldots, \tau\} \end{cases}$$

where $\delta^{t-1}_m = \mathbf{r}^{t-1}_{i,m} + \gamma_m \hat{V}(\phi^t_m, \theta^{t-1}_m) - V(\phi^{t-1}_m, \theta^{t-1}_m)$ is the TD error, Γ_m is the learning rate, and γ_m is the discount rate. In this thesis, action $A = (\alpha^t_m, \mathbf{b}^t_m)$. At the end of τ shots, the local gradients are passed to the coordinator agent before the next outer-loop phase. The inner-loop algorithm is in Alg. 10.

Algorithm 10 Offline innerloop training of local agent

```
1: Initialize $T = 0$, $B^0_m$, $\mathbf{v}_m$, $\Gamma_m$, $\gamma_m$, $\eta_m$, $\tau$.
2: **while** true **do**
3:     $t \leftarrow T + 1$, receive new preferences $\mathbf{W}^t_m$ if available.
4:     Receive $\theta^0$ from coordinator agent, initialize $\theta_m = \theta^0$.
5:     **while** $t \leq T + \tau$ **do**
6:         **Observe and remember:**
7:             Get new service request and add to pipeline at time $t$.
8:             Observe environment variables and past payments.
9:             Retrieve details of all requests in current pipeline.
10:            Create state vector $S^t_m$ and add to memory.
11:            Infer $\phi^t_m$, $L^t_{fm}$ from *curiosity*.
12:            Infer $\epsilon^t_m$ from *credit assignment*.
13:            With $\epsilon^t_m$, update backwards past $r_{i,m}$'s in memory.
14:        **Take action:**
15:            Infer actions $\alpha^t_m$,$\mathbf{b}^t_m$ from *actor-critic RL with FSP*.
16:            Collect all bids with backoff decision $\alpha = 0$:
17:                Calculate backoff cost $\mathbf{q}^t_m$, update $r^t_{e,m}$ in memory.
18:                Add those before deadline to pipeline at $t + 1$.
19:                Drop the rest as lost bids with penalty $\mathbf{c}^t_m$.
20:            Submit bids with $\alpha = 1$, with prices $\mathbf{b}^t_m$.
21:        **Collect rewards:**
22:            Observe bidding results $\mathbf{z}^t_m$ and payments $\mathbf{p}^t$.
23:            Collect all lost bids with $z = 0$:
24:                Calculate penalty $\mathbf{c}^t_m$, update $r^t_{e,m}$ in memory.
25:                Collect requests before deadline and can rebid:
26:                Add those to pipeline at $t + 1$; drop the rest.
27:            Collect all won bids with $z = 1$:
28:                Calculate auction utility, update $r^t_{e,m}$ in memory.
29:                Update $B^t_m$.
30:            Get other ext. rewards, update $r^t_{e,m}$ in memory.
31:            Calculate and add $r^t_{i,m}$ in memory.
32:        **Update learning model:**
33:            Train *actor-critic RL with FSP* and *curiosity*.
34:            Train *credit assignment* if long-term reward available.
35:            Update $\theta_m$ with gradient $\nabla_{\theta^t_m}\mathcal{J}_m = \delta_m \nabla_{\theta_m} \ln \pi_m$.
36:        $t \leftarrow t + 1$
37:    **end while**
38:    Pass $\nabla_{\theta^\tau_m}\mathcal{J}_m$ to coordinator agent.
39:    $T \leftarrow T + \tau$.
40: **end while**
```

6.5.2 Federated learning in the outer loop

While independent local agents with the local models learn for τ shots, the coordinator agent with the generic model waits with the original parameters θ^0, until the next update in the outer-loop phase (Fig.6.1). In the outer-loop phase, the goal of the coordinator agent is to maximize all local agents' sum of returns: $\mathcal{J} = \sum_m \mathcal{J}_m(\theta_m^\tau)$. After τ shots, at the end of the previous inner-loop phase, the generic model's parameters are θ^0, and it uses the local models' gradients to update its parameters: $\theta^{0'} = \theta^0 + \Gamma \nabla_{\theta^0} \mathcal{J}$. Since each individually updated parameter $\theta_m^t, \forall t \in \{1, \dots, \tau\}$ is a function of θ^0, using chain rule, the generic model's parameter update is:

$$\theta^{0'} = \theta^0 + \sum_m \left(\nabla_{\theta_m^\tau} \mathcal{J}_m(\theta_m^\tau) \prod_{t=1}^{\tau-1} \left(\mathcal{I} - \Gamma_m \nabla^2_{\theta_m^t} \mathcal{J}_m(\theta_m^t) \right) \right)$$

where $\mathcal{I}$ is the identity matrix. Although it is computationally expensive, it can be approximated by a first-order derivative with the assumption that both Γ and τ are small [52, 103]. $\theta^{0'} = \theta^0 + \sum_m \Gamma \delta_m^\tau \nabla_{\theta_m^\tau} \ln \pi_m^\tau(\theta_m^\tau)$ is the simplified update rule. The setup in this thesis meets the assumptions with $\Gamma = 0.1$ and $\tau = 3$.

The author uses asynchronous federated learning to implement the parallel stochastic gradient ascent method (Sec. 6.7). It does not require all local models to be trained and updated at the same time: each model is trained based on the availability of new local data. Whenever the local model finishes training for τ shots, the local agent transmits the gradients to the coordinator agent and gets updated model parameters from it. This reduces data rate needed for gradient and parameter communication and further increases learning efficiency.

6.5.3 Adaptive online retraining

After the offline training, and once the model is deployed in a real-world setting, the credit-assignment module continuously predicts rewards. The current reward prediction accuracy is compared to the moving average of past N prediction accuracies, if it falls below the past average, a short n-shot retraining cycle is triggered. In the simulation in this chapter, the author uses $N = 10$ and $n = 1$. The algorithm is described in Alg. 11.

Algorithm 11 Online adaptive retraining of local agent

```
1: Initialize $t = 0$, $B_m^0$, $\mathbf{v}_m$, $\Gamma_m$, $\gamma_m$, $\eta_m$, $\tau$, and moving average period $N$ of *credit assignment*'s prediction loss.
2: Initialize with $\theta$ from coordinator agent.
3: while true do
4:     $t \leftarrow t + 1$, receive new preferences $\mathbf{W}_m^t$ if available.
5:     Observe and remember.
6:     Take action.
7:     Collect rewards.
8:     if long-term reward is available then
9:         Calculate and store prediction loss of *credit assignment*.
10:        if current prediction loss ¿ past $N$ average then
11:            Update learning model.
12:        end if
13:    end if
14: end while
```

Simulation results in Sec. 6.6.2 show the effectiveness of this adaptive online retraining approach. Sec. 6.7 mentions practical considerations in online retraining.

6.6 Evaluation

6.6.1 Simulation setup

The evaluation is done on both cycles: the offline two-phase training, and the online testing / retraining. The coordinator agent with the generic model is only present in the training cycle, it collects gradients from all local agents and learns a generic model. Once deployed in the test environment, all agents are initialized with the same generic model, but then diverge from it by adapting to the environment through online retraining. All agents are independent bidders with private observations and model parameters that are not shared with any other agent. In both cycles, the author considers a V2X system as defined in Sec. 3.2: vehicles are bidders who request networking services (commodities); road-side unit or base station acts as auctioneer that controls admission of service requests and assigns them to different computing sites (commodity sellers), which own resources and execute services.

The algorithm is trained and evaluated in a similar environment as described in Table 3.1

of Chapter 3, with the only difference that there are now multiple reward signals from the auctioneer; also, the test environment has only half of the resource capacity as in the previous chapter, creating a much higher contention situation.

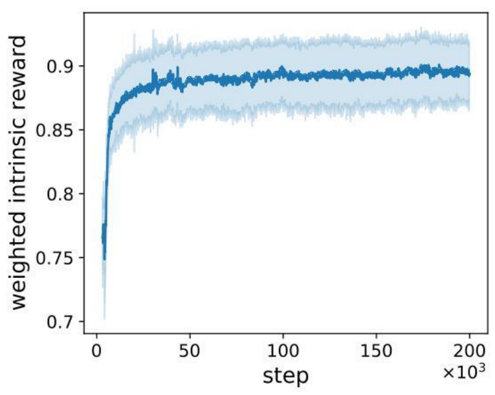
To evaluate the long-term multi-objective learning algorithm, the performance metrics in Sec. 3.4 is used to monitor how the bidders perform on the following objectives:

- Maximize individual short-term (immediate) auction utility: as defined in Eq. 6.18.
- Minimize system short-term resource utilization variation: load-balancing effect is achieved by encouraging bidding at time of low system utilization [140]. Short-term resource utilization is the ratio of resources effectively utilized at computing sites at the time of ACA admission.
- Minimize long-term individual offloading failure rate (OFR): average ratio of offloading requests rejected by ACA during admission control. In fact, OFR should include all failed service executions at computing sites until deadline, rather than only those rejected by the ACA. However, this means feedback of bidding result to the bidders is delayed, and the length of delay is specific to each service request. As a simplification, the author uses rejection rate as a proxy to OFR. This is justified by the fact that the system responsiveness (i.e., the ratio of successfully executed requests to all accepted requests) is ca. 99%.
- Maximize long-term system fairness: the author uses J-index [68] of payments over the last $\mathcal{T}$ time steps: Fairness $= \frac{(\sum_m \sum_{t-\mathcal{T}}^{t} p_m^t)^2}{|M| \sum_m (\sum_{t-\mathcal{T}} t p_m^t)^2}, \forall m \in M.$

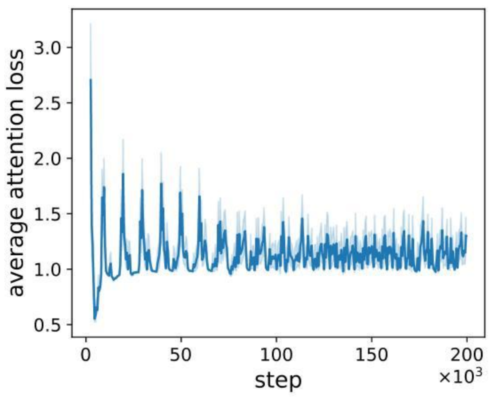
The two short-term rewards on auction utility and resource utilization are available immediately after the auction round. The two long-term rewards on offloading failure rate (OFR) and fairness are available only after a 2000-time-step delay.

The same simulator is used for both offline training and online testing. However, there are significant differences to the environment setups. In the training environment, besides bidders and the auctioneer, there is a coordinator agent that is only active during the outer-loop training phase to learn parameters for a generic model (Sec. 6.5.2). The generic model is incrementally updated and used to initialize all local agents at the beginning of each inner-loop training phase. During every inner-loop, each bidder randomly selects a preference vector for the objectives and acts independently.

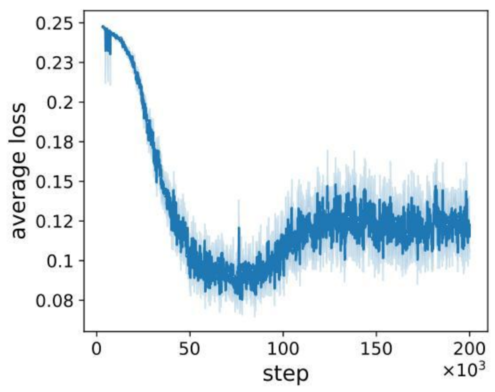
In the test environment, there is no coordinator agent, the bidders are initialized with the generic MOODY model in the beginning of the simulation, and they uniform randomly select a preference vector with values in [0, 1] whenever they get in range of the



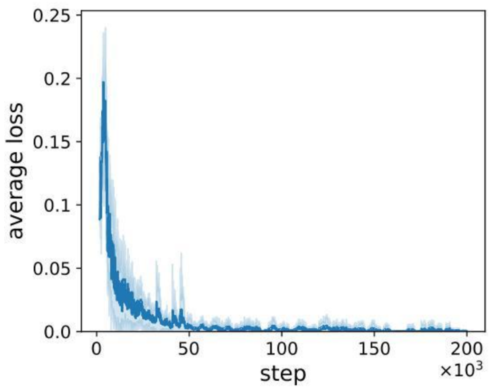
(a) Average RL reward of all bidders increases over time.



(b) Average loss in credit assignment module decreases over time.



(c) Average loss in curiosity module's forward submodule decreases over time.



(d) Average loss in curiosity module's inverse submodule decreases over time.

Figure 6.3: Training results

RSU, which depends on the vehicle arrival rate, the speed and the traffic light phases of the simulator (Table 3.1). Throughout evaluation, their credit assignment modules continuously predict rewards and trigger a short, adaptive retraining cycle according to Sec. 6.5.3. Besides these setup differences, the test environment also differs significantly from the training environment in resource capacity, vehicle arrival rate and speed, and traffic light phases. Table 3.1 summarizes the differences.

6.6.2 Realistic setup results

Fig. 6.3 shows how all modules of **MOODY** converge to a local optimum in the two-phase training cycle. In the low-contention setup of the training environment, we reach close to optimal long-term objectives (i.e., OFR close to 0 and fairness close to 1).

In the following test/retrain cycles, the author compares the performance of 1) **MOODY** bidders initialized with the generic model for multiple objectives, 2) **MALFOY** bidders with the state-of-the-art single-objective algorithm from [139], pretrained independently with a scalarized objective, 3) benchmark **AC** bidders with only the actor-critic module. The tests are run separately, each test has only one algorithm for all bidders in the simulation and run multiple times. The confidence intervals are reported across all runs. In all tests, bidders' preference vectors change randomly over time, drawn from a uniform distribution.

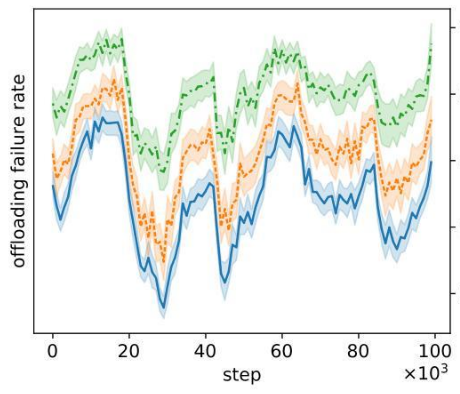
During testing, each MOODY bidder decides independently whether to trigger a retraining cycle. In this simulation, once retraining is triggered, the modules learn with 1 shot in each retraining cycle. The MALFOY bidders are retrained for a fixed 10k time steps (i.e., simulated milliseconds) at the beginning of the deployment in the test environment. The AC bidders are not retrained.

Fig. 6.4a and 6.4b compare performance on the achievement of system long-term fairness and individual long-term OFR. In fact, in all objectives, MOODY outperforms other bidders: MOODY's fairness score is close to 1, compared to MALFOY's 0.89 and AC's 0.86; MOODY achieves 46% higher utility than MALFOY and 77% higher than AC; MOODY also achieves 16-30% lower offloading failure rate with 5-14% less system utilization. Although the average utilizations with MOODY and MALFOY bidders are similar, MOODY lowers load variation by 19% compared to MALFOY.

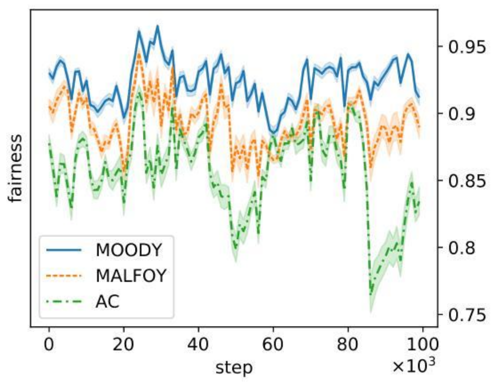
Fig. 6.4c shows an example of how retraining contributes to the decrease in prediction loss for one of the bidders: the retraining cycles are triggered by low reward prediction accuracy. The vertical gray lines are where retraining cycles occur. The bidder triggers a one-shot retraining cycle whenever the prediction accuracy of rewards reduces to below the moving average of the past 10 prediction accuracies. As shown in Sec. 6.4c, the retraining cycles are frequently, almost continuously triggered in the beginning of the deployment in test environment. Overall, the MOODY bidders spend 9-15% of time in retraining cycles, with an average of 12%.

Fig. 6.4d shows correlation between achievements of the two long-term objectives: improvement in fairness is correlated to reduction in failure rate. In fact, we also see such correlation between other objectives. The reward signals on the system objectives help bidders learn this correlation, and by considering system objectives, the bidders effectively earn higher reward on their individual objectives, at the same time the auctioneer and commodity sellers achieve their objectives through incentivization.

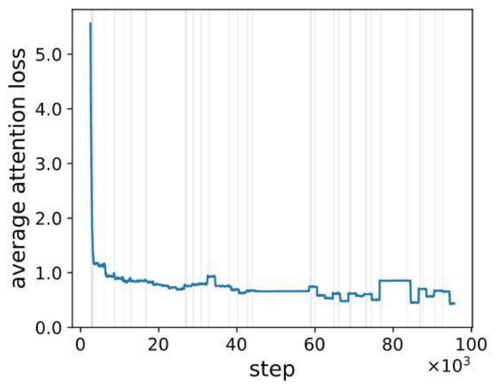
All of the evaluations in Fig. 6.4 are done with the same type of algorithms in the test environment (i.e., "homogeneous"). However, in real life, drivers may run different



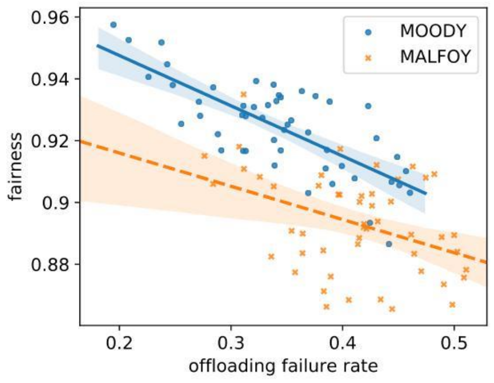
(a) MOODY reduces long-term individual OFR by 16% compared to MALFOY, 30% compared to AC.



(b) MOODY achieves average fairness of 0.92, compared to MALFOY: 0.89 and AC: 0.86.



(c) MOODY bidders' retrain cycles are 12% of the time (gray lines are retrain cycles).



(d) Achievement of higher system fairness is correlated to the achievement of lower individual OFR.

Figure 6.4: Objective achievement in test and retraining cycles

algorithms (i.e., "heterogeneous"). Fig. 6.5 shows the cumulative distribution function (CDF) of each bidder's OFR performance, when the two algorithms compete in the same environment.

The blue solid line labeled "MOODY" shows the performance of MOODY bidders in either the homogeneous (all-MOODY) or the heterogeneous environments with different percentage of MOODY bidders–the average performance among vehicles with MOODY algorithm in all environments are hardly different. In other words, they are unimpacted by the existence of other algorithms. Hence, to simplify the figure, we show their performance in one single curve. The dotted lines show performance in environments with different mix of MOODY and MALFOY bidders. The rightmost line shows average performance of MALFOY bidders in a homogeneous, all-MALFOY en-

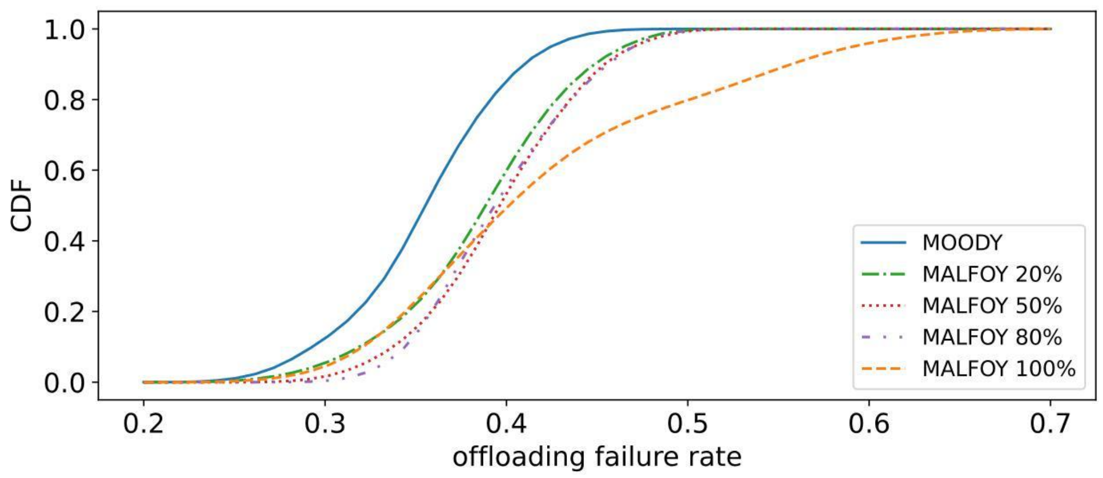


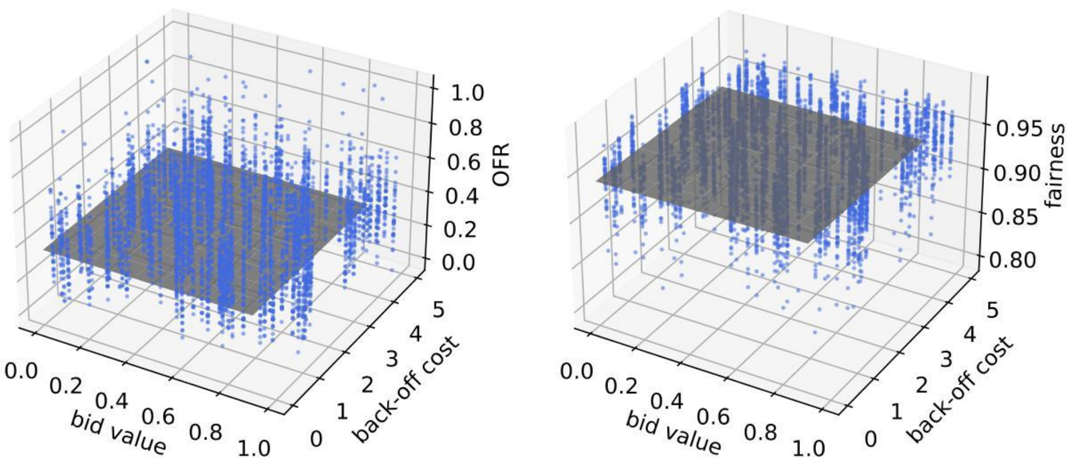
Figure 6.5: In a heterogeneous test environment with competing algorithms, MALFOY performance improves with MOODY unimpacted. System fairness also improves.

vironment, which has the worst performance of all environments. Interestingly, in all of the heterogeneous environments, MALFOY bidders' OFR performance improved, compared to the all-MALFOY environment, reducing the difference to MOODY bidders by 50%. Overall system fairness also improved significantly. These improvements do not depend on the percentage of MOODY bidders in the environment, indicating that even the presence of very few MOODY bidders can enhance overall performance.

To summarize: the author tests MOODY's transfer learning capability by evaluating its performance in more dynamic test environments and allowing the bidders to change their objective preferences. Evaluation results show that 1) bidders initialized with MOODY and adaptively retrained outperform bidders with other state-of-the-art learning algorithms in all objectives; 2) the MOODY bidders demonstrate good generalization and transfer learning property, adapting to preference changes and dynamicity in the environment; 3) the presence of MOODY bidders in the environment improves the performance of bidders with other algorithms and system overall fairness.

6.6.3 Sensitivity analysis

Similar to the sensitivity analysis we did with MALFOY in Fig. 5.8: the same analysis is done with MOODY. Results in Fig. 6.6 show that MOODY is a robust algorithm that is insensitive to hyperparameter changes. Both OFR and fairness performance are insensitive to changes to bid valuation $v_{m,k}$ and backoff cost $q_{m,k}$.



(a) Individual OFR (b) System fairnness

Figure 6.6: Sensitivity analysis shows target achievement is not sensitive to changes in hyperparameters such as bid value and backoff cost.

6.7 Practicality considerations

To speed up the training process, the author trains each module asynchronously in federated learning (Sec. 6.5.2). Before each inner loop begins, the local agent is initialized with generic model parameters. Then, the agent joins the auction whenever it receives a request. Since each local agent receives requests randomly and makes independent decisions, they finish the inner-loop training phase at different time steps. Furthermore, each module trains at different time intervals, depending on their input and batch size. In this chapter, whenever the coordinator agent receives new gradients from any local agent for any of the modules, it trains the module and updates the parameters. The asynchronous training approach reduces maximum data rate for gradient transmission and training time: the time for training once (i.e., one-shot) is the **maximum** duration among the modules 1) RL with credit assignment (*RL+credit*), 2) supervised learning (*supervised*) and 3) curiosity learning (*curiosity*). Without the asynchronous training, one-shot training time would be the **sum** of all modules. More importantly, asynchronous training reduces the online retraining time after deployment.

After deployment, bidders decide independently when to retrain the model to adapt to new objectives and environments (Sec. 6.5.3). The retraining is easily separated from the main program that infers bidding decisions in runtime (i.e., *out-of-critical-path*). With retraining off the critical path, we can ensure fast decision-making even with retraining.

Table 6.4: Performance test

Modules	Training		Inference	
	Nr.calls	Per call (millisec)	Nr.calls	Per call (millisec)
RL+credit	108	5484	431	29
supervised	108	112	431	0
curiosity	197	3092	431	0
data prep	1275	10	431	29
Time per shot(tested with Nano)	max. of async. trained modules +data prep:	**5494**	sum of all modules + data prep:	**58**
Time per shot(estim. AGX Orin)	$1/10^{th}$ of Nano:	**550**	$1/10^{th}$ of Nano:	**6**

6.7.1 Test on a single-board computer

The author tests real-life training and inference speed of the proposed algorithm on an Nvidia Jetson Nano single-board computer with GPU. This represents the OBU of a vehicle, hence, a bidder. The author runs the training and inference repeatedly and records the average time for one shot. The results are shown in Table 6.4. As mentioned previously, time for one-shot training is the maximum time among all modules; time for each inference is the sum of all modules, plus time for data preparation. The author provides the actually measured performance on Nano and an estimated performance on the most up-to-date AGX Orin. Although the theoretical performance difference between the two is more than 100 times, multiple benchmark tests on various AI applications show a more realistic performance difference of approximately 10 times (see Nvidia website for Jetson modules technical specifications and benchmarks). The author therefore estimates that with AGX Orin, training one shot takes ca. 550ms, and every inference takes ca. 6ms. Speed can be further increased through fewer neural network layers and number of nodes, smaller batch size, shorter input length, etc. The impact on performance needs to be analyzed in detail in future work.

These results show that despite the complexity of the proposed solution, bidders can perform runtime inference, on current hardware, with a reaction time of 6ms. V2X applications (e.g., segmentation, motion planning) typically run on the time scale of seconds, an inference speed in milliseconds makes the proposed algorithm a good candidate for real-life deployment. The retraining cycle is longer, for which the author believes that *out-of-critical-path* few-shot retraining holds great promise. Even without that optimization, the retraining cycle lasts only a few seconds, well below the frequency of changes in a V2X environment that may trigger retraining.

6.8 Conclusion of the chapter

The author combines offline federated learning and online few-shot learning to solve an MOP in a dynamic environment. Through extensive offline training, we get an optimal initial model that learns the best initialization point. From this point, it can quickly find a solution on the Pareto frontier, even without retraining, when the agent's objectives change. Only in a significantly different environment, the author allows each bidding agent adaptive, online, few-shot retraining to customize its model, needing very few data points.

The author shows empirically that the new multi-objective algorithm outperforms the benchmark algorithms in all objectives. Furthermore, the proposed algorithm increases bottom-line resource efficiency, such that other algorithms in the environment also benefit from improved offloading success rate and fairness.

The proposed algorithm can be easily modularized, each module trained separately and asynchronously. Coupled with the adaptive few-shot online training method, the algorithm is a very good candidate for real-life deployment.

Simulation results show that bidders learn the correlation between different objectives. In fact, multiple objectives in real-life are typically correlated to each other. There may exist a hierarchy or network of objectives, and we should guide the learning process with this knowledge of the objective structure in future work.

Chapter 7

Conclusion of the thesis

7.1 A summary

The thesis studied the optimal choices agents can make in a distributed decision-making scenario, using resource allocation in V2X as an example application. DRACO, the core MARL+FSP short-term, single-objective algorithm, showed how the agents learned to best trade off backoff time and bidding price; in the process, they managed to compensate for disadvantages in initial parameterization. Further sensitivity analysis in the subsequent chapters also showed the algorithm's robustness in both individual and system objective achievements, when various initial parameters changed, including initial budget, bid valuation and backoff cost. As a result, the agents performed significantly better in very different environments. The author showed that private and system objectives could be aligned without sacrificing either user autonomy or system-wide resource efficiency, despite the distributed design with limited information sharing.

Based on DRACO and with the help of 1) a feature extraction submodule for generalization, 2) an attention layer for long-term credit assignment and 3) a curiosity module for sparse reward signals, the author developed MALFOY. Agents using this long-term, single-objective algorithm behaved more aggressively and selfishly when competing against other agents when the long-term goal was to maximize cumulated private payoff. However, this selfish behavior had a negative impact on overall social welfare. The effect could be corrected when the service provider broadcasted system reward signals to users, such as system fairness and resource utilization. In the subsequent chapter, the author demonstrated that although users could choose to ignore system reward signals, it was beneficial for users to utilize the signal to achieve their individual objectives.

The results implied that by designing the appropriate system reward signals, the service providers could incentivize users to willingly consider system objectives. Empirical results showed that agents utilized these reward signals and had enhanced predictive power as well as better alignment between objectives; MALFOY further improved performance over DRACO.

In MOODY, the third proposed algorithm, the author combined offline federated learning and online few-shot learning to solve a long-term multi-objective problem in a dynamic environment. Through extensive offline training, the author got an optimal initial model for solving different multi-objective decision-making problems; this initial model found the best initialization point from where it could quickly find a local optimal on the Pareto frontier, among the objectives of low offloading failure rate, high system fairness, high short-term utility and high system resource utilization, even without retraining when the agent's objectives changed. Only once the environment had changed significantly and the agent's prediction accuracy of rewards decreased, the author allowed each bidding agent few-shot online training to customize its model, needing very few data points. The author showed empirically that the new multi-objective algorithm outperformed the benchmark algorithms in all objectives.

The proposed solution combined various methods to improve different aspects of the algorithm: 1) to balance between exploitation and exploration, the author let the algorithm learn both the best response and the behavioral strategies through FSP, encouraged exploration of less visited states through the forward submodule of curiosity learning, and used a replay buffer. 2) To further improve data efficiency, the author extracted features from data of different time lengths, using a convolutional neural network, and guided by the agent's actions (i.e., supervised learning through the inverse module of curiosity learning). 3) The breakdown of long-term objectives through the attentional network gave the multiple objectives a structure; it also provided a way to predict delayed rewards and assess the need to retrain in runtime. An analysis of the modules showed that the different modules worked together also asynchronously, and their positive effects were compounded.

7.2 Conclusions

In this thesis, it was curious to see how a self-interested agent learned to game the system to maximize its gain, using the very limited means available to it — the option to back off, and the possibility to prioritize its tasks through a hypothetical bidding price. Even as the author restricted the agent's knowledge of the system, both in quantity and in quality, increased environment dynamicity, introduced inequality at initialization, added multiple objectives and changing preferences over time, the learning algorithm was remarkably robust.

All of the proposed algorithms generalized to very different, previously unseen environments. Each user in the network had its own, constant-size model, and all shared information for modeling was of constant size as well. The distributed nature meant it was easily scalable to huge number of users without increased complexity. The proposed final model could be modularized, each module could be trained separately and asynchronously. Furthermore, the algorithm could assess the need to retrain and update the model with only a few shots in runtime (adaptive online retraining). Therefore, the proposed solution was applicable to a wide range of practical, distributed resource allocation problems — it could learn in a dynamic and adversarial environment with no or very limited *a priori* information, many autonomous users with private objectives and many custom service requests, and it required little maintenance after deployment. The author found such applications in e.g., telecommunications, energy, Internet of Things, vehicular networks, cloud computing, etc.

7.3 Future works

There are many potential future topics following this work. In this thesis, the author used a combinatorial single auction mechanism in which multiple bidders on the user side interacted with a single auctioneer on the operating side. In a completely distributed, ad-hoc network, every user can offer its computation resources as a services provider — a double auction mechanism between users, eliminating the need for a central auctioneer, would provide more flexibility and efficiency for the resource allocation. Previous studies such as [66, 124, 136] proposed double auction mechanisms and optimization algorithms in various networking resource allocation applications. To extend future work in this direction, the author needs to extend the application scenario, analyze the game theoretic properties of the proposed algorithm in a double auction, and simulate a more complex interaction mechanism.

This thesis simulated agents' preference of objectives with uniform randomly generated weights, and scalarized the rewards with a linear objective function, with the assumption that the individual objectives are independent from each other. There are two potential improvements to this approach:1) the method for sampling preferences may impact the approximation of the Pareto frontier and the performance of the initial model. Future work should consider different sampling methods such as proposed in [123] and [75]. 2) Simulation results showed that agents learned the correlation between different objectives. In fact, multiple objectives in real-life are typically correlated to each other.

It can be beneficial to consider hierarchical objectives, for example, in hierarchical RL (HRL), different value functions are learned at different temporal scales [19, 76].

Curriculum learning [21] can also improve the multi-objective algorithm's performance and generalization properties — if we let the algorithm start with learning one objective, then we increase the number of objectives (i.e. "complexity") gradually in the learning process. Concretely, in future work, the author can first train the FSP module independently with short-term rewards, then add the other modules and the long-term reward signals incrementally.

Current work limited information sharing between agents, the setup befitted applications with privacy requirements or sparse signaling; by allowing more information sharing between the agents, or even sharing of experiences, the algorithms can be modified for more cooperative scenarios such as robotics. Authors of [41] suggested the sharing of experiences between similar agents: these agents shared some of the model parameters (e.g. in the critic) but had individual states and actions.

Bibliography

[1] Draco source code. `https://github.com/DRACOsource/draco`.

[2] Malfoy source code. `https://github.com/DRACOsource/malfoy`.

[3] Moody source code. `https://github.com/moodysourcecode/moody`.

[4] C-v2x use cases: Methodology, examples and service level requirements. *5GAA Automotive Association*, 2019.

[5] Iso 20078:2019 road vehicles-extended vehicle (exve) web services. *International Organization for Standardization*, 2019.

[6] Service-based architecture in 5g: case study and deployment recommendations, 2019.

[7] C-v2x use cases volume ii: Examples and service level requirements. *5GAA Automotive Association*, 2020.

[8] Satyam Agarwal, Francesco Malandrino, Carla-Fabiana Chiasserini, and Swades De. Joint vnf placement and cpu allocation in 5g. In *IEEE INFOCOM*, 2018.

[9] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*, 2018.

[10] Mahmoud Almasri, Ali Mansour, Christophe Moy, Ammar Assoum, Denis Le Jeune, and Christophe Osswald. Dynamic decision-making process in the opportunistic spectrum access. *Advances in Science, Technology and Engineering Systems Journal*, 2020.

[11] Qamrul Hasan Ansari, Elisabeth Köbis, and Jen-Chih Yao. Vector variational inequalities and vector optimization. *Cham: Springer International Publishing AG*, 2018.

[12] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*, 2018.

[13] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 2002.

[14] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 2004.

[15] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vini-

cius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Meireles Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 2020.

[16] Sabur Baidya, Yu-Jen Ku, Hengyu Zhao, Jishen Zhao, and Sujit Dey. Vehicular and edge computing for emerging connected and autonomous vehicle applications. In *ACM/IEEE DAC*, 2020.

[17] Rojeena Bajracharya, Rakesh Shrestha, Syed Ali Hassan, Kostromitin Konstantin, and Haejoon Jung. Dynamic pricing for intelligent transportation system in the 6g unlicensed band. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[18] Bikramjit Banerjee and Jing Peng. Performance bounded reinforcement learning in strategic interactions. In *AAAI*, 2004.

[19] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 2003.

[20] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *SIMUL*, 2011.

[21] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[22] Carlos J. Bernardos and Mikko A. Uusitalo. European vision for the 6g network ecosystem. *The 5G Infrastructure Association*, 2021.

[23] Kshipra Bhawalkar and Tim Roughgarden. Welfare guarantees for combinatorial auctions with item bidding. In *ACM SIAM*, 2011.

[24] Marcel Blöcher, Ramin Khalili, Lin Wang, and Patrick Eugster. Letting off steam: Distributed runtime traffic scheduling for service function chaining. In *IEEE INFOCOM*, 2020.

[25] Giacomo Bonanno. Game theory. 2018.

[26] Alexandra Bousia, Elli Kartsakli, Angelos Antonopoulos, Luis Alonso, and Christos Verikoukis. Multiobjective auction-based switching-off scheme in heterogeneous networks: To bid or not to bid? *IEEE Transactions on Vehicular Technology*, 2016.

[27] Michael Bowling. Convergence and no-regret in multiagent learning. In *NeurIPS*, 2005.

[28] Michael Bowling and Manuela Veloso. An analysis of stochastic game theory for multiagent reinforcement learning. Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2000.

[29] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 2002.

[30] Alberto Broggi, Pietro Cerri, Stefano Debattisti, Maria Chiara Laghi, Paolo Medici, Matteo Panciroli, and Antonio Prioletti. Proud-public road urban driverless test: Architecture and results. In *IEEE Intelligent Vehicles Symposium Proceedings*, 2014.

[31] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

[32] Yang Cai and Christos Papadimitriou. Simultaneous bayesian auctions and computational complexity. In *ACM Conference on Economics and Computation*, 2014.

[33] Federico Cali, Marco Conti, and Enrico Gregori. Ieee 802.11 protocol: design and performance evaluation of an adaptive backoff mechanism. *IEEE JSAC*, 2000.

[34] Valeria Cardellini, Vittoria De Nitto Personé, Valerio Di Valerio, Francisco Facchinei, Vincenzo Grassi, Francesco Lo Presti, and Veronica Piccialli. A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*, 2016.

[35] Bi-ke Chen, Chen Gong, and Jian Yang. Importance-aware semantic segmentation for autonomous driving system. In *IJCAI*, 2017.

[36] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE JSAC*, 2018.

[37] Xu Chen. Decentralized computation offloading game for mobile cloud computing. *IEEE Trans. on Parallel and Distributed Systems*, 2014.

[38] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. on Networking*, 2015.

[39] Jin-Hee Cho, Yating Wang, Ray Chen, Kevin S Chan, and Ananthram Swami. A survey on modeling and optimizing multi-objective systems. *IEEE Communications Surveys & Tutorials*, 2017.

[40] Sukjin Choo, Joonwoo Kim, and Sangheon Pack. Optimal task offloading and resource allocation in software-defined vehicular edge computing. In *IEEE ICTC*, 2018.

[41] Filippos Christianos, Georgios Papoudakis, Muhammad A Rahman, and Stefano V Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *ICML*, 2021.

[42] Laurene Claussmann, Marc Revilloud, Dominique Gruyer, and Sébastien Glaser. A review of motion planning for highway autonomous driving. *IEEE Trans. on Intelligent Transp. Systems*, 2019.

[43] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[44] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, pages 1–50, 2021.

[45] Stéphane Durand and Bruno Gaujal. Complexity and optimality of the best response algorithm in random potential games. In *International Symposium on Algorithmic Game Theory*. Springer, 2016.

[46] Prajit K Dutta and Tapan Mitra. Maximum theorems for convex structures with an application to the theory of optimal intertemporal allocation. *Journal of Mathematical Economics*, 1989.

[47] Liran Einav, Chiara Farronato, Jonathan Levin, and Neel Sundaresan. Auctions versus

posted prices in online markets. *Journal of Political Economy*, 2018.

[48] Joan Feigenbaum, Michael Schapira, and Scott Shenker. Distributed algorithmic mechanism design. In *Algorithmic Game Theory*. Cambridge University Press, 2007.

[49] Michal Feldman, Hu Fu, Nick Gravin, and Brendan Lucier. Simultaneous auctions are (almost) efficient. In *ACM Symposium on Theory of Computing*, 2013.

[50] Donald F Ferguson, Christos Nikolaou, Jakka Sairamesh, and Yechiam Yemini. Economic models for allocating resources in computer systems. *Market-based control: a paradigm for distributed resource allocation*, 1996.

[51] Johan Ferret, Raphael Marinier, Matthieu Geist, and Olivier Pietquin. Self-attentional credit assignment for transfer in reinforcement learning. In *IJCAI*, 2020.

[52] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[53] Bidding game source code. `https://github.com/DRACOsource/biddinggame`, 2021.

[54] Saul Gass and Thomas Saaty. The computational algorithm for the parametric objective function. *Naval research logistics quarterly*, 2(1-2):39–45, 1955.

[55] Hend Gedawy, Karim Habak, Khaled A. Harras, and Mounir Hamdi. Ramos: A resource-aware multi-objective system for edge computing. *IEEE Transactions on Mobile Computing*, 2021.

[56] Yacov Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3):296–297, 1971.

[57] Zhu Han, Dusit Niyato, Walid Saad, Tamer Başar, and Are Hjørungnes. *Game theory in wireless and communication networks: theory, models, and applications*. Cambridge university press, 2012.

[58] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 2000.

[59] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):1–59, 2022.

[60] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, 2015.

[61] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[62] Todd Hester and Peter Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 2013.

[63] Markus Hofmarcher, Thomas Unterthiner, José Arjona-Medina, Günter Klambauer, Sepp Hochreiter, and Bernhard Nessler. Visual scene understanding for autonomous driving

using semantic segmentation. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019.

[64] He Huang and Robert J Kauffman. On the design of sponsored keyword advertising slot auctions: An analysis of a generalized second-price auction approach. *Electronic Commerce Research and Applications*, 2011.

[65] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 2019.

[66] George Iosifidis and Iordanis Koutsopoulos. Double auction mechanisms for resource allocation in autonomous networks. *IEEE Journal on Selected Areas in Communications*, 2009.

[67] Amir Jafari, Amy Greenwald, David Gondek, and Gunes Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *ICML*, 2001.

[68] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory*, 1984.

[69] Catholijn Jonker, Reyhan Aydogan, Tim Baarslag, Katsuhide Fujita, Takayuki Ito, and Koen Hindriks. Automated negotiating agents competition (anac). In *AAAI*, 2017.

[70] Ying Ju, Yuchao Chen, Zhiwei Cao, Lei Liu, Qingqi Pei, Ming Xiao, Kaoru Ota, Mianxiong Dong, and Victor CM Leung. Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[71] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 2005.

[72] Mert Kayaalp, Stefan Vlaski, and Ali H Sayed. Dif-maml: Decentralized multi-agent meta-learning. *IEEE Open Journal of Signal Processing*, 2022.

[73] Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. In *NeurIPS*, 2018.

[74] Mehrdad Khaledi and Alhussein A Abouzeid. Optimal bidding in repeated wireless spectrum auctions with budget constraints. In *IEEE GLOBECOM*, 2016.

[75] Esmaile Khorram, Kazhal Khaledian, and Mehrdad Khaledyan. A numerical method for constructing the pareto front of multi-objective optimization problems. *Journal of Computational and Applied Mathematics*, 2014.

[76] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. In *NeurIPS*, 2016.

[77] Neeraj Kumar, Sudip Misra, Joel JPC Rodrigues, and Mohammad S Obaidat. Coalition games for spatio-temporal big data in internet of vehicles environment: a comparative analysis. *IEEE IoT Journal*, 2015.

[78] Neeraj Kumar, Joel JPC Rodrigues, and Naveen Chilamkurti. Bayesian coalition game as-a-service for content distribution in internet of vehicles. *IEEE IoT Journal*, 2014.

[79] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Trans. on Networking*, 2018.

[80] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, 2017.

[81] David S. Leslie and E.J. Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 2006.

[82] Lingxiang Li, Marie Siew, and Tony QS Quek. Learning-based pricing for privacy-preserving job offloading in mobile edge computing. In *IEEE ICASSP*, 2019.

[83] Zhongguo Li and Zhengtao Ding. Distributed multiobjective optimization for network resource allocation of multiagent systems. *IEEE Transactions on Cybernetics*, 2021.

[84] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *ICML*, 2018.

[85] Lei Liu, Jie Feng, Xuanyu Mu, Qingqi Pei, Dapeng Lan, and Ming Xiao. Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[86] George Loukas, Yongpil Yoon, Georgia Sakellari, Tuan Vuong, and Ryan Heartfield. Computation offloading of a vehicle's continuous intrusion detection workload for energy efficiency and performance. *Simulation Modelling Practice and Theory*, 2017.

[87] Hugo Lucas, Rabia Ferroukhi, and IRENA Hawila, Diala. Renewable energy auctions in developing countries. *International Renewable Energy Agency*, 2013.

[88] Xinchen Lyu, Hui Tian, Cigdem Sengul, and Ping Zhang. Multiuser joint task offloading and resource optimization in proximate clouds. *IEEE Trans. on Vehicular Technology*, 2016.

[89] Guifu Ma, Xiaowei Wang, Manjiang Hu, Wenjie Ouyang, Xiaolong Chen, and Yang Li. Drl-based computation offloading with queue stability for vehicular-cloud-assisted mobile edge computing systems. *IEEE Transactions on Intelligent Vehicles*, 2022.

[90] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Comm. Surveys & Tutorials*, 2017.

[91] Timothy A Mann, Sven Gowal, Ray Jiang, Huiyi Hu, Balaji Lakshminarayanan, and Andras Gyorgy. Learning from delayed outcomes with intermediate observations. *arXiv preprint arXiv:1807.09387*, 2018.

[92] Jason R Marden, Gürdal Arslan, and Jeff S Shamma. Cooperative control and potential games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 2009.

[93] Francisco J Martinez, Manuel Fogue, Manuel Coll, Juan-Carlos Cano, Carlos T Calafate, and Pietro Manzoni. Assessing the impact of a realistic radio propagation model on vanet scenarios using real maps. In *NCA*, 2010.

[94] Mohammad Masdari, Mehdi Nouzad, and Suat Ozdemir. Qos-driven metaheuristic service composition schemes: a comprehensive overview. *Springer AI Review*, 2021.

[95] Ahlem Masmoudi, Kais Mnif, and Faouzi Zarai. A survey on radio resource allocation for v2x communication. *Wireless Communications and Mobile Computing*, 2019.

[96] Maja J Mataric. Reward functions for accelerated learning. In *Machine learning proceedings*. 1994.

[97] Panayotis Mertikopoulos and Zhengyuan Zhou. Learning in games with continuous action sets and unknown payoff functions. *Mathematical Programming*, 2019.

[98] Marvin Minsky. Steps toward artificial intelligence. *IEEE IRE*, 1961.

[99] Dov Monderer and Lloyd S Shapley. Potential games. *Games and economic behavior*, 1996.

[100] Abdel-Illah Mouaddib, Mathieu Boussard, and Maroua Bouzid. Towards a formal framework for multi-objective multiagent planning. In *AAMAS*, 2007.

[101] R.B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.

[102] Yadati Narahari. *Game theory and mechanism design*. World Scientific, 2014.

[103] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[104] Zhaolong Ning, Kaiyuan Zhang, Xiaojie Wang, Lei Guo, Xiping Hu, Jun Huang, Bin Hu, and Ricky YK Kwok. Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[105] Noam Nisan and Amir Ronen. Computationally feasible vcg mechanisms. *Journal of Artificial Intelligence Research*, 2007.

[106] António Nogueira, Paulo Salvador, Rui Valadas, and Antonio Pacheco. Fitting self-similar traffic by a superposition of mmpps modeling the distribution at multiple time scales. *IEICE Transactions on Communications*, 2004.

[107] Chang-song Oh and Jong-min Yoon. Hardware acceleration technology for deep-learning in autonomous vehicles. In *IEEE BigComp*, 2019.

[108] Jean Oh and Stephen F Smith. A few good agents: multi-agent social learning. In *AAMAS*, 2008.

[109] Efe A Ok. *Real analysis with economic applications*, volume 10. Princeton University Press, 2007.

[110] Christos H Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 2008.

[111] Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli. Policy gradient approaches for multi-objective sequential decision making. In *IEEE International Joint Conference on Neural Networks*, 2014.

[112] Simon Parsons and Michael Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2002.

[113] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[114] Fioravante Patrone, Lucia Pusillo, and Stef Tijs. Multicriteria games and potentials. *Top*, 2007.

[115] Steven Perkins, Panayotis Mertikopoulos, and David S Leslie. Mixed-strategy learning with continuous action sets. *IEEE Trans. on Automatic Control*, 2015.

[116] Patrice Perny, Paul Weng, Judy Goldsmith, and Josiah Hanna. Approximation of lorenz-optimal solutions in multiobjective markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, 2013.

[117] Matteo Pirotta, Simone Parisi, and Marcello Restelli. Multi-objective reinforcement learning with continuous pareto frontier approximation. In *AAAI*, 2015.

[118] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[119] Martin L Puterman and Shelby L Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 1979.

[120] Gabriel De Oliveira Ramos, Roxana Radulescu, and Ann Nowe. A budged-balanced tolling scheme for efficient equilibria under heterogeneous preferences. In *AAMAS ALA workshop*, 2019.

[121] Carlo Raquel and Xin Yao. Dynamic multi-objective optimization: a survey of the state-of-the-art. In *Evolutionary computation for dynamic optimization problems*. Springer, 2013.

[122] Stephen A Rhoades. The herfindahl-hirschman index. *Fed. Res. Bull.*, 1993.

[123] Jong-hyun Ryu, Sujin Kim, and Hong Wan. Pareto front approximation with adaptive weighted sum method in multiobjective simulation optimization. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE, 2009.

[124] Parnia Samimi, Youness Teimouri, and Muriati Mukhtar. A combinatorial double auction resource allocation model in cloud computing. *Information Sciences*, 2016.

[125] Robert M Schindler and Robert Schindler. *Pricing strategies: a marketing approach*. sage, 2011.

[126] Stefan Schneider, Ramin Khalili, Adnan Manzoor, Haydar Qarawlus, Rafael Schellenberg, Holger Karl, and Artur Hecker. Self-learning multi-objective service coordination using deep reinforcement learning. *IEEE Trans. on Network and Service Management*, 2021.

[127] Howard M Schwartz. *Multi-agent machine learning: A reinforcement approach*. John Wiley & Sons, 2014.

[128] Zawar Shah, Siddarth Rau, and Adeel Baig. Throughput comparison of ieee 802.11 ac and ieee 802.11 n in an indoor environment with interference. In *IEEE ITNAC*, 2015.

[129] Behrooz Shahriari. *Generic Online Learning for Partial Visible & Dynamic Environment with Delayed Feedback*. PhD thesis, San Jose State University, 2017.

[130] Farshad Shams, Giacomo Bacci, and Marco Luise. Energy-efficient power control for multiple-relay cooperative networks using q-learning. *IEEE Trans. on Wireless Comm.*, 2014.

[131] Lloyd S Shapley and Fred D Rigby. Equilibrium points in games with vector payoffs. *Naval Research Logistics Quarterly*, 1959.

[132] H. Shen and L. Chen. A resource usage intensity aware load balancing method for virtual machine migration in cloud datacenters. *IEEE Trans. on Cloud Computing*, 2020.

[133] Smitha Shivshankar and Abbas Jamalipour. An evolutionary game theory-based approach to cooperation in vanets under different network conditions. *IEEE Trans. on Vehicular Technology*, 2014.

[134] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *NeurIPS*, 2015.

[135] Jun Sun, Eytan Modiano, and Lizhong Zheng. Wireless channel allocation using an auction algorithm. *IEEE JSAC*, 2006.

[136] Wen Sun, Jiajia Liu, Yanlin Yue, and Haibin Zhang. Double auction-based resource allocation for mobile edge computing in industrial internet of things. *IEEE Transactions on Industrial Informatics*, 2018.

[137] Yuxuan Sun, Sheng Zhou, and Zhisheng Niu. Distributed task replication for vehicular edge computing: Performance analysis and learning-based algorithm. *IEEE Transactions on Wireless Communications*, 2021.

[138] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[139] Jing Tan, Ramin Khalili, and Holger Karl. Learning to bid long-term: Multi-agent reinforcement learning with long-term and sparse reward in repeated auction games. In *AAAI Workshop on Reinforcement Learning in Games*, 2022.

[140] Jing Tan, Ramin Khalili, Holger Karl, and Artur Hecker. Multi-agent distributed reinforcement learning for making decentralized offloading decisions. *IEEE INFOCOM*, 2022.

[141] Jing Tan, Ramin Khalili, Holger Karl, and Artur Hecker. Multi-agent reinforcement learning for long-term network resource allocation through auction: A v2x application. *Computer Communications*, 2022.

[142] Ming Tan. Multi-agent reinforcement learning: independent vs. cooperative agents. In *ICML*, 1993.

[143] Yinglei Teng, F Richard Yu, Ke Han, Yifei Wei, and Yong Zhang. Reinforcement-learning-based double auction design for dynamic spectrum access in cognitive radio networks. *Wireless Personal Communications*, 2013.

[144] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Australasian joint conference on artificial intelligence*, 2008.

[145] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The*

Journal of finance, 1961.

[146] J Von Neumann and O Morgenstern. Theory of games and economic behavior. 1944.

[147] Michal Vondra and Zdenek Becvar. Qos-ensuring distribution of computation load among cloud-enabled small cells. In *IEEE CloudNet*, 2014.

[148] Kai Wang, Minghong Lin, Florin Ciucu, Adam Wierman, and Chuang Lin. Characterizing the impact of the workload on the value of dynamic resizing in data centers. In *IEEE INFOCOM*, 2013.

[149] Rui Wang, Qingfu Zhang, and Tao Zhang. Decomposition-based algorithms using pareto adaptive scalarizing methods. *IEEE Transactions on Evolutionary Computation*, 2016.

[150] Dawei Wei, Junying Zhang, Mohammad Shojafar, Saru Kumari, Ning Xi, and Jianfeng Ma. Privacy-aware multiagent deep reinforcement learning for task offloading in vanet. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[151] Michael Weinberg and Jeffrey S Rosenschein. Best-response multiagent learning in non-stationary environments. In *AAMAS*, 2004.

[152] Md Whaiduzzaman, Mehdi Sookhak, Abdullah Gani, and Rajkumar Buyya. A survey on vehicular cloud computing. *Journal of Network and Computer applications*, 2014.

[153] Shichao Xia, Zhixiu Yao, Guangfu Wu, and Yun Li. Distributed offloading for cooperative intelligent transportation under heterogeneous networks. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[154] Chen Xu, Lingyang Song, Zhu Han, Dou Li, and Bingli Jiao. Resource allocation using a reverse iterative combinatorial auction for device-to-device underlay cellular networks. In *IEEE GLOBECOM*, 2012.

[155] Chen Xu, Lingyang Song, Zhu Han, Qun Zhao, Xiaoli Wang, and Bingli Jiao. Interference-aware resource allocation for device-to-device communications as an underlay using sequential second price auction. In *IEEE ICC*, 2012.

[156] Xiaolong Xu, Chenyi Yang, Muhammad Bilal, Weimin Li, and Huihui Wang. Computation offloading for energy and delay trade-offs with traffic flow prediction in edge computing-enabled iov. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[157] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *ICML*, 2018.

[158] Liang Yao, Xiaolong Xu, Muhammad Bilal, and Huihui Wang. Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[159] Xiaohu You et al. Towards 6g wireless communication networks: Vision, enabling technologies, and new paradigm shifts. *Science China Information Sciences*, 2021.

[160] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

[161] Lofti Zadeh. Optimality and non-scalar-valued performance criteria. *IEEE transactions on Automatic Control*, 8(1):59–60, 1963.

[162] Xiaoyu Zhu, Yueyi Luo, Anfeng Liu, Neal N Xiong, Mianxiong Dong, and Shaobo Zhang. A deep reinforcement learning-based resource management game in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Erklärung (Declaration of Academic Honesty)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der auf-geführten Quellen und Hilfsmittel angefertigt habe. Die selbstständige und eigen-ständige Anfertigung versichert an Eides statt:

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used. The independent and unaided completion of this thesis is affirmed by affidavit:

Potsdam, 11.03.2023

Jing Tan