Université de Lille
Faculté des Sciences et Technologies

Load-balancing and Task Allocation in Dynamic Multi-Agent Systems

Pierre LAGUE

Master Informatique
Master in Computer Science

Département d'Informatique
Faculté des Sciences et Technologies

March, 2025

This report partially satisfies the requirements defined for the Project/Internship course, in the 3rd year, of the Master in Computer Science.

Candidate: Pierre LAGUE, No. 42315164,
pierre.lague.etu@univ-lille.fr
Scientific Guidance: Maxime MORGE, maxime.morge@univ-lyon1.fr

Département d'Informatique
Faculté des Sciences et Technologies
Campus Cité Scientifique, Bât. M3 extension, 59655 Villeneuve-d'Ascq

March, 2025

I would like to dedicate this work to my mother and father, as well as to my siblings. Thank you all for your unconditional support and affection.

Acknowledgements

I would like to thank my supervisor Mr. Maxime Morge for his piece of advice, reactivity and support in this work. I would also like to thank all the other reviewers of my work for their feedback to continuously improve this thesis. I thank Mr. Philippe Preux and Mr. Philippe Mathieu for their teachings in Reinforcement Learning and Multi-Agent Systems during this second year of the masters degree. Finally, I would like to thank my classmate François Muller whose remarks on my work have provoked deeper reflection than anticipated.

Abstract

The rapid advancements in multi-agent reinforcement learning (MARL) and multi-agent systems (MAS) have introduced novel approaches to solving complex real-world problems such as decentralized decision-making, intelligent energy grids, and collaborative robotics. This thesis explores the challenges and methodologies associated with load balancing and task allocation in dynamic multi-agent systems (MAS). I present state-of-the-art review of existing methods, categorizing them into centralized, decentralized, and hybrid approaches. The study highlights the trade-offs between scalability, computational efficiency, and robustness in different frameworks, emphasizing the limitations of static and deterministic algorithms in dynamic environments. I introduce and analyze centralized methods such as market-based methods (FMC_TA) and centralized auction-based method (SCA), as well as decentralized game-theoretic and optimization-based techniques, such as the Consensus-Based Bundle Algorithm (CBBA) and Cooperative Deep Q-Learning (CDQL). Ultimately I present hybrid methods leveraging both centralized and decentralized benefits (CLDS) and Multi-Agent Reinforcement Learning-based methods (CQDL), demonstrating their effectiveness in adaptive task allocation. The results indicate that each methods stands with its own benefits and that concepts from each of them could be integrated into one another. I also conclude that incorporating learning-based strategies and agent communication enhances system performance and resilience to uncertainties in dynamic systems.

Keywords: Multi-Agent Systems, Task Allocation, Load-Balancing, Distributed Artificial Intelligence, Reinforcement Learning, Centralized Methods, Decentralized Methods, Centralized Training and Decentralized Execution

Résumé

Les avancées rapides en apprentissage par renforcement multi-agent (MARL) et en systèmes multi-agents (MAS) ont introduit de nouvelles approches pour résoudre des problèmes complexes du monde réel, tels que la prise de décision décentralisée, les réseaux énergétiques intelligents et la robotique collaborative. Ce mémoire explore les défis et les méthodologies liés à l'équilibrage de charge et à l'allocation de tâches dans des systèmes multi-agents dynamiques (MAS). Je présente un état de l'art des méthodes, en les classant en approches centralisées, décentralisées et hybrides. L'étude met en évidence les compromis entre évolutivité, efficacité computationnelle et robustesse dans différents cadres, en soulignant les limites des algorithmes statiques et déterministes dans des environnements dynamiques. J'introduis et analyse des techniques centralisées basées sur des approches de marché (FMC_TA) et d'enchères (SCA), ainsi que des méthodes décentralisées basées sur la théorie des jeux et l'optimisation, telles que l'algorithme de regroupement basé sur le consensus (CBBA) et l'apprentissage profond coopératif Q-Learning (CDQL), démontrant leur efficacité dans l'allocation adaptative des tâches. Les résultats indiquent que l'intégration de stratégies d'apprentissage et de communication entre agents améliore les performances du système et sa résilience face aux incertitudes.

Mots-clés: Systèmes Multi-Agents, Allocation de Tâches, Equilibrage de Charges, Intelligence Artificielle Distribuée, Apprentissage par Renforcement, Méthodes Centralisées, Méthodes Decentralisées, Entraînement Centralisé et Execution Décentralisée

Contents

List of Figures vii

List of Tables ix

List of Acronyms xi

1 Introduction 3
- 1.1 Overview of Multi-Agent Systems . . . 3
- 1.2 Overview of the Load Balancing Problem . . . 4
- 1.3 Motivation . . . 5
- 1.4 Objectives and Scope . . . 5
- 1.5 Thesis Outline . . . 5

2 Background 7
- 2.1 Distributed Artificial Intelligence . . . 7
  - 2.1.1 Software Agents . . . 8
- 2.2 Multi-Agent System . . . 9
- 2.3 Markov Decision Process . . . 10
  - 2.3.1 Reinforcement Learning . . . 10
  - 2.3.2 Multi-Agent Reinforcement Learning . . . 12
- 2.4 Task Allocation and Load Balancing . . . 13
  - 2.4.1 Task Allocation in MAS . . . 13
  - 2.4.2 Load Balancing in MAS . . . 14
- 2.5 Major Challenges . . . 15
  - 2.5.1 Temporal Constraints . . . 15
  - 2.5.2 Communicational Constraints . . . 16
  - 2.5.3 Spatial Constraints . . . 16
  - 2.5.4 Robustness to Uncertainty . . . 17
  - 2.5.5 Dynamic Adaptation . . . 17
- 2.6 Performance Evaluation . . . 19

3 Methods For Load-Balancing and Task Allocation in dynamic MAS 21
- 3.1 Centralized Methods . . . 21
  - 3.1.1 Fisher Market-Based Approaches . . . 21

3.1.2 Centralized Auction-Based Approaches 25
Stochastic Clustering Auction 26
3.1.3 Conclusion on Centralized Methods 28
3.2 Decentralized Methods 28
3.2.1 Game Theory-Based Method 30
Distributed Stochastic Algorithm 32
3.2.2 Distributed Optimization-Based Methods 33
Consensus-Based Decentralized Auctions 34
Adaptive Load Balancing 40
3.2.3 Conclusion on Decentralized Methods 46
3.3 Centralized Training, Decentralized Execution 47
3.3.1 Centralized-Decentralized Reinforcement Learning 48
CLDS Algorithm 49
Proposed Details on the Reinforcement Learning Framework 51
3.3.2 Multi-Agent Deep Reinforcement Learning 52
Cooperative Deep Reinforcement Learning Strategy 52
3.3.3 Task Allocation Process using CQDL 53
3.3.4 Conclusion on CTDE 57

4 Discussion **59**
4.1 About Complexities 59
4.2 About Limitations 61

5 Conclusion **65**

6 Appendix A: CLDS Algorithm **67**

7 Appendix B: Comparative Table **69**

References **71**

List of Figures

1.1 Recent work by OpenAI researchers [1] have displayed the intricate interactions between agents playing hide (blue agents) and seek (red agents) 4

List of Tables

3.1 Experimental Settings for Small-World and Scale-Free Networks . . 56
3.2 Comparison of Utility Ratio and Execution Time in Setting 1 56
3.3 Comparison of Utility Ratio and Execution Time in Setting 2 57

7.1 Comparison of Centralized, Decentralized, and Hybrid Load Balancing Methods 70

List of Acronyms

AI	Artificial Intelligence
CBAA	Consensus-Based Auction Algorithm
CBBA	Consensus-Based Bundle Algorithm
CDRL	Cooperative Deep Reinforcement Learning
CLDS	Centralized Learning Distributed Scheduling
CN	Communicating Neighborhoods
CQDL	Cooperative Deep-Q-Learning
CTDE	Centralized Training and Decentralized Execution
DAI	Distributed Artificial Intelligence
DOM	Distributed optimization methods
DPG	Deterministic Policy Gradient
DPS	Distributed Problem Solving
DQL	Deep Q-Learning
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
DSA	Decentralized Stochastic Algorithm
FMC_TA	Fisher Market-Clearing Task Allocation
GDAP	Greedy Distributed Allocation Protocol
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MADRL	Multi-Agent Deep Reinforcement Learning
MAMRSS	Multi-Agent Multi-Resource Stochastic System
MARL	Multi-Agent Reinforcement Learning

MAS	Multi-Agent Systems
MATA	Multi-Agent Task Allocation
MDPs	Markov Decision Processes
MILP	Mixer Integer Linear Programming
NBS	Nash Bargaining Solution
NCN	Non-Communicating Neighborhoods
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RMAAC	Robust Multi-Agent Actor-Critic
RMAQ	Robust Multi-Agent Q-learning
SA	Situational Awareness
SCA	Stochastic Clustering Auction
SR	Selection Rules
TAP CQDL	Task Allocation Process using Cooperative Deep Q-learning
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

This chapter introduces the work presented in this thesis. Particularly, the introduction to the key concepts of the thesis, then the objectives and scope of the research is described briefly. The chapter will conclude on an overview of the thesis outline.

1.1 Overview of Multi-Agent Systems

Multi-Agent Systems (MAS) is a powerful computational model that use distributed agents to model or solve complex problems. MAS have been a very active research topic over the years thanks to the very wide application panel it proposes. In a MAS, agents are independent entities that can represent real-world entities, such as biological cells, virus particles, ants, individual computers or even human beings [1]. Each agent has a set of encoded rules or behaviors that dictate how they interact with other agents making MAS particularly well-suited for decentralized computing. In a decentralized system, there is no central controller, and global properties emerge from the local interactions between agents. Interactions between agents can be cooperative or competitive. That is, the agents can pursue the system's goal by communicating and sharing a similar goal, or the agents can pursue their own interest. The difficulty being figuring out what course of actions should the agents follow in order to maximize a given metric or reward. As these agents interact with each other they are able to solve complex tasks such as load balancing which could not be completed efficiently by a single agent alone [2].

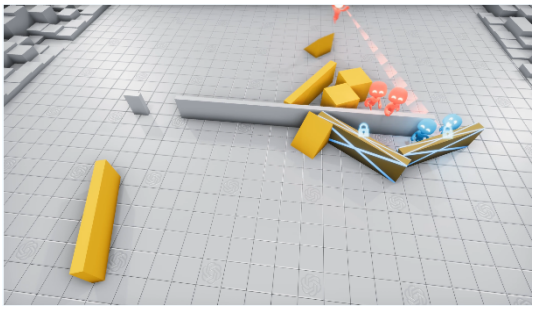

Figure 1.1: Recent work by OpenAI researchers [1] have displayed the intricate interactions between agents playing hide (blue agents) and seek (red agents)

1.2 Overview of the Load Balancing Problem

Load balancing is a crucial aspect of optimizing the performance of parallel and distributed systems. The core idea is to distribute workload evenly across multiple processors or computing nodes to prevent any single node from becoming overloaded and bottle-necking the system. A sub-problem, within the broader framework of load balancing, is task allocation. Effective task allocation is a prerequisite for achieving well-balanced loads across a system, making the two concepts highly interdependent. The following example is a well fitted analogy :

- Load Balancing: The head chef ensures all chefs in the kitchen have roughly equal workloads.
- Task Allocation: Assigning specific dishes to chefs based on their expertise, current workload, and station availability.

Mathematically speaking, load balancing is a global optimization problem where the objective is to minimize variance in resource utilization across the system. Where task allocation is local decision-making or assignment problem that contributes to achieving the global load balancing goal. Load balancing and task allocation are NP-hard problems when it comes to solving them in a multi-agent environment with specific constraints [3].

Both of these problems are widely studied in their distributed version [4], making them particularly interesting to study in MAS. However the aim to solve real life problems such as rescue missions, or managing network flow through computing nodes implies that the environment in which the agents take their decision is "complex". That is, unknown, or partially unknown, to the agents, stochastic, constrained (spatially, temporally and communication-wise). In addition to being complex, these environments can also have a dynamic aspect (change of task frequency, change of agent capacity to handle a task etc.). When implementing the agents and allowing them to learn their role, all of these constraints and dynamic behaviors must be kept in mind.

In this sense, the design of the algorithms and processes that will allow load balancing or task allocation must include spatial, temporal and communicational constraints. Spatial constraints are linked to the location of either the agent or the task; temporal constraints are concerned with the deadline to start or end any task; communicational constraints are related to the communication network topography (which agents are neighbors to one another).To accomplish any complex task, agents need to communicate with one another, either finding consensus, sharing information or even being competitive against each other.

1.3 Motivation

This thesis marks the culmination of my Master's degree in Computer Science at the University of Lille and embodies my dedication to understanding the complexities of multi-agent systems. It is inspired by the rapid advancements in Multi-Agent Reinforcement Learning (MARL), which have shown potential for addressing real-world challenges, such as decentralized decision-making in autonomous vehicles, intelligent energy grids, collaborative robotics, and, as of recently, agent-based learning for large language models and multi-modal models.

Driven by the increasing demand for intelligent systems that are both efficient and adaptive, this research explores dynamic load balancing and task allocation in complex multi-agent environments. The insights gained from this work align with my aspiration to pursue a Ph.D. in reinforcement learning and multi-agent systems. My long-term goal is to contribute to research that link theory and real-life applications.

1.4 Objectives and Scope

This thesis aims to present **a state-of-the-art review of the approaches and challenges involved in achieving efficient, robust, and dynamic load balancing and task allocation in complex multi-agent systems.** By analyzing existing methodologies, identifying limitations, and highlighting results, the thesis seeks to provide a foundation for understanding how to effectively and dynamically distribute workloads or allocate tasks in distributed, dynamic environments. This work, however, will not delve into the intricacies of all the possible algorithms, but we will go into the details of the ones that yield significant results.

1.5 Thesis Outline

During this work we will detail various approaches to solving load balancing and task allocation problems in a dynamic fashion in a MAS.

In section 2, we will first introduce all the necessary background knowledge and foundational work for this thesis. That is, introduce the concepts of distributed artificial intelligence and what intelligent agents are. Then we will formally define what a Multi-Agent Systems is and introduce the theoretical foundations of Markov decision process, further extended into reinforcement learning and multi-agent reinforcement learning. In addition to this, we will provide a formal definition of load balancing and task allocation problems. Finally, we will go through all the challenges, and how they are expressed in the literature, hinting the possible solutions to these challenges and how they are addressed. We will also briefly talk about how algorithms are commonly evaluated to assess their performance in load balancing.

The following chapter 3 is the heart of our work. We aim to present the existing methods under 3 panels:

- Centralized Methods – These rely on a single control unit, often referred to as a central brain, which gathers global information and makes decisions for all agents. This approach ensures optimal coordination but suffers from scalability issues, communication overhead, and vulnerability to single points of failure.
- Decentralized Methods – These eliminate the need for a central control unit, allowing agents to independently learn and adapt based on local information and interactions. While more scalable and robust, decentralized approaches introduce challenges in coordination, convergence, and efficiency due to the lack of global monitoring.
- Centralized Training, Decentralized Execution Paradigm - This hybrid approach makes use of a centralized training phase, where agents learn optimal policies using global knowledge, but during execution, they operate independently based on local observations. This method balances the efficiency of centralized learning with the scalability and resilience of decentralized execution.

The final section of our work will be aimed at debating each of the 3 panels and related methods, by presenting the advantages and disadvantages of each, and putting the studied methods in parallel. After this discussion section, we will conclude on our work.

In this work, each piece of information related to an article has been rigorously cited. Articles are not necessarily treated in a chronological way but more in a constructive and complementary way. For the sake of simplicity we have bounded our research up to articles published in 2024.

Chapter 2

Background

This chapter provides the theoretical foundations necessary to understand the core concepts that underpin our work. We begin by formally introducing Distributed Artificial Intelligence (DAI), software agents, and multi-agent systems, as these form the fundamental framework upon which our research is built. We then present Reinforcement Learning (RL) and its extension to multi-agent settings (Multi-Agent Reinforcement Learning (MARL)), which are essential paradigms for developing adaptive solutions. Then, we formally define both the load balancing and task allocation problems, as they represent the specific problems we aim to address. Finally, we will review the challenges to which the scientific community is confronted to when it comes to solving the problems of task allocation and load balancing. Understanding these concepts is crucial, as they serve as building blocks for the algorithms and methodologies presented in subsequent chapters.

2.1 Distributed Artificial Intelligence

Distributed Artificial Intelligence is a subfield of Artificial Intelligence (AI) that focuses on designing systems where multiple autonomous entities, known as agents, interact and collaborate to solve complex problems. These systems are characterized by decentralization, parallelism, and the ability to handle dynamic and heterogeneous environments [5].

DAI can be mathematically represented as: $DAI = \{A, E, O\}$ where :

- $A = \{a_1, a_2, ..., a_n\}$ is a set of autonomous agents.

- E is the environment in which agents operate (including constraints).
- O is the set of objectives, either shared or individual.

Within DAI we find two research areas : Distributed Problem Solving (DPS) and Multi-agent systems. DPS focuses on the problem and how to solve it with multiple programmed entities collaborating together. In MAS, as stated before, the components are intelligent agents which have some autonomous characteristics. On one hand DPS handles agents that have a predefined range of actions and interactions, on the other hand, MAS possesses the property of reasoning the coordination problem among agents themselves with no predefined scenario [6].

Now that we have settled the foundations of DAI, we can delve into how software agents are defined.

2.1.1 Software Agents

A software agent is a computational entity that performs tasks for its user within a computing environment, operating autonomously to achieve specific objectives. Agents are distinguished from simple programs by their autonomy, reactivity and social ability [7]. They can also be classified based on their ability to reason, learn and interact with other agents or systems [8]. There are 4 types of agents that are widely used :

- Reactive Agents: respond to environmental stimuli without long-term planning.
- Deliberative Agents: use symbolic reasoning and planning to make decisions.
- Hybrid Agents: combine reactive and deliberative approaches.
- Learning Agents: employ techniques like reinforcement learning to improve over time.

Generally, learning agents are dependent on each other. They interact with other agents in order to meet their design objectives. Thus agent forms group to achieve the system goals. This grouping constitutes the multi-agent system. Agents in cooperative multi-agent system coordinate their actions with other agents to fulfill its goals. For cooperative multi-agent systems, task allocation and, eventually, load balancing is an important requirement, not only for the objective of the system but for its efficiency.

A software agent can be defined as a tuple [7] $A = (E, P, S, \pi)$ where :

- E represents the environment in which the agent operates.
- P is the perception function that maps environmental states to percepts.

- S is the internal state of the agent (knowledge, goals, history).
- π is the policy, or decision-making function that determines the agent's actions based on it's perception and state S : $\pi(P, S) \longrightarrow Actions$.

Now that we have formally defined what a software agent is, we can analyze the multi-agent paradigm and how it is defined.

2.2 Multi-Agent System

As stated in the introduction, a MAS is a system in which multiple agents interact with each other to achieve their goals. Drawing particularities from [9], [7], and [8], a multi-agent system can be formally defined as a tuple : $MAS = (A, E, I, O, R)$ where :

- $A = \{a_1, a_2, ...a_n\}$ is a finite set of agents.
- E represents the environment and the rules in which the agent operates (similar to DAI).
- I represents the interactions between agents consisting of communication protocols, resource sharing protocols or coordination directives.
- O represents the organizational structure, including roles, responsibilities and authority relationships.
- R represents the set of relationships and constraints, temporal and spatial constraints, dependencies between tasks and agents, and resource allocation constraints.

Additionally, the system can be [4] :

- Decentralized (resp. centralized) : no single agent has complete control over the system (resp. one agent monitors the system and directs other agents).
- Distributed : meaning the agents are distributed across the environment.
- Scalable : the system can accommodate changes in size and complexity.
- Adaptive : the system can adjust to dynamic changes in the environment.

Although some centralized approaches [10, 11] yield significant results when it comes to solving the problem of task allocation, they have flaws like being in a state of single-point of failure when a single *central brain* handles the transactions from the cohort of agents as well as poor scalability in general.

Famous scientific outbreaks in this field have shown that multi-agent systems excelled at certain tasks using machine learning methods like reinforcement learning

for single agent tasks (Alpha Go [12]) and multi-agent reinforcement learning for larger environment tasks with complex agent interactions (Alpha Star [13]). We will focus on defining these two key concepts: Reinforcement Learning and Multi-Agent Reinforcement Learning that are essential to understand the underlying theory behind some of the approaches to solving load balancing.

2.3 Markov Decision Process

Markov Decision Processes (MDPs) are defined as a discrete-time stochastic control process characterized by a set of states S, a set of actions A, transition probabilities P, and reward functions R [14]. At each time step, the decision-maker observes the current state $s \in S$, selects an action $a \in A$, and then the system transitions to a new state s' according to the probability distribution $P(s'|s,a)$, i.e. the probability of transitioning to state s' knowing the agent executed action a in state s. This transition yields the corresponding reward $R(s,a,s')$, which is the reward obtained from the transition from state s to state s' executing action a. The objective is to determine an optimal policy π that maximizes the expected cumulative reward over time considering a specified criterion such as a discount factor γ. Methods such as value iteration, policy iteration, and linear programming allow to find this optimal policy [15].

2.3.1 Reinforcement Learning

Reinforcement Learning can be formally defined as a sequential decision-making framework represented by a Markov Decision Process [16] [17], which is characterized by a tuple $MDP = (S, A, P, R, \gamma)$ where :

- S is the state space, representing all the possible states of the environment.
- A is the action space, containing all possible actions available to the agent.
- $P : S \times A \times S \longrightarrow [0,1]$ is the state transition probability function. $P(s'|s,a)$ is the probability of transitioning to state s' when taking action a in state s.
- $R : S \times A \times S \longrightarrow \mathbf{R}$ is the reward function. $R(s',a,s)$ represents the immediate reward received when transitioning from state s to state s' by taking action a.
- $\gamma \in [0,1]$ is the discount factor that determines the importance of future rewards. A higher γ will make the agents favor long-term rewards, where a lower γ will make the agent favor immediate rewards.

The objective in RL is to find an optimal action policy $\pi* : S \longrightarrow A$ that maximizes the expected cumulative discounted rewards:

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})] \tag{2.1}$$

This value function $V^{\pi}(s)$ represents the expected return when starting from state s and following policy π thereafter. This policy can be found through various methods like value-based methods such as *Q-Learning* [18, 19]. *Q-Learning* relies on learning the quality of state-action (s, a) pairs, represented by a Q-Value which estimates the expected cumulative reward achievable from a state s by taking action a and following the optimal after. A tabular version estimates the Q-Value by storing it and updating it in a Q-Table. Q-learning has yielded significant results in discrete state spaces, but when confronted with continuous state spaces, the tabular nature Q-learning makes learning intractable. To address this, Deep Q-Networks (DQN) [20] use neural networks to approximate the Q-function, allowing agents to generalize over complex state spaces. DQN incorporates techniques like experience replay (storing and reusing past experiences) and target networks (stabilizing updates) to improve learning stability. For each approach, the Q-values are updated using the bellman equation as referral:

$$Q(s, a) \longleftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)] \tag{2.2}$$

Where :

- $Q(s, a)$: the Q-Value for state s and action a.
- α : the learning rate controlling the influence of new information.
- r : the immediate reward received after taking action a in state s.
- γ : the discount factor, balancing immediate and long term reward.
- $max_{a'} Q(s', a')a$: the maximum predicted Q-value for the next state s'.
- s' : the next state resulting from the action a.

More complex approaches such as policy-based reinforcement learning methods [21] directly optimize a parameterized policy $\pi(a|s, \theta)$ to maximize the expected cumulative reward $J(\theta) = \mathbb{E}_{\pi_\theta} [\sum_{t=0}^{\infty} \gamma^t r_t]$. Unlike value-based methods, these approaches do not require a value function approximation and are well-suited for continuous or high-dimensional action spaces. The optimization often relies on gradient-based techniques, with the policy gradient theorem providing the foundation [22]. Extensions like actor-critic methods [23] [12] combine policy optimization (actor) with value estimation (critic) for improved stability. Proximal Policy Optimization (PPO) introduces an objective to constrain updates and maintain stability, while Deterministic Policy Gradient (DPG) methods are effective for continuous control.

Policy-based methods are particularly relevant in Multi-Agent Reinforcement Learning, where agents must optimize their individual policies while considering interactions and shared environments.

2.3.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning extends the single-agent Reinforcement Learning framework to environments where multiple agents learn simultaneously. The framework can be formalized as a Markov Game (also known as a stochastic game) [16, 17] defined by the tuple $(N, S, \{A_i\}_{i \in N}, P, \{R_i\}_{i \in N}, \gamma)$ where :

- Elements S, P, and γ are identical to the elements in the single-agent RL framework presented earlier.
- $N = \{1, ..., n\}$ is the finite set of agents. A_i is the action space of agent i.
- $R_i : S \times A_1 \times ... \times A_n \times S \longrightarrow \mathbf{R}$ is the reward function for agent i.

Each agent aims to find a policy $\pi_i : S \longrightarrow \Delta(A_i)$ that maximizes its expected discounted reward :

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R_i(s_t, a_{1_t}, ..., a_{n_t}, s_{t+1})] \tag{2.3}$$

where $\pi = (\pi_1, ..., \pi_n)$ is the joint policy of all agents.

In MARL, the interaction among agents often introduces unique challenges. The environment becomes non-stationary, as the policies of other agents evolve over time. Furthermore, depending on the problem, agents may need to cooperate, compete, or navigate a combination of both, which requires careful design of reward structures. For instance, cooperative environments often involve a global reward function, while competitive scenarios may be modeled as zero-sum games. Hybrid settings blend these extremes, complicating policy optimization.

A prominent framework for MARL is Centralized Training and Decentralized Execution (CTDE), where agents leverage global information during training but act independently at runtime. Algorithms like Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [24] extend actor-critic methods to such setups. Despite progress, MARL faces significant hurdles, including scalability issues due to the exponential growth of the joint action space, communication overhead in distributed systems, and challenges in balancing exploration and exploitation. These factors make MARL an active area of research with applications spanning robotics, autonomous vehicles, and resource allocation.

2.4 Task Allocation and Load Balancing

The following section aims to present a general framework for both the task allocation and load balancing problems. This framework is voluntarily generalized and will be detailed in further chapters when we analyze a given approach to solve them.

2.4.1 Task Allocation in MAS

The Multi-Agent Task Allocation (MATA) problem can be formally defined as an optimization problem within a MAS framework. We consider MAS to be a MAS as defined in 2.2, then the task allocation problem is defined by the tuple (A, T, C, K, Q) [25, 26] where :

- $A = \{a_1, ..., a_n\}$ is the finite set of agents.
- $T = \{t_1, ..., t_m\}$ is the finite set of tasks to be allocated.
- $C : A \times T \longrightarrow \mathbf{R}^+$ represents the cost function for agent-task pairs.
- $K : T \longrightarrow \mathbf{N}$ specifies the number of agents required for each task.
- $Q : 2^A \times T \longrightarrow \mathbf{R}$ defines the quality of task execution by a group of agents.

In certain cases [27] [28] the allocation must satisfy certain constraints. As mentioned in 1.2 constraints can be spatial, temporal, and communicational in certain cases. For the sake of simplicity, we will present 3 constraints that are applicable to a variety of systems and algorithms :

- Task Coverage : $\forall t \in T : |a \in A | allocated(a, t)| = K(t)$ i.e. for each tasks, the correct amount of agents are assigned to the task.
- Agent Capacity : $\forall a \in A : \sum_{t \in T} allocated(a, t) \leq cap(a)$ i.e. for each agent assigned to a task, the sum of the allocated tasks is below the agents' capacity.
- Temporal Constraints : $\forall t, t' \in T : precedes(t, t') \longrightarrow completion\_time(t) \leq start\_time(t')$ i.e. if is there is a precedence relationship between task t and t', the completion time of task t must be before the start time of task t'.

Here, $allocated(a, t)$ is a binary function indicating if agent a is allocated to task t, $cap(a)$ represents the capacity of agent a, $precedes(t, t')$ indicated task t precedes task t'.

Finally, the global optimization objective of the system can be defined as :

$$minimize \sum_{t \in T} \sum_{S \subseteq A} C(S, t) \times x(S, t) \tag{2.4}$$

with $x(S,t)$ a binary variable that indicates whether a subset of agents $S \subseteq A$ is assigned to task t. The equation 2.4 is subject to the following constraints: (1) each task is assigned to only one subset $S \subseteq A$, for each agent of subset S and for each task $t \in T$, (2) the number of tasks t_i assigned to agent a_j is lesser than agent a_j's capacity. Finally (3), $x(S,t) \in 0,1$ is a binary function indicating that task t is assigned to agent a, and $C(S,t)$ represents the cost of agent subset S executing task t.

Formally, these constraints can be formalized as such:

1. $\sum_{S \subseteq A} x(S,t) = 1, \forall t \in T$
2. $\sum_{t \in T} \sum_{S \subseteq A_{a_j \in S}} x(S,t) \leq cap(a_j), \forall a_j \in A$
3. $x(S,t) \in \{0,1\}, \forall S \subseteq A, \forall t \in T$

In some dynamic environments [29], the problem extends to include time-varying task arrival or task spawn rate (tasks appear faster or slower), agent availability changes and dynamic cost functions based on the time of completion (depending on how soon the task has been completed), as well as variations in agent capacity.

2.4.2 Load Balancing in MAS

As presented in section 1.2 the task allocation problem is a prerequisite for effective load balancing in a MAS. Given an initial task allocation problem, and according to [30, 31] the load balancing problem can extend the system state with (L, R, B) where :

- $L = l_1(t), ..., l_m(t)$ represent dynamic workloads derived from allocated tasks.
- $R : A \longrightarrow \mathbf{R}^+$ defines agent resource capacities.
- $B : 2^A \longrightarrow \mathbf{R}^+$ measures system balance.

The system state at time t is characterized by: $LD(t) = \{w_1(t), ..., w_n(t)\}$ where $w_i(t) = \sum_{t \in T} l_i(t)$, T_i being tasks allocated to agent i. Depending on the problem the optimization function can be simply minimizing the load variance across nodes [32]. A more complex optimization function can also take into account a measure of load imbalance and the system overhead due to redistribution [33]. Load imbalance is defined as : $\gamma(t) = \frac{(max_{i \in A} w_i(t) - min_{i \in A}}{\bar{w}(t))}$ with $w_i(t)$ being the workload of agent i and $\bar{w}(t)$ the average workload in the system at time t [31]. As for system overhead it includes communication and migration costs [30]. This problem is subject to the same kind of constraints as the task allocation problem but has additional constraints regarding the resource capacity and the total workload conservation. In fact, for any agent i, his workload must be lesser or equal to his resource capacity $R(a_i)$ and the

sum of the workload of all agents has to be equal to the total workload of the system as per [31]. Although there are solutions that allow these problems to be solved in polynomial or pseudo-polynomial time (centrally or distributedly, respectively) [10], the constraints (temporal, spatial or communicational) represent the main difficulty of these problems.

Now that we have defined the key concepts of our study, we will now present the challenges to which the scientific community has been confronted while developing the solutions we will present in a later chapter.

2.5 Major Challenges

As introduced in the section above, load balancing and task allocation are critical components in distributed computing systems, aiming to optimize resource utilization, minimize response times, and ensure system reliability. Many challenges are faced in this context, particularly concerning temporal, spatial and communicational constraints, robustness to uncertainty, and dynamic adaptation. This section aims to present these challenges in order to pave the way to present the solutions that have been provided in the next chapter.

2.5.1 Temporal Constraints

Temporal constraints involve the timing requirements associated with task execution and resource availability as presented in 2.4.1. The difficulty being, scheduling tasks within specific time windows, ensuring timely completion to meet deadlines or respect preceding relationships. In MAS, dynamically allocating tasks under time constraints necessitates algorithms that can adapt to varying workloads and agent capabilities. Amador et al. [10] addresses the problem by proposing a task allocation algorithm that allows tasks to be easily sequenced to yield high-quality solutions. This algorithm first finds allocations that are fair (envyfree, i.e. for a given allocated task, other agents won't envy it), balancing the load and sharing important tasks between agents, and efficient (Pareto optimal [1]) in a simplified version of the problem. Such constraints can be overcome in polynomial time using a Fisher market [2] with agents as buyers and tasks as goods. It then heuristically schedules the allocations, taking into account temporal constraints on shared tasks. Other authors such as Choudhury et al. [34] address the problem of dynamically allocating tasks to multiple agents under time window constraints and task completion uncertainty, aiming to minimize unsuccessful tasks at the end of the operation horizon. Generally,

[1] denoting a distribution of wealth such that any redistribution or other change beneficial to one individual is detrimental to one or more others.

[2] A Fisher market is an economic model for resource allocation where agents (buyers) with fixed budgets bid on goods (tasks), and prices adjust dynamically until supply meets demand, resulting in Pareto optimal and envy-free allocations

these time constraints are respected thanks to an efficient communication among agents, making communicational constraints another important challenge.

2.5.2 Communicational Constraints

Effective communication is vital for coordinating distributed tasks. In the reinforcement learning paradigm, agents are supposed to have a purely local knowledge of the environment and of the neighboring agents. This assumption is in agreement with the MARL framework. In fact, MARL is most interesting where real life forces agents to act without a-priori arranged communication channels and must rely on action-feedback mechanisms. However, it is of interest to understand the effects of communication on the system efficiency [35], where the agents are augmented with some sort of communication capabilities. This is why Schaerf et al. [36] worked on *neighborhoods* as a local concept between agents. The authors assume that each agent can communicate only with some of the other agents, which they call its neighbors. They consider a relation neighbor-of and assume it is reflexive, symmetric and transitive. As a consequence, the relation neighbor-of partitions the population into equivalence classes, that they call neighborhoods. Neighbors effectively communicate performance indicators related to tasks and agents allowing a augmentation of agent knowledge. We will later see that this sort of communication yields both good and bad results.

There are other challenges such as managing communication overhead (amount of time spent communicating instead of solving the problem), ensuring data consistency, and handling latency issues. High communication costs can negate the benefits of parallelism, making it essential to design load balancing algorithms that minimize inter-process communication. Lifflander et al. [37] proposes a unified, reduced-order model that combines computation, communication, and memory considerations to describe "work" in a distributed system, allowing an optimizer to explore complex compromises in task placement. However, these kind of communicational constraints are very related to the spatial constraints of the agents or resources positions.

2.5.3 Spatial Constraints

Spatial constraints are very dependant on temporal constraints and communicational constraints. For example, in geographically distributed systems, communication latencies are non-negligible and must be factored into load balancing strategies. The perceived processing time of a request includes both the routing time to the server and the actual processing time, which depends on the server's load. Skowron and Rzadca [38] address this by considering load balancing algorithms that account for communication-balanced assignments, ensuring that the reduction in processing

time from moving a task to a less loaded server outweighs the additional communication latency. Rutten and Mukherjee [39] analyze load balancing on spatial graphs, developing a coupling-based approach to establish mean-field approximations for a large class of graphs, including spatial ones. Interestingly, Schaerf et al. [36] consider different network topologies defining the neighbors and find out that certain network architectures used to communicate between agents undermine the performance of the system when it has heterogeneous agent populations (communicating and non communicating). The reason for this is that members of a communicating group tend to be very conservative, in the sense that they mostly use the best resource. And the bigger the group, the more conservative the members tend to be. However, communication among agents is not completely useless, but it is something that has to be manipulated carefully given an experimental setup and the content of the communication has to properly represent the agent state.

2.5.4 Robustness to Uncertainty

Uncertainty in MAS comes from factors such as environmental change, incomplete information, and unpredictable agent behaviors. Traditional Multi-Agent Reinforcement Learning algorithms often assume accurate models of the environment and other agents, which is not often the case in real-world scenarios. This can lead to suboptimal performance when agents encounter unexpected situations. To address this, robust MARL frameworks have been developed. For instance, Zhang et al. [40] introduce a robust MARL approach that accounts for model uncertainties by formulating the problem as a robust Markov game, enabling agents to learn policies that are resilient to such uncertainties. He et al. [41] introduce a Markov Game model with state perturbation adversaries to account for state uncertainties, proposing robust equilibrium concepts and algorithms like Robust Multi-Agent Q-learning (RMAQ) and Robust Multi-Agent Actor-Critic (RMAAC) to enhance policy robustness. Similarly, Shi et al. [42] focus on distributed robust Markov games, where each agent aims to maximize its worst-case performance within a defined uncertainty set, offering sample-efficient algorithms with finite-sample complexity guarantees [3]. This approach ensures robust policies even when the environment deviates from expected conditions. This also requires an adaptive characteristic from the agents in order to properly anticipate or react to a changing environment or other agents unexpected behavior.

2.5.5 Dynamic Adaptation

Dynamic adaptation refers to an agent's ability to adapt its behavior according to a changing environment or agent interactions. Conventional MARL methods struggle

[3] In MARL it refers to algorithms designed to learn effective policies using a minimal number of samples, while providing theoretical assurances on their performance based on the available data.

with dynamic adaptation due to their dependence on static policies learned during training, which do not generalize well to evolving environments. This limitation can deteriorate the system's responsiveness, flexibility and overall integrity. To overcome this, adaptive strategies have been proposed. For example Schaerf et al. [36] introduce a framework for adaptive load balancing called a Multi-Agent Multi-Resource Stochastic System (MAMRSS) in which each of the resources has a certain capacity, which is a real number; this capacity changes over time, as determined by a probabilistic function C. At each time point each agent is either idle or engaged. If it is idle, it may submit a new job with a probability given by P. Each job has a certain size which is also a real number. The size of any submitted job is determined by the probabilistic function D. For each new job the agent selects one of the resources according to a selection rule which are adaptive to the characteristics of the agents and the environment. That way agent, who rely on purely local information, are able to adapt to environment change via C and D, but can also adapt to new agent behavior via P. More recently, Fernander et al. [43] proposed an adaptive asynchronous work-stealing method that collects real-time information about system nodes to improve task distribution by dynamically selecting appropriate nodes for task execution. This method has proven itself 10.1% more efficient that standard state-of-the-art methods in the literature. Most of these approaches are in a decentralized setting where each agent has it's own course of action and decision making ability. Alternatively, a centralized method introduced by Zhang et al. [11] called Stochastic Clustering Auction (SCA) adapts to changes in task requirements or agent capabilities during runtime, ensuring robust performance under dynamic conditions. Additionally, the algorithm handles differences in agent capabilities by incorporating individual cost functions, allowing an efficient task distribution across diverse agents, this ensures the system and central auctioneer a broader range of action tackling the constraint of highly dynamic environments. Another approach proposed in a recent PhD Thesis by E. Beauprez [44] presents a system designed to perform ongoing concurrent negotiations while tasks are being executed. This means that as new tasks arrive or as conditions change (e.g. environment capacity change or resource fluctuation), agents can dynamically adjust the workload distribution. This hybrid approach mixes local-decision making with global benefits and quickly adapts to a dynamic environment because decisions don't need the approval of a central controller. The author has shown that this system ensures an even task distribution across the system and prevents any agent of becoming a bottleneck. The decentralized aspect inherently supports scalability allowing the system to perform, even as the number of tasks and agents grow.

2.6 Performance Evaluation

Evaluating algorithms designed for load balancing or task allocation in multi-agent systems involves multiple key performance metric to ensure efficiency and effectiveness. Common metrics include response time, throughput, resource utilization and makespan are critical factors in assessing the performance of such algorithms. Generally, agent performance metrics ensure that the load balancing or task allocation adheres to the constraints of the system. In terms of computing performance, the time complexity of each algorithm is commonly cited as an indicator of performance and scalability. Other systems will prefer evaluating the algorithm over metrics such as system stability, agent workload distribution and total computational overhead to compare different approaches or setups. Overall, studies highlight the importance of balancing global optimality with real-time computational feasibility, a critical factor in multi-agent load balancing strategies. Metrics are to be considered useful for a given application of load balancing and they all find theoretical justification behind the experimental setup [45].

Now that we have presented the foundations of distributed artificial intelligence, software agents, multi-agent systems, single and multi-agent reinforcement learning, and problems such as task allocation and load balancing, we can delve into the methods that have been developed that yield significant results into solving these problems.

Chapter 3

Methods For Load-Balancing and Task Allocation in dynamic MAS

.

3.1 Centralized Methods

Centralized methods rely on a single controller to make task allocation and load distribution decisions across the entire system. These approaches are advantageous for their ability to leverage global information for optimal resource utilization but face scalability and robustness challenges as stated in the previous chapter. From an engineering point of view, these approaches also have a single point of failure in the system which makes them highly risky. We will see that such methods can take roots in optimization strategies ranging from market clearing to Markov chain Monte Carlo approach.

3.1.1 Fisher Market-Based Approaches

The Fisher market model offers an economic framework to address task allocation challenges in multi-agent systems. In this approach, tasks are treated as "goods," agents act as "buyers" with fixed budgets, and a centralized controller adjusts task

prices dynamically to achieve equilibrium between supply and demand. This ensures two desirable properties: envy-free allocations, where no agent prefers tasks assigned to others, and Pareto optimality, where no reallocation can improve one agent's outcome without worsening another's.

Amador et al. [10] extended this model to solve the dynamic multi-agent task allocation problem under spatial and temporal constraints with the Fisher Market-Clearing Task Allocation (FMC_TA) algorithm. Their work addresses situations where agents and tasks are distributed geographically, and tasks must be executed within strict deadlines while respecting precedence relationships. By formulating the problem as a Fisher market, the centralized controller dynamically adjusts allocations as the environment evolves, balancing fairness and efficiency.

This approach ensures a fair task distribution. That is, tasks are allocated evenly, respecting agent capacities and workloads constraints. It also ensures efficiency. In fact, allocations are Pareto optimal, minimizing system-wide costs while meeting constraints. Finally, the approach dynamically adapts to changes in task demands and agent availability in real-time. This algorithm consists of two main stages: Task Allocation and Scheduling. I will propose a detailed account of these two steps.

1. Task Allocation Phase

This phase uses a Fisher market mechanism to allocate tasks to agents fairly (envy-free) and efficiently (Pareto-optimal) in a simplified model of the problem.

Initialization

- Let $A = \{a_1, a_2, \ldots, a_n\}$ be the set of agents.
- Let $T = \{t_1, t_2, \ldots, t_m\}$ be the set of tasks (further in the algorithm, the author refer to the tasks as v_i).
- Each agent a_i is endowed with an equal budget of money.

Preference Matrix

The algorithm first constructs a preference matrix R, where r_{ij} represents the utility for agent a_i to execute task t_j:

$$r_{ij} = \delta(t_j, \tau_i + \rho(a_i, t_j)) \cdot \max_q \mathrm{Cap}(t_j, q) - \pi(CT_i, \Delta w),$$

where:

- $\delta(t_j, \cdot)$ represents the soft deadline function (utility decay over time). For tasks where there are no deadlines, $\delta(t_j, \cdot) = 1$.
- τ_i is the agent a_i's current time.

- $\rho(a_i, t_j)$ is the travel time from a_i's location to t_j.
- $\text{Cap}(t_j, q)$ represents the utility function for q agents sharing t_j. All agents sharing a task must be present before task execution can begin, and the amount of time an agent must spend on the task is equal to its allocated fraction of the workload; these portions may be of different sizes. $Cap(t_j, q)$, can represent the minimum required or maximum allowed numbers of agents (by setting capability to 0 for fewer or more agents, respectively), as well as changes in execution quality due to synergies or coordination costs.
- $\pi(CT_i, \Delta w)$ is the penalty for interrupting a_i's current task CT_i having done Δw amount of work on that task, and moving to new task t_j.

Market Clearing Prices and Allocation

Using the polynomial-time algorithm by Devanur et al. [46], the authors compute the market-clearing prices p and allocation matrix X, where:

$$x_{ij} = \text{Fraction of task } t_j \text{ allocated to agent } a_i.$$

The allocation represented by X satisfies envy-freeness and Pareto optimality in the system. Once that the allocation is defined, a scheduling phase begins.

2. Scheduling Phase

This phase refines the allocation by ordering and scheduling tasks for each agent, considering spatial and temporal constraints. It can be considered as a sort of centralized consensus phase where the agents receive new orders for an allocation that is in agreement with the system's constraints.

Task Sequencing

Mainly, for each agent a_i, a new sequencing is defined in order to prioritize tasks allocated in X by maximizing $\text{Cap}(t_j, q)$ to prioritize tasks with high cooperative utility. Additionally, ties are broken in favor of older tasks, imposing a unique ordering across agents that prevents deadlock and avoids task starvation. That is, a situation where multiple agents are stuck because each is waiting for a task held by another agent, creating a circular wait.

Initial Scheduling

The algorithm begins by computing the initial schedule σ_i for agent a_i, ensuring:

$$t'_k - t_k = x_{ik} \cdot w(t_k), \quad t_{k+1} - t'_k \geq \rho(t_k, t_{k+1}),$$

where t_k and t'_k are start and end times for task v_k (namely task t_j allocated to agent a_i in X_{ij}). In other words, the spatial-temporal constraints require that the time spent on each task must equal a_i's assigned share of the workload $w(t_k)$ and that agents must have sufficient time to move between tasks because they can only perform tasks at their current location (illustrated by the distance between the current task t_k and the next task t_{k+1} : $\rho(t_k, t_{k+1})$). Once the spatial-temporal constraints are respected, the authors define an adjusting phase for inter-agent constraints to respect the necessary conditions for agents to cooperate on a task as well as the implications for the subsequent tasks in the schedule.

Adjusting Inter-Agent Constraints

These constraints state that agents cannot cooperate on a shared task if all the assigned agents have not arrived to the task. The first step of the adjustment consists of ensuring all agents arrive before execution begins :

$$\tau_j = \max\{t_k | k \text{ is allocated to } t_j\}.$$

Such actions can have impact on the subsequent tasks of the schedule. Consequently, the authors allow the delay of subsequent tasks if necessary:

$$t_k = \max\{t_{k-1} + \rho(t_{k-1}, t_k), \tau_k\}.$$

This constraint can be solved by monotonically iterating through the sorted times in the initial schedules in $O(mlog(m))$ time, with m being the number of tasks.

Task Reordering

Following the adjustments respecting inter-agents constraints, a task reordering phase is instantiated. This reordering happens because, as stated above, in the schedule of an agent, shared tasks can delay non-shared tasks. Subsequently, the algorithm tries to move the delayed non-shared tasks earlier in the schedule in order to minimize the overall impact of these delays. However, multiple conditions must be met :

- For $k > 1$ (i.e., the shared task is not the first in the schedule):

$$\tau_k^{s_i} - \rho(v_{k-1}^{s_i}, v_k^{s_i}) + x_{k+1}^{s_i} w(v_{k+1}^{s_i}) + \rho(v_{k+1}^{s_i}, v_k^{s_i}) \leq \tau_k^{s_i}.$$

  The components of this equation are:

  - $\tau_k^{s_i}$: the start time of the shared task $v_k^{s_i}$.
  - $\rho(v_{k-1}^{s_i}, v_k^{s_i})$: the travel time between tasks $v_{k-1}^{s_i}$ and $v_k^{s_i}$.

- $x_{k+1}^{s_i}$: the fraction of the non-shared task $v_{k+1}^{s_i}$ allocated to agent s_i.
- $w(v_{k+1}^{s_i})$: the duration of the non-shared task $v_{k+1}^{s_i}$.
- $\rho(v_{k+1}^{s_i}, v_k^{s_i})$: the travel time between the non-shared task $v_{k+1}^{s_i}$ and the shared task $v_k^{s_i}$.

In other words, a non-shared task can be moved earlier if it won't cause any further delays to the shared task that follows. The algorithm evaluates whether shifting the non-shared task earlier would still allow the shared task to start on time, considering factors like travel time and task duration.

- For $k = 1$ (i.e., the shared task is the first in the schedule):

$$t + x_{k+1}^{s_i} w(v_{k+1}^{s_i}) + 2\rho(a_i, v_{k+1}^{s_i}) \leq \tau_1^{s_i}.$$

In this equation:

- t represents the current time.
- $\rho(a_i, v_{k+1}^{s_i})$ represents the travel time from the agent a_i's current location to the non-shared task $v_{k+1}^{s_i}$.
- $\tau_1^{s_i}$ represents the start time of the first shared task $v_1^{s_i}$.

Here the algorithm checks whether the non-shared task can be completed before the shared task is supposed to start.

Following this process, non-shared tasks delayed by shared tasks are reordered to reduce delays without violating constraints. If the schedules are changed, the start times are, again, updated following the exact same logic. Knowing that each task is considered once, this step requires $O(m)$ time. FMC_TA thus runs in worst-case polynomial time to compute allocations and schedule tasks centrally which is a significant result for large scale simulations.

This algorithm shows that FMC_TA effectively and efficiently allocates and schedules tasks for agents in the complex, dynamic settings characteristic of constrained multi-agent system.

3.1.2 Centralized Auction-Based Approaches

Fisher market-based methods and Stochastic Clustering Auction, presented by Zhan et al. [11], share a conceptual foundation in their goal to optimize resource allocation. While Fisher markets leverage equilibrium pricing to allocate divisible goods efficiently, SCA extends this idea to the allocation of discrete tasks by treating task clusters as resources and employing a stochastic optimization approach to solve the allocation. This transition from market-driven dynamics to stochastic clustering

enables SCA to handle complex, heterogeneous environments with dynamic constraints.

Similarly as for the FMC_TA algorithm, the Stochastic Clustering Auction addresses the NP-hard task allocation problem. However, it is now applied to heterogeneous multi-agent teams by viewing task allocation as an optimization problem over task clusters. The goal is to allocate tasks among agents such that a global or regional cost function is minimized. The elements defining the system include:

- $H = \{h_1, h_2, ..., h_k\}$: a set of k heterogeneous agents.
- $T = \{t_1, t_2, ..., t_n\}$: a set of n tasks.
- $A = \{a_1, a_2, ..., a_k\}$: the allocation where $\bigcup_{i=1}^{k} a_i = T, a_i \in T$ and the cluster of tasks a_i is assigned to agent h_i.

In the paper, the authors present a cost associated to the allocation A. This cost is given by either

$$C(A) = \sum_{i=1}^{k} c(a_i) \tag{3.1}$$

or

$$C(A) = max_i c(a_i) \tag{3.2}$$

where $c(a_i)$ is the minimum cost of agent i completing the set of tasks a_i. The problem is to solve the optimization problem $min_A C(A)$. In a way, the cost function in Equation 3.1 can be used to represent the total distance traveled or the total energy used by the agents to achieve this allocation. While the cost function in Equation 3.2 can be used to represent the maximum time taken to accomplish the tasks.

Stochastic Clustering Auction

The authors define the SCA algorithm as a minimization of the cost $C(A)$ using a Markov chain Monte Carlo approach combined with hill-climbing to probabilistically explore the space of possible allocations. Mainly, the idea is to start with an initial allocation A for k clusters and to probabilistically reduce $C(A)$ by rearranging the tasks T among the clusters. The key steps are described in the following paragraphs.

Initial Partition and Cost Approximation

Initially, the set of tasks T is partitioned into k clusters to form an initial allocation $A = \{a_1, a_2, ..., a_k\}$. Each cluster a_i being an unordered subset of T.

Having the an initial partition, it is possible for the agents to approximate the cost function $c_i(a_i)$ (i.e. cost function for agent h_i with tasks a_i assigned) using either

equations 3.1 or 3.2. This cost is submitted to the auctioneer and is considered a bid for the allocation. The auctioneer computes the global cost $C(A)$ again using either equations 3.1 or 3.2 and sets a high temperature T (to which I will refer later on).

Task Reallocation

Having received all the bids, the auctioneer can now reallocate tasks to minimize the global cost $C(A)$ with what the authors introduce as a single move or a dual move.

Single Move: The auctioneer randomly selects a task $t_i \in a_l$ from agent h_l and reassigns this task to agent h_j. This move results in the initial allocation having 2 modified clusters. Then the agents can compute the costs of their new allocation (based on equations 3.1 or 3.2). The likelihood that such a transfer of task t_i from agent h_l to agent h_j is given by

$$P_S(i,j,k,T) = \frac{\exp\left(-\frac{C(A_i^{(j,l)})}{T}\right)}{\sum_{j=1}^{k}\exp\left(-\frac{C(A_j^{(j,l)})}{T}\right)} \tag{3.3}$$

with $C(A_j^{(j,l)})$ being the new global cost the auctioneer computed from the costs the agents computed based on their new allocations. T is the temperature, and the denominator normalizes the probabilities over all potential allocations.

Dual Move (Task Swap): The auctioneer randomly selects two tasks in a_l and a_m, task t_i from agent h_l and task t_j from agent h_m. They are swapped between the agents, resulting in a new allocation A that has two modified clusters. Similarly, agents can compute their costs (based on equations 3.1 or 3.2) and the auctioneer can compute the new global cost. The likelihood of a dual move is given by

$$P_D(i,j,l,m,T) = \frac{\exp(-\frac{C^{(2)}}{T})}{\sum_{p=1}^{2}\exp(-\frac{C^{(p)}}{T})}, \tag{3.4}$$

where $C^{(1)} = C(A)$, the cost before swapping the tasks and $C^{(2)} = C(A_{i,j}^{(l,m)})$, the cost after swapping tasks t_i and t_j in the allocations of agents h_l and h_m respectively.

Acceptance or Rejection

If either equations 3.3 or 3.4 falls into acceptance likelihood, the change is accepted, the allocation A is updated and the global cost $C(A)$ is logged. However, if either

equations 3.3 or 3.4 falls into rejection likelihood, the auctioneer does not update A and returns to the previous step of reallocating tasks.

The auction ends when the termination criteria is satisfied. That is, if $T < T_{cut}$, where T_{cut} is a threshold temperature, then the auction is terminated and the final allocation is $A* = A_i^{(j,l)}$ or $A* = A_{i,j}^{(l,m)}$ with a final global cost of $C(A)*$. Again, if the criteria is not satisfied, the authors reduce $T = \frac{T}{\beta}$ where $\beta > 1$ and start the reallocation step.

The authors show that SCA provides a flexible, easily implementable and efficient method for task allocation in multi-agent systems, accommodating centralized architectures. The use of stochastic optimization ensures near-optimal solutions within practical computational limits. By adjusting the temperature parameter T , the algorithm balances solution quality and runtime, making it adaptable to varying mission requirements.

3.1.3 Conclusion on Centralized Methods

In this section, we have delved into the details of two centralized methods to solve a task allocation problem. These methods rely on a single controller and assume that one of the agents or the *central brain* of the system has full knowledge of the environment by initially knowing it as displayed in sections 3.1.1 and 3.1.2. Each of these methods propose a way of reaching near-optimal solutions for a task allocation problem, either using a fisher market clearing approach or a stochastic exploration approach, and propose original ways of tackling common challenges in multi-agent systems as stated in 2.5. However, the centralized aspect of these methods makes it debatable that centralized multi-agent systems and algorithms properly illustrate real life interactions between entities of heterogeneous nature, as well as the hierarchical relationship of certain agents. One could wonder how such interactions can be modeled and taken into account in problems like dynamic task allocation or load balancing. In order to answer that question and push even further the idea of scalability, fault tolerance and reduced communication overhead, researchers have have explored decentralized methods opening a wide panel of applications and algorithms.

3.2 Decentralized Methods

As stated in chapter 2 dynamic load balancing in Multi-Agent Systems is a crucial challenge in distributed computing, multi-robot coordination, and networked systems. Unlike centralized approaches, which rely on a single authority to allocate tasks or balance workloads, decentralized methods leverage agent autonomy, local decision-making, and inter-agent communication to achieve efficient load balancing.

An important distinction must be made in the following section : cooperative and non cooperative setups. In cooperative game theory, players (agents) can form

binding agreements and collaborate to achieve a common goal. The focus is on coalitions, fair profit distribution, and joint optimization. Solutions includeNash Bargaining Solution (NBS) and Shapley value, ensuring fairness and Pareto-optimality [47, 48]. Alternatively, in non-cooperative games, players make independent and self-interested decisions without binding agreements using Markov Games [49]. Each player maximizes its own utility, leading to strategic competition. The key solution concept is Nash Equilibrium, where no player benefits from unilaterally changing its strategy [50]. Algorithms such as Decentralized Stochastic Algorithm (DSA) leveraging heuristics [49] are used in order to converge to this equilibrium.

Exiting game theory background, cooperative and non cooperative setups can be illustrated as follows:

- Cooperative MAS prioritize shared objectives through structured coordination allowing the system to perform well in dynamic environments. However, each agent doesn't necessarily evolve independently and each information needs to be relayed to every agent, creating challenges in communication overhead and straying from real life scenarios where communications are limited and the system is partially known.

- Noncooperative MAS will model agents that are self-interested optimizing individual rewards and having to approximate their environments by communicating with other agents. This approach reduces communication overhead and lowers the computational complexity of having to aggregate all the cooperative agents information. However, it can limit the system in terms of dynamics because of the competitive agent incentives.

Additionally, some authors consider the agents to be of different nature (if defined by their goal or capabilities) and able to adapt to dynamic behavior which opens the question of inter-agent interaction to solve load balancing [36] and if communication among them really is beneficial. Other studies consider the system as a global auction where each agent is its own auctioneer. In this setup, communication between agents is extremely important and a consensus process allows a near-optimal dynamic allocation of tasks in the system with varying properties [29].

In this section I will address methods using the non-cooperative framework, where agents are of heterogeneous nature with purely local information. I will try to delve into the complexities of methods taking roots in game theory and distributed optimization. I will show that each of these methods contribute to more effective and scalable load balancing in MAS. The strong game theoretic background will only be detailed it when it is directly linked to MAS formulation.

3.2.1 Game Theory-Based Method

Game Theory provides a mathematical framework to model interactions of various nature among autonomous agents in MAS. It is leveraged to analyze how locally self-interested agents can cooperate or compete to achieve a global performance. Specifically, in MAS, game theory is widely used for resource allocation, coordination and decision-making under various constraints.

This section aims to present methods leveraging game-theory in a decentralized fashion to solve the task allocation problem. In a decentralized system, game theory allows agents to negotiate, make independent decisions, and align their strategies to benefit the system globally without a central controller aggregating the local information of each agent.

Chapman et al. [49] propose a novel decentralized technique for planning agent schedules in a dynamic task allocation problem. The contributions of this article are the following:

- Markov Game Formulation: the task allocation problem is formulated as a markov game, where agents act in a stochastic, multi-state environment in order to optimize a global utility function. The dynamic aspect of the environment comes from the fact that tasks have varying deadlines and processing requirements.
- Approximation with Static Potential Games: the authors specify that since solving markov games is computationally challenging, they approximate them using a sequence of static potential games (defined later in this section), allowing an efficient solution.
- Decentralized Solution via the Decentralized Stochastic Algorithm: the paper presents a distribution method for solving approximate games. This method essentially allows agents to coordinate dynamically with limited communication.

Markov Game Formulation

Markov Games are an extension of noncooperative games for repeated interactions in which the game played by the agents at each time-step t, varies probabilistically as a function of the state and the choice of strategies in the previous round [49]. Formally a markov game is a tuple $\Gamma =< N, \Omega, \{\{S_i, u_i\}_{i \in N}\}_{\omega \in \Omega}, q >$. The model defined by the authors comprises:

- A set of states $\omega \in \Omega$, each of which defines a set of tasks $X = \{x_1, ..., x_j, ...\}$, each task has a deadline $t^d_{x_j}$, a number of processing units, y_{x_j}, (number of agents to process the task), and a task utility function, $u_{x_j}(s) : S \longrightarrow R$ (local utility function).

- A set of agents $N = \{1, 2, ..., i, ..., n\}$, each with a strategy space S_i, composed of a sequence of tasks to attend to. Additionally, an agent has a utility function $u_i(s_i, s_{-i}) : S \longrightarrow R$.
- A state transition function $q(\omega^{t+1}|\omega^t)$, and
- A global utility function $u_g(s) : S \longrightarrow R$.

The transition function q describes how new tasks are generated and introduced into the system. When agents attend to such new tasks, their utility function comes into play. It represents the payoff for completing a task. However, certain conditions must be met, i.e. the completion time of the task $t^c_{x_j}(s)$ by agents x_j following a given strategy s, has to be inferior to the hard deadline for the task, and if the correct number of agents attend to this task completing it before the hard deadline. If these conditions are met, the payoff is a discount factor β [1] that incorporates any benefit of completing the task earlier. Otherwise, the utility is 0. The authors specify that tasks incomplete at deadline are equivalent to unattended tasks. Also, having more agents than necessary to complete the task increases the payoff proportionally to the additional number of agents. Finally, the global utility function ranks the overall allocation of tasks to agents and is an aggregation of task utilities :

$$u_g(s) = \sum_{x_j \in X} u_{x_j}(s) \tag{3.5}$$

The authors emphasize that in their model, an agent's strategy space is the set of all permutations of assignments to tasks at each period. That is, a strategy selects an action for each time step for every subsequent state of the environment. That means, an agent's strategy is a set of vectors of actions, with one vector for each state of the environment. Given the real-life problems to which the authors want to apply their method, evaluating and negotiating a set of joint strategies for the number of agents and the size of action vectors of extremely large size, would take too long. Moreover, the system in which agents will be deployed will require them to make decisions over a very short time-frame. This issue illustrates the main difficulty with multi-agent systems and distributed algorithms : computational and timely cost.

To make real-time decision-making computationally feasible, the authors approximate the full Markov game using a sequence of static potential games [2]. Instead

[1] The value of β represents a trade–off between the number of tasks completed and the timeliness of those completed tasks. As the authors aim to maximize the number of tasks completed, they chose a value close to 1, however, if timeliness was their main concern, they would have chosen a lower value.

[2] A potential game is a game where any change in an agent's utility is reflected by a corresponding change in a global potential function. This ensures the game has at least one pure strategy Nash equilibrium and allows for efficient decentralized optimization.

of considering all future states, agents optimize their strategies over a finite horizon of w time steps. This reduces the complexity of computing optimal strategies while still capturing the most relevant decision-making factors.

Each agent's strategy space consists of possible assignments over this limited horizon, making it in agreement with real-world applications where fast decisions are required. The decentralized aspect of the approach is brought by the implementation of the Decentralized Stochastic Algorithm to allow agents to negotiate and optimize their actions dynamically. The authors show that this approach ensures that the system is robust to communication constraints and can adapt to partial observability.

Distributed Stochastic Algorithm

The Decentralized Stochastic Algorithm [51] is used in the paper as a decentralized, local search method for optimizing task allocation in a multi-agent system. Since the task allocation problem is approximated as a sequence of potential games, DSA ensures that agents can make independent yet coordinated decisions, leading to globally efficient outcomes. The DSA operates iteratively, allowing agents to adapt their task assignments dynamically by making local improvements. The authors take into account limited range view in order for the agents to base themselves only on local observation. The agents can only observe tasks within a radius r. The agent can only communicate with other agents within r, and an agents' utility is computed only over visible tasks. The authors slightly modify the DSA so that each agent makes independent decisions while following the potential game structure ensuring convergence of the algorithm. Also, the information spreads indirectly through agent interactions allowing the system to function even with limited connectivity.

The algorithm the authors propose is detailed in Algorithm 1, I have taken the liberty to simplify some steps for the sake of the reader. The overall structure remains in agreement with the original paper [49].

In this algorithm, the probability p is tuned to balance exploration and stability. After the for-loop, a process of recycling is done by using the previous assignment as a starting point for the next, allowing to reduce computation time and quickly adjust to new tasks (with potentially pressing deadlines). It then removes any completed or expired tasks to avoid unnecessary reallocation.

This method proves itself very efficient because of it being decentralized allowing agents to make independent decisions, the probabilistic agent activation with p prevents oscillations and instability to newly arriving tasks, additionally, the potential game approximation setup guarantees a convergence to a Nash equilibrium with DSA. In terms of application, authors find that this method is as efficient as centralized methods in unlimited range of view, but when the communication and view are limited to a certain radius, the proposed method outperforms the centralized ones. This performance illustrates the benefits of decentralized methods allowing

Algorithm 1 Distributed Stochastic Algorithm (DSA) for Task Allocation

Input: Set of agents N, set of tasks X, time horizon w, activation probability p
Initialize: Each agent $i \in N$ selects an initial strategy s_i (sequence of tasks over w steps allocated to agent i)
while Tasks remain uncompleted **do**
    for each agent $i \in N$ **in parallel do**
        With probability p, agent i is activated
        if agent i is activated **then**
            Compute current utility $u_i(s_i, s_{-i})$
            Determine best-response strategy s_i^{new} that maximizes u_i
            if $u_i(s_i^{\text{new}}, s_{-i}) > u_i(s_i, s_{-i})$ **then**
                Update strategy: $s_i \leftarrow s_i^{\text{new}}$
            end if
        end if
    end for
    Advance time step: $t \leftarrow t + 1$
    Shift decision window: retain last $w - 1$ steps, recompute next step
    Remove completed or expired tasks from X
end while
Output: Optimized task allocation strategy for all agents with Nash equilibrium.

agents to efficiently take decisions and approximate their global environment while remaining computationally viable with a short time frame for decision-making.

3.2.2 Distributed Optimization-Based Methods

Dynamic load balancing in Multi-Agent Systems is crucial for optimizing resource utilization and ensuring efficient task distribution among agents. Distributed optimization methods (DOM) have been extensively studied to address this challenge, enabling agents to make decentralized decisions while achieving global objectives. Even though DOM and the previously introduced game-theoretic methods 3.2.1 both aim to achieve effective dynamic load balancing they differ fundamentally in their underlying principles.

As previously stated, game-theoretic methods model the interactions among agents as strategic games, where agents act based on their own interests, which may or may not align with a global objective. Alternatively, DOM focus on collaboratively minimizing a global cost function such as the time taken per jobs, or maximizing a global utility function such as the throughput of completed jobs [36]. The primary goal is to find an optimal solution that benefits the entire system. For instance, consensus-based algorithms [29] enable agents to agree on certain variables' values to achieve system-wide optimization. These methods are typically designed

for cooperative scenarios where agents are willing to share information and work towards a collective goal.

In this section I will present methods that propose a robust dynamic load balancing. Some proofs linked to articles will only be cited but not detailed as it does not bring any true value to my work.

Consensus-Based Decentralized Auctions

Choi et al. [29] present a novel approach to decentralized task allocation for autonomous vehicles. The paper introduces not one but two algorithms : Consensus-Based Auction Algorithm (CBAA) for single-task assignments and its extension, Consensus-Based Bundle Algorithm (CBBA), for multi-task assignments. These two algorithms merge auction-based market mechanisms for task selection with a decentralized consensus process for conflict resolution resulting in a conflict free allocation. In addition to presenting the two algorithms, the authors also propose a formal proof of convergence and worst-case performance guarantees, as well as a comparison with existing sequential auction algorithms, showing that their method outperforms them in terms of convergence speed and assignment efficiency.

CBAA - Definition

The first algorithm, CBAA, solves the single-assignment problem. That is, where a fleet of N_u agents must assign N_t tasks such that :

- Each task is assigned to at most one agent (conflict free assignment)
- Each agent is assigned to at most one task

The problem is formulated as the following integer optimization problem:

$$\max \sum_{i=1}^{N_u} \sum_{j=1}^{N_t} c_{ij} x_{ij} \tag{3.6}$$

where:

- x_{ij} is a binary variable (allocation vector) indicating if task j is assigned to agent i
- c_{ij} is the reward agent i receives from completing task j

As per the algorithm's definition, the constraint of single task assignment per agent must be respected $\sum_{j=1}^{N_t} x_{ij} \leq 1, \forall i \in I$, each element in the allocation matrix has to be less or equal to one i.e. each agent can select at most one task. A similar constraint on the tasks is formulated with $\sum_{i=1}^{N_u} x_{ij} \leq 1, \forall j \in J$. Additionally, the number of tasks allocated is bounded by the system's constraints $\sum_{i=1}^{N_u} \sum_{j=1}^{N_t} x_{ij} =$

$N_{min}, N_{min} = \min(N_t, N_u)$, which makes sense. In fact, you cannot allocate more tasks (resp. agents) than there are available tasks (resp. agents).

CBAA - Algorithm

The algorithm operates through two iterative phases until all tasks have been assigned respecting the constraints, resulting in a conflict free assignment. The first phase is called Auction Process, where each agent independently bids for the task that maximizes its local reward. The winning bid is stored and updated iteratively. Each agents maintain two vectors: $x_i \in \{0,1\}^{N_t}$ agent i's task list, and $y_i \in R^{N_t}$ the winning bid list (which is always kept up-to-date). The first phase is described in Algorithm 2

Algorithm 2 CBAA - Phase 1: Auction Process (Task Selection)

Require: c_{ij}: Task bid for agent i, $\mathbf{y}_i$: Winning bid list, $\mathbf{x}_i$: Task assignment list

1: Initialize $\mathbf{x}_i \leftarrow \mathbf{0}$, $\mathbf{y}_i \leftarrow \mathbf{0}$
2: **if** $\sum_j x_{ij} = 0$ **then** ▷ Check if agent is unassigned
3: **for** $j \in J$ **do**
4: $h_{ij} \leftarrow \mathbb{1}(c_{ij} > y_{ij})$ ▷ Identify valid tasks
5: **end for**
6: **if** $\sum_j h_{ij} > 0$ **then** ▷ If there are valid tasks
7: $J_i \leftarrow \arg\max_j (c_{ij} \cdot h_{ij})$ ▷ Select best task
8: $x_{iJ_i} \leftarrow 1$ ▷ Indicate task J_i assigned to agent i
9: $y_{iJ_i} \leftarrow c_{iJ_i}$ ▷ Update bid
10: **end if**
11: **end if**

Algorithm 2 shows the procedure of agent i's phase 1 at iteration t, where one iteration consists of a single run of phase 1 and phase 2. Authors specify that each agent's iteration count can be different, which allows for the possibility that each agent has different iteration periods. An unassigned agent i (equivalently, an agent with $\sum_j x_{ij}(t) = 0$) first computes the valid task list h_i, where $h_{ij} \in \{0,1\}^{N_t}$. The intuition is, if there are bids on a task j that are higher than the winning bid list for task j, then it is a valid task for agent i, because he could claim it. If there are valid tasks, it then selects a task J_i giving it the maximum score based on the current list of winning bids (line 7 of Algorithm 2), and updates its task x_i and the winning bids list y_i accordingly. That is, set $x_i j = 1$ (resp. $y_{ij} = c_{iJ_i}$) when task J is being assigned to agent i. Also, in the case that the agent has already been assigned a task $\sum_j x_{ij}(t) \neq 0$, this selection process is skipped, and the agent moves to phase 2.

The second phase of the CBAA algorithm is the consensus process, where each agent exchanges its winning bid list with its direct neighbors and updates its bid values to reach an agreement on the highest bid. The neighbors are defined by a

graph $G(t)$ representing the undirected communication network at time t with a symmetric adjacency matrix such that $g_{ik} = 1$ if a link exists between agent i and k at time t and 0 otherwise. The authors specify that they include self connections, i.e $g_{ii} = 1, \forall i$. Upon receiving information from a neighbor k, agent i will, for each task j, update his winning bid list with what agent k bid on task j. Then updates the winner agent z_{ij} (line 5 of algorithm 3, index of the agent that holds the highest bid for task J_i). Finally, if the winner for task J_i isn't the agent i, he unallocates himself in his allocation vector (line 7 of algorithm 3). If all the tasks are not allocated, phase 1 starts again. The second phase of the algorithm is described in Algorithm 3.

Algorithm 3 CBAA - Phase 2: Consensus Process (Conflict Resolution)

Require: $\mathbf{y}_i$: Winning bid list, $\mathbf{x}_i$: Task assignment list, G: Communication network

1: Send $\mathbf{y}_i$ to all neighbors k where $g_{ik} = 1$
2: Receive $\mathbf{y}_k$ from all neighbors k
3: **for** $j \in J$ **do**
4: $\quad y_{ij} \leftarrow \max_k(g_{ik} \cdot y_{kj})$ ▷ Max-consensus update
5: $\quad z_{ij} \leftarrow \arg\max_k(g_{ik} \cdot y_{kj})$ ▷ Determine winner
6: $\quad$ **if** $z_{iJ_i} \neq i$ **then** ▷ If outbid, release task
7: $\quad\quad x_{iJ_i} \leftarrow 0$
8: $\quad$ **end if**
9: **end for**

As a fail-safe, the authors assume that ties occurring in determining J_i in phase 1 or z_{iJi} in phase 2 are resolved in a systematic way. For example, a lexicographical tiebreaking heuristic based on the agent and the task identification numbers can be used.

CBBA - Definition

After defining the CBAA, the authors introduce the Consensus-Based Bundle Algorithm, which extends the CBAA to handle multi-task assignment for decentralized task allocation in MAS. The main difference being that CBBA allows an agent to bid on multiple tasks as a sequence (bundle). This method still ensures a conflict-free allocation. Consequently, the formulation of the problem is slightly different :

$$\max \sum_{i=1}^{N_u} \sum_{j=1}^{N_t} c_{ij}(x_i, p_i) x_{ij} \tag{3.7}$$

where:

- x_{ij} is a binary variable (allocation vector) indicating if task j is assigned to agent i

- $c_{ij}(x_i, p_i)$ is the reward function based on agent i's task assignment x_i and the execution sequence (path) p_i
- p_i the path, is an ordered sequence of tasks assigned to agent i

With these new elements, the logic of the constraints remain the same. Each agent can handle at most L_t tasks (system parameter the user can change) $\sum_{j=1}^{N_t} x_{ij} \leq L_t, \forall i \in I$. Similarly to CBAA, each task is assigned to at most one agent. Finally, the total number of assigned tasks is bounded for the same logical reasons stated in 3.2.2: $\sum_{i=1}^{N_u} \sum_{j=1}^{N_t} x_{ij} = N_{min}, N_{min} = \min(N_t, N_u L_t)$.

CBBA - Algorithm

The CBBA also consists of two iterative phases. The first being the bundle construction where each agent greedily (highest reward first), and independently builds a sequence of tasks based on local bid values. The second phase is the consensus phase where agents exchange their bid values and winning agent list with neighbors. Tasks with conflicts are released and agents adjust their bundle accordingly. In this section I will go in the detail of each of those phases and discuss how it produces a robust and dynamic task allocation solution.

Similarly, CBBA has knowledge of y_i the winning bid list, storing the highest bids observed for each task. However, the first phase of CBBA also introduces three new elements (the path p_i described earlier being one of these elements) :

- b_i: bundle of tasks assigned to agent i, stored in the order they were selected
- z_i: winning agent list, indicating which agent currently holds each task

Each agent builds a bundle by iteratively adding tasks that provide the highest marginal gain:

$$c_{ij}[b_i] = \max_{n \leq |p_i|} S_i(p_i \bigoplus_n j) - S_i(p_i) \tag{3.8}$$

where $S_i(p_i)$ is the total score of agent i for its current path p_i and $S_i(p_i \bigoplus_n j)$ represents the total score when inserting task j at position n in the path to maximize gain. The agent selects the task that maximizes its marginal gain defined in 3.2.2 subject to bid constraints:

$$h_{ij} = \mathbb{1}(c_{ij} > y_{ij}), J_i = arg\max_j(c_{ij}.h_{ij}) \tag{3.9}$$

Meaning that the task must be valid, i.e. the bid agent i makes on task j must be higher than the actual winning bid on task j. And that J_i is the task that has the highest bid in the scalar product of $c_{ij}.h_{ij}$ (which is a matrix of $N_t \times N_u$ indicating the reward for completing valid task j for agent i). After selecting the tasks, the

bundle is updated recursively:

$$b_i \longleftarrow b_i \bigoplus_{end} \{J_i\}, p_i \longleftarrow p_i \bigoplus_{n_{i,J_i}} \{J_i\} \tag{3.10}$$

Algorithm 4 CBBA - Phase 1: Bundle Construction (Task Selection)

Require: c_{ij}: Task bid for agent i, $\mathbf{y}_i$: Winning bid list, $\mathbf{z}_i$: Winning agent list, $\mathbf{b}_i$: Bundle of tasks, $\mathbf{p}_i$: Path sequence

Initialize $\mathbf{y}_i \leftarrow \mathbf{0}$, $\mathbf{z}_i \leftarrow \emptyset$, $\mathbf{b}_i \leftarrow \emptyset$, $\mathbf{p}_i \leftarrow \emptyset$
while $|\mathbf{b}_i| < L_t$ **do** ▷ Check if the bundle is full
  for $j \in J \setminus \mathbf{b}_i$ **do** ▷ Loop over unassigned tasks
    Compute marginal gain: $c_{ij}[\mathbf{b}_i] = \max_{n \leq |\mathbf{p}_i|} S_i(\mathbf{p}_i \oplus_n j) - S_i(\mathbf{p}_i)$
    Compute eligibility: $h_{ij} = I(c_{ij} > y_{ij})$
  end for
  if $\sum_j h_{ij} > 0$ **then** ▷ If valid tasks exist
    $J_i \leftarrow \arg\max_j (c_{ij} \cdot h_{ij})$ ▷ Select task with highest gain
    Determine optimal insertion position: $n_{i,J_i} = \arg\max_n S_i(\mathbf{p}_i \oplus_n J_i)$
    Update bundle: $\mathbf{b}_i \leftarrow \mathbf{b}_i \oplus_{\text{end}} \{J_i\}$
    Update path: $\mathbf{p}_i \leftarrow \mathbf{p}_i \oplus_{n_{i,J_i}} \{J_i\}$
    Update winning bid: $y_{i,J_i} \leftarrow c_{i,J_i}$
    Update winning agent: $z_{i,J_i} \leftarrow i$
  else
    break ▷ Exit if no valid tasks remain
  end if
end while

where n_{i,J_i} is the optimal insertion position and $\bigoplus_{end}$ means the task J_i is added at the end of the bundle. This process continues until the bundle reaches its limit $|b_i| = L_t$ or no valid tasks remain. Algorithm 4 describes the process of the first phase.

The second phase of CBBA consists of solving conflicts and re-iterating the first phase until a consensus is reached. Following a network defined by a undirected graph G, agents communicate three pieces of information with their neighbors (according to the adjacency matrix of the graph, similarly to CBAA): the winning bid list y_i, the winning agent list z_i, and a timestamp vector s_i, tracking the last update time. With the information received, each agent will compare bid values with its neighbors and apply a rule that updates y_i with the highest bid among the neighbors for task j, and update z_i with the winning agent of task j. To resolve conflicts, CBBA applies a set of update rules: (i) if an agent receives a higher bid for a task it previously won, it releases the task and updates its bundle accordingly, (ii) if conflicting information about a task's winning agent exists, the agent follows a max-consensus rule, prioritizing the highest observed bid and most recent update thanks to the timestamp information s_i, and (iii) in cases of ties, a predefined priority mechanism ensures deterministic resolution (the table of rules is detailed in

the article [29]). If another agent k has a higher bid, then agent i releases task j by removing it from b_i and p_i. Additionally, any tasks selected after j in the bundle are removed to maintain consistency. From here, the algorithm returns to the first phase, and new tasks are added ensuring all agents to reach an agreement on task allocations.

CBBA - Convergence

Authors prove that CBBA guarantees convergence in at most DN_{min} iterations, where: D is the diameter of the communication network (i.e. the longest shortest path between agents), and N_{min} is the number of assigned tasks. According to the theorem presented in the article, if the network is connected and every agent has a consistent scoring function, CBBA converges to a conflict-free assignment within : $T_C \leq DN_{min}$. Proof is detailed in the original paper. Moreover, the authors also demonstrate that CBBA guarantees a solution that is at least 50% optimal compared to the best centralized solution. Compared to previous consensus-based methods, agents do not need to share identical Situational Awareness (SA). In fact, only the bid values y_i are shared, so even if two agents perceive different environments, the auction mechanism inherent to each agent naturally selects the best bids without requiring SA synchronization in a centralized way [52, 53]. Authors show that the same results are obtained with CBAA in terms of convergence and optimality. In fact, applying CBBA with $L_t = 1$ results in executing CBAA.

CBBA - Efficiency

Unlike traditional consensus-based methods that require full situational awareness agreement, CBBA only relies on bid exchanges, making it robust to communication constraints and inconsistencies in environmental perception. This decentralized and computationally efficient approach makes CBBA well-suited for dynamic, distributed multi-agent task allocation problems, such as robotic coordination, Unmanned Aerial Vehicle (UAV) fleet management, and distributed sensor networks. Compared to traditional auction algorithms such as Sequential Auctions [54] and Prim Allocation [55], where tasks are allocated one at a time and rely on a single auctioneer, CBBA is computationally more efficient and decentralized allowing for asynchronous task allocation. And compared to other decentralized methods such as Implicit Coordination [56] and ETSP-ASSIGNMT [57] that require consistent situational awareness across all agents that make it difficult to maintain in real-world scenario with communication limitations, CBBA outperforms these methods by requiring only bid exchanges across a simple network and using a max-consensus approach achieving faster convergence while remaining consistent. Finally, when

compared to centralized task allocation strategies such as Mixer Integer Linear Programming (MILP) [58] that provide globally optimal solutions but are intractable for real-time MAS, CBBA, though suboptimal in the worst CBBA, achieves near optimal performance in practice while being computationally viable and scalable to large numbers of agents and tasks.

The results demonstrate that CBBA outperforms existing decentralized auction-based methods in terms of convergence speed, computational efficiency, and robustness to communication failures. Numerical simulations confirm that CBBA achieves near-optimal assignments while maintaining a low computational burden. Although it can be debated that homogeneous agents do not illustrate well real-life scenarios, research could be made on extending CBBA to heterogeneous agent capabilities and incorporate learning-based task allocation instead of having deterministic rules predefined. In fact, I can speculate that CBBA would not perform well in adversarial or uncertain environments. Which is why, I will introduce a technique that incorporates heterogeneous agents able to follow adaptive or non adaptive selection rules for load balancing in the next section.

Adaptive Load Balancing

Previously introduced auction-based methods rely on deterministic procedures and a set of rules that eventually converges to conflict free task allocation among a fleet of agents. In spite of being scalable and computationally viable, these methods only perform well in environments that are stable and predictable. To delve further into decentralized systems being adaptive, robust, and in agreement with real-life scenarios constraints, authors such as Schaerf et al. [36] propose a framework called Multi-Agent Multi-Resource Stochastic System (MAMRSS) system to address the problem of load balancing. This framework involves a set of agents, a set of resources, probabilistically changing resource capacities, probabilistic assignment of new jobs to agents, and probabilistic job sizes. An agent must select a resource for each new job, and the efficiency with which the resource handles the job depends on the capacity of the resource over the lifetime of the job as well as the number of other jobs handled by the resource over that period of time. They show that performance measure for the system aims at globally optimizing the resource usage in the system while ensuring fairness (that is, a system shouldn't be made efficient at the expense of any particular agent). This section aims to present this method that takes its roots in the reinforcement learning framework. However, in the paper, the authors give close to no detail about the actual RL algorithm they used. I will try to clarify this for the reader.

Multi-Agent Multi-Resource Stochastic System

Before introducing the framework, it is important to state the interpretation of reinforcement learning and load balancing the authors adopt. The model they adopt is inspired of models used in distributed AI and organization theory. Namely, they assume a strict separation between agents and resources. Jobs arrive to agents who make decisions about which resource they will move to in order to execute the jobs. To illustrate their problem, the authors mention a problem introduced by Arthur et al. [59] (inspired by the El Farol bar problem): An agent, embedded in a multi-agent system, has to select among a set of bars (or a set of restaurants). Each agent makes an autonomous decision but the performance of the bar (and therefore of the agents that use it) is a function of its capacity and of the number of agents that use it. The decision of going to a bar is a stochastic process but the decision of which bar to use is an autonomous decision of the respective agent. The model presented is a general model where such situations can be investigated. In these situations a job arrives to an agent who decides upon the resource (e.g., restaurant) where his job should be executed; there is a-priori no association between agents and resources since agents rely on purely local information.

Formally, the Multi-Agent Multi-Resource Stochastic System, is defined as a 6-tuple:

$$M =< A, R, P, D, C, SR >$$

where:

- $A = \{a_1, a_2, ..., a_N\}$ is the set of agents (decision-makers).
- $R = \{r_1, r_2, ..., r_M\}$ is the set of resources (job execution sites).
- $P : A \times \mathbb{N} \longrightarrow [0, 1]$ is a job submission function where $P(a, t)$ is the probability that agent a submits a job at time t.
- $D : A \times \mathbb{N} \longrightarrow \mathbb{R}^+$ is a job size function where $D(a, t)$ gives the size of the job submitted at time t.
- $C : R \times \mathbb{N} \longrightarrow \mathbb{R}^+$ is a probabilistic resource capacity function where $C(r, t)$ denotes the available capacity of resource r at time t.
- SR is a resource selection rule that determines how agents assign jobs to resources.

Intuitively, each of the resources has a certain capacity, which is a real number; this capacity changes over time, as determined by the function C. At each time point each agent is either idle or engaged. If it is idle, it may submit a new job with probability given by P. Each job has a certain size which is also a real number. The size of any submitted job is determined by the function D. (the authors will

use the unit "token" when referring to job sizes and resource capacities). For each new job the agent selects one of the resources according to the selection rule SR. The authors emphasize that any job can be run on any resources, and that there is no limit on the number of jobs served simultaneously by a given resource, meaning no queuing occurs. This last aspect makes the framework stray from real-life applications where, given the a system, queuing has to occur if for some reason only one resource remains available after the others fail (e.g. a server cannot endlessly process jobs with no time cost). Although it can be argued that given a selection rule SR agents would deliberately stop submitting jobs to the last standing resource in order to avoid overflowing it, the global system performance would decline, such assumption reduces significantly the panel of applications.

The job execution model is quite straightforward. Each resource has a capacity $C(r,t)$ at time t, and it must distribute this capacity among all currently executing jobs. If resource r is serving k jobs at time t, each job receives an equal share of the capacity capacity per job $= \frac{C(r,t)}{k(t)}$. The execution time T_j of job j with size S_j assigned to resource r is given by:

$$T_j = \sum_{t=t_{start}}^{t_{stop}} \frac{S_j}{\text{capacity per job}(t)}$$

where t_{start} is the time the job was submitted, t_{stop} is the time the job completes, and capacity per job(t) is the capacity allocated to a job at time t.

The system's objective is to minimize the average execution time per token (or maximize the system throughput) defined as $\mathbb{E}[\frac{T}{S}]$, where T is the total execution time, and S is the total number of tokens processed.

Multi-Agent Reinforcement Learning Framework

The MARL framework [3] is expressed through the local information each agent carries and the selection rule SR. In fact, each agent maintains an efficiency estimator ee_A, which represents its learned assessment of each resource's effectiveness. In addition to ee_A, agent A keeps a vector $jd_A \in \mathbb{N}^{|R|}$, which stores the number of completed jobs which were submitted by agent A to each of the resources, since the beginning of time. The efficiency estimator is defined as:

$$ee_A = \{ee_A(r_1), ee_A(r_2), ..., ee_A(r_M)\}$$

[3] In this article [36] the authors give close to no details on the implementation of their so called "MARL" framework and the technology used to lead the experiments.

where $ee_A(r)$ represents the estimated performance of resource r. It is updated each time a job assigned to resource r is completed:

$$ee_A(r) \longleftarrow WT + (1 - W)ee_A(r)$$

where:

- W is a dynamic learning rate parameter.
- T is the time per token for the job, computed as :

$$T = \frac{t_{stop} - t_{start}}{S}$$

W is a dynamic learning rate, in fact it is resource-specific and is defined as:

$$W = w + \frac{(1 - w)}{jd_A(r)}$$

where w is a constant that controls how much recent experiences influence learning. The term $\frac{(1-w)}{jd_A(r)}$ is a correcting factor, which has a major effect only when $jd_A(r)$ is low; when $jd_A(R)$ increases, reaching a value of several hundreds, this term becomes negligible with respect to w. After defining what the agent effectively learns throughout the execution, the authors define how resources are selected based on ee_A and jd_A.

As specified in the definition of the MAMRSS, the resource selection probability $pd_A(r)$ determines how likely an agent is to choose a particular resource. The authors first define a derived version of this function : $pd'_A(r)$

$$pd'_A(r) = \begin{cases} ee_A(r)^{-n} & \text{if } jd_A(r) > 0 \\ E[ee_A]^{-n} & \text{if } jd_A(r) = 0 \end{cases} \tag{3.11}$$

Here, $n \in \mathbb{R}^+$ and $E[ee_A]^{-n}$ the average of the values of $ee_A(r)$ over all resources with more than 0 jobs done. The authors define the submission probability function as the normalized version of $pd'_A(r)$:

$$pd_A(r) = \frac{pd'_A(r)}{\sigma}, \sigma = \sum_r pd'_A(r)$$

[4]

How the function pd_A is constructed makes it highly biased towards resources that have performed well in the past (the order of magnitude of pd'_A defined in Equation 3.11 is much more important when $jd_A(r) > 0$). The strength of the bias

[4]The use of the normalization factor σ is because if no jobs have been completed over all the resources, the choice is random with a uniform probability distribution

comes from n which is the exploration factor. Higher values of n mean stronger exploitation (choosing historically best resource), while lower values of n encourages more exploration. Although this bias of always choosing the historically most efficient resource seems like cheating, it is in general not a good policy; it does not allow agents to exploit changes, or improvements in the capacity or load on other resources.

To summarize the MARL framework, the authors defined a MAMRSS in which agents use a family of adaptive resource selection-rules, parametrized by a pair (w, n). These parameters serve as knobs to tune and optimize the system's performance.

Experimental Setup

In the article, the authors define an experimental setup where they compare adaptive selection rules (defined in the previous paragraph) to non-adaptive selection rules (agents assigned to resources), and a load-querying selection rule where agents query each resource and submit the job to the less crowded one. The simulation is setup as such:

- Number of agents $N = 100$
- Number of resources $M = 5$

The 5 resources have, at first, a fixed capacity : $[c_1 = 40, c_2 = 20, c_3 = 20, c_4 = 10, c_5 = 10]$, c_i being the capacity of resource i. The authors are particularly motivated to be as close as possible with real situations (for such a setup can be derived into many situations). Hence, they assume that each point in time corresponds to a second and count the time in minutes, hours (point of reference at which changes in the system will occur), days, and weeks. The probability of submitting a job at each second, which corresponds to the load of the system, can vary over time; this is the crucial factor to which the agents must adapt. Agents can submit jobs at any second, but the probability of such submission may change. In particular the article concentrates on three different values of this quantity:

- Low Load (L_{lo})=0.1%
- High Load (L_{hi})=0.3%
- Peak Load (L_{peak})=1%

The authors also assume that the job size function is a uniform distribution over the integer range $[50, 150]$. Finally, the performance metric is the execution time per token T defined earlier.

Performance Findings

First, the experiments demonstrate that adaptive resource-Selection Rules (SR) significantly improve system performance compared to non-adaptive approaches. While fixed resource allocations perform well in static environments, they fail to adjust to dynamic changes in load and resource availability. Adaptive SR, by contrast, allow agents to respond effectively to fluctuating conditions. When the system is at its peak load L_{peak}, adaptive SR outperform fixed allocations by reducing average execution time per token by 5 to 10%, in dynamic environments where load varies unpredictably, adaptive SR offer an efficiency gain of up to 20% over static strategies. However, the load-querying SR achieves the best possible efficiency, but require real-time communication, which is not always possible in uncertain of partial environments.

Moreover, the study identifies that certain values of the model's parameters w (learning rate) and n (exploration factor) will lead to different conclusions:

- A moderate level of exploration ($n = 3$ to 5) achieves the best performance, ensuring adaptation while avoiding excessive inefficiency.
- Higher values of w leads to faster adaptation in dynamic settings, while low values of w cause agents to respond slowly.
- If n is set too high, resources a overwhelmed because agents choose the historically best resources, as expected.
- If n is set too low, the decision-making is inefficient as agents fail to use best resources.

These findings perfectly illustrate the trade-off between exploration and exploitation that is a common issue in reinforcement learning.

Then, the authors assess the effect of dynamic load conditions where the load is periodically changed every hour, and another scenario where the load changes in random fluctuations. The article shows that adaptive SR always outperform the static SR, showing that adaptive reinforcement learning is globally effective in handling uncertain environments.

In this work, I previously mentioned the study of interaction between agents of different nature as it is an important factor to take into consideration in MAS. Here the article considers the non-cooperative setup literally and uncover a "parasite effect":

- Agents that engage in less exploration benefit from the exploration effort of others. Such result does not necessarily illustrate the fairness researchers would seek in a MAS.

- Exploitative agents achieve lower execution times at the expense of cooperative agents.
- When all agents behave in their own interest (pure exploitation, n high), the system performs worse overall, demonstrating a Prisoner's Dilemma effect (resonates with the findings in game-theory based methods).

Further into agent interaction, I also need to mention inter-agent communication. Up to this point of the study, the authors have assumed that there was no direct communication among the agents, which is in agreement with the MARL framework where interesting results occur when real-life forces agents to act without a-priori arranged communication channels and only rely on feedback mechanism and local information. However, the authors wished to observe the effect of agents with augmented capabilities such as a naive communication on the system efficiency.

The authors introduce the concept of neighborhood in which agents, with the reflexive, symmetric, and transitive relation *neighbor of*, can share information. This relation partitions the population into neighborhoods. And in such neighborhoods, when a decision is made (i.e. when an agent chooses a resource), the average of the neighborhood efficiency estimator ee_A is shared. That is, the average of ee_A among all agents in a same neighborhood. While neighborhoods may not follow the same selection rule SR, agents within a same neighborhood all follow the same SR. The results of the experiment with Communicating Neighborhoods (CN) and Non-Communicating Neighborhoods (NCN) are the following:

- Large communication neighborhoods (20+ agents) tend to be too conservative relying on the best resource only. Since the communicated information is the average ee_A, the perception of the system is too static. For values of w and n that give the best results in non communicating neighborhoods, they give poor results in communicating neighborhoods in separate experiments.
- When confronting communicating and non communicating neighborhoods of large size, both display this conservative aspect using the best resource, but CN outperform NCN because they use it in a *smarter* way due to the fairer picture of the environment provided by the communication mechanism.

These results lead to the conclusion that communication may not provide useful means to improve the performance of the system in the article setting. However, it remains interesting to see that CN outperform NCN when confronted with the exact same setting.

3.2.3 Conclusion on Decentralized Methods

In this section I have presented decentralized methods leveraging agent autonomy, local decision-making and agent interaction. We've seen that some methods use of

game-theoretic background in section 3.2.1 considering setups as non-cooperative games to form allocations and then using stochastic algorithms to optimize these allocation while ensuring a Nash Equilibrium and fairness among agents such that none would benefit from changing its final allocation. Other methods leverage distributed optimization means in Section 3.2.2 considering the system as a global auction where each agent is its own auctioneer and buyer. These methods also introduce naive communication mechanisms, that don't always benefit the system globally, but illustrate how complex systems could benefit from such capabilities. The proposed decentralized methods are shown to be more efficient than their centralized analog and much more robust to uncertain environments. Which, in real-life scenarios, is preferable since we cannot always ensure the environment stability and integrity. They also incorporate agents of different nature to illustrate cooperation, or conservatism, to better understand the system.

In the next section, we will see the Centralized Training and Decentralized Execution that leverages a centralized training phase, where agents learn optimal policies using global knowledge, but during execution, they operate independently based on local observations.

3.3 Centralized Training, Decentralized Execution

Deep Reinforcement Learning (DRL) is a very famous area of artificial intelligence and having been applied mostly to single-agent settings, it has achieved outstanding success in a wide range of applications. In the context of load balancing or task allocation, DRL involves more than one independent learner. As we have seen before, it is important for each agent to collaborate jointly to maximize a global metric or ensure fairness among agents. Researchers have attempted to tackle Multi-Agent Deep Reinforcement Learning (MADRL) problems by using single-agent DRL algorithms [60]. Despite promising results the main issue pointed out by the community was the *environment non-stationarity* that goes against stable DRL methods because individual agents can no longer perceive their environment as being stationary since it is also influenced by other agent's activity. This issue has promoted the widely used paradigm known as Centralized Training and Decentralized Execution [61, 62]. Moreover, mechanisms such as agent-communication that my work introduced in the previous section as well as policy-gradient techniques adapted to MAS [63] have been proposed to handle the common situation where every agent only has a partial view of their learning environment and other agents.

In this section, I will present methods using the CTDE paradigm. I will emphasize on methods leveraging reinforcement learning and deep learning and present the advantages each of the methods get from using the CTDE paradigm and how it

represents an overall advantageous approach for tackling challenges linked to solving problems in multi-agent systems.

3.3.1 Centralized-Decentralized Reinforcement Learning

As discussed in Section 3.1 addressing centralized methods, Fisher market-based or auction-based methods have been extensively utilized in load balancing for their ability to model resource allocation as a competitive equilibrium problem. These methods perform well in distributed systems by treating resources as divisible goods and users or tasks as buyers with budgets or bids, aiming to maximize utility through market-clearing prices for FMC_TA or stochastically minimizing the cost of allocations throughout the system for SCA. While effective in scenarios with well-defined preferences and budgets, these methods often fall short in dynamic environments where workloads and resource conditions fluctuate unpredictably or when the system needs to be deployed on a larger scale, these sequential methods do not yield the best results, where decentralized methods introduced in Section 3.2 are able to adapt to dynamic and varying environments.

Centralized reinforcement learning addresses these limitations by replacing static equilibrium-driven approaches with adaptive, trial-and-error-based decision-making frameworks. Unlike Fisher market models or auction models, RL agents dynamically learn optimal policies by interacting with the system, continuously updating their strategies to reflect real-time changes making the agents much more robust not relying on heuristic strategies (e.g. hill-climbing) but purely on the environment state and their previous experience. This adaptability makes RL-based frameworks particularly suitable for cloud computing, or when interacting with fleets of robots, where workloads are inherently volatile, the environment changes, and system metrics evolve rapidly. Moradi et al. [64] introduces a novel approach called Centralized Learning Distributed Scheduling (CLDS) for job scheduling in Grid computing environments. This paper proposes a novel framework for multi-agent job scheduling in Grid computing. CLDS effectively integrates centralized learning with distributed scheduling [65] to address critical challenges such as load balancing, fault tolerance, scalability, and adaptiveness in heterogeneous and dynamic grid environments.

The main contribution of this method is that it employs a single modular learner agent to manage a global utility table, which is used to assess the efficiency of available resources. This modular learner, if failing, can be replaced by any other agent in the system while learning. In addition to this modular *manager*, multiple scheduler agents are responsible for job scheduling. They provide local rewards based on real-time feedback from job execution, which are taken into account by the *manager* to update the utility table. Schedulers use the updated utility table for resource allocation decisions ensuring a consistent view of the grid across all agents to maximize utility.

CLDS Algorithm

As previously introduced, the CLDS algorithm is composed of multiple components such as a learner agent, responsible for aggregating feedback from scheduler agents. The latter handles job allocation to resources using the utility table. They also generate rewards based on the performance of the allocated resources and send them to the learner agent. The utility table U (which is a near equivalent of the standard Q-table used in tabular Q-learning [18], except it is not a mapping of state and action but rather a map of *hot points* where utility is maximized), is a vector where each entry $U(q)$ represents the efficiency of resource q. It is updated iteratively based on the feedback from the scheduler agents. Finally, reward vectors are generated by scheduler agents for all resources based on job performance. Each element in a reward vector corresponds to the performance of a specific resource.

The author defines two steps in the CLDS algorithm that occur in discrete time steps.

Generating Local Rewards (Scheduler Agents)

In order to for the scheduler agents to perceive the performances of the resources of the environment, they must begin by generating local information from the current utility table (initialized at 0 for each resource). To do so, each scheduler agent allocates jobs to resources, and after the job execution, rewards are generated for each resource based on job performance :

1. For a completed job j_k submitted to resource r_q, the reward is :

$$R(r_q) = \frac{\text{JobSize}(j_k)}{\text{TimetoCompletion}(j_k)} \tag{3.12}$$

   where $\text{JobSize}(j_k)$ is the size of the job, $\text{TimetoCompletion}(j_k)$ is the total time taken by r_q to complete j_k. This way, it is ensured that resources completing jobs faster receive higher rewards (which is in agreement with the paper's application for job scheduling).

2. For an unfinished job j_k, a negative reward is applied (penalty) :

$$R(r_q) = -\frac{1}{\text{JobSize}(j_k)}. \tag{3.13}$$

   That way larger jobs produce small penalties, as they require more time to complete.

3. The scheduler agent sums up all the positive and negative rewards for each resource and forms the reward vector $R(q)$. Each element of the reward vector corresponds to the cumulative local reward for a specific resource.

Updating the Utility Table (Learner Agent): After generating local reward information with the scheduler agents, the learner agent collects reward vectors $R(q)$ from all scheduler agents and updates the global utility table as follows :

$$U(q) = (1 - \alpha) \times U(q) + \alpha \sum iR_i(q) \tag{3.14}$$

where:

- $U(q)$ is the utility of resource q.
- α is the learning rate, determining how quickly new rewards influence the utility.
- $R_i(q)$ is the reward for resource q provided by scheduler agent i.

This update allows the learner agent to provide a global view of resource efficiency for the scheduler agents. After an update, the utility table is shared with all the scheduler agents and they refer to it during the next step.

Job Scheduling (Scheduler Agents): Using the previously updated utility table, each scheduler agent allocates jobs in its queue to the most efficient resources :

1. For each job j_m the scheduler selects the resource r_q with the highest utility value in the utility table U:

$$r_q = \arg\max_q U(q) \tag{3.15}$$

2. The scheduler submits the job j_m to r_q and records an entry in the scheduled job list with the following details: job id j_m, resource id r_q, start time t, and completion time t'.

After being submitted the job is removed from the scheduler's queue. This scheduling phase, in contrast with the centralized learning phase, is decentralized among all agents.

As mentioned when introducing the algorithm, the authors made sure that fault was tolerated and handled during the sequential process.

Fault Handling: In case the learner agent fails, any scheduler agent can assume it's role. The learner agent has no special capabilities beyond aggregating and updating the utility table, making it replaceable by any scheduler agent. I assume, the authors implemented it quite easily by making the agents inherently all capable of doing so but activating this "learner" role when needed. The learning is ensured with the reinforcement learning framework

Proposed Details on the Reinforcement Learning Framework

The design of the RL framework for the CLDS ensures adaptability to dynamic and heterogeneous environments without requiring prior knowledge or an explicit model of the system. The authors giving no specification in their article, I assume it is defined as follows for my work:

- State Representation : the state of the system is represented by the utility table U maintained by the learner agent. This table reflects the performance of all resources based on rewards, so it allows a feedback. It is a very simple way of representing the grid.
- Action Space : the actions correspond to the job scheduling decisions made by the scheduler agents. In fact, they can choose a resource from the state of U. Specifically using 3.15.
- Reward Function : the reward is central to the learning process and ensures that resource allocation aligns with system goal. This reward is computed using either 3.12 or 3.13 if, respectively, the job was finished or not. After that the learner agent can use 3.14 to update the utility table with the collected rewards.
- Coordination and Convergence : by sharing the updated utility table with all scheduler agents, the method avoids inconsistencies and miscoordination that could arise in decentralized systems. Each scheduler agent relies on this globally updated utility table to make independent, yet synchronized, job scheduling decisions. Over time, the reinforcement learning process allows the system to converge to a sub-optimal or near-optimal scheduling policy. This means that as the utility table evolved, the scheduler agents allocate jobs in a way that balances loads, improves resource utilization, and minimizes response times, even under varying system loads and scales.

In addition to being a strong combination of centralized and decentralized paradigms, the authors specify that CLDS has a relatively low computational complexity thanks to its efficient communication and computation design. In fact, the communication cost grows linearly with the number of scheduler agents (N) and resources (Q), as each scheduler agent sends a reward vector of size Q to the learner agent at each time step. The learner agent aggregates these vectors, updates the global utility table, and broadcasts the updated table back to all scheduler agents. The total communication cost per time step is $O(N.Q)$, making it scalable for large systems. As for scheduler agents, their complexity is linked to the reward generation and resource selection based on the utility table. For J jobs and Q resources, the complexity is $O(N.J)$ for reward calculation and $O(Q)$ for resource selection. Finally, the learner

can aggregate the rewards with a complexity of $O(N.Q)$ making it very lightweight and scalable tackling one of the major issues of centralized methods. And while being initially developed for job scheduling, it is very easy to adapt it to more general load balancing problems. However, CLDS is only capable of reaching near-optimal policy in dynamic environment, but considering it's efficiency and robustness it is an interesting trade-off.

3.3.2 Multi-Agent Deep Reinforcement Learning

Starting from the foundations of centralized learning from the previous section, this section will address an article that explores the multi-agent reinforcement learning for task allocation in dynamic environments. The proposed approach, Task Allocation Process using Cooperative Deep Q-learning (TAP CQDL) [66], leverages Deep Q-Networks [20] to enable agents to learn efficient task distribution strategies while adapting to an evolving system state. To address the challenges of decentralized execution, the study integrates Cooperative Deep Reinforcement Learning (CDRL) with communication-driven learning using CommNet [67], a neural network model that acts as a simple controller for multi-agent reinforcement learning that is able to learn continuous communication between a dynamically changing set of agents.

The study builds upon fundamental concepts in reinforcement learning and Markov Decision Processes referred to in the introduction of my work 2.2. Particularly, Q-learning [18] is used as an off-policy RL method, updating Q-values for state-action pairs iteratively. However, when facing a high dimensional space, this method falls short or performance. In this article, the Deep Q-Learning (DQL) framework is used since the state space the authors will use is continuous. In this framework, the tabular aspect is dropped and the Q-value of a state-action pair is estimated using neural networks. The article also leverages extensions such as Independent Q-learning [60] allowing multiple agents to learn in parallel.

Cooperative Deep Reinforcement Learning Strategy

The strategy introduced in the paper is called Cooperative Deep-Q-Learning (CQDL), and it enables agents to coordinate their actions using communication-enhanced Q-learning. The core formulation of the setup follows standard , but the authors propose a few modifications that I will present.

Q-function Representation

Similarly to the standard definition, each agent a maintains a Q-network parametrized by θ, estimating the action-value function:

$$Q(s, a; \theta) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{3.16}$$

where s is the state, a is the action and R_t is the discounted future reward.

In the DQL setting, the bellman update is slightly different than for the Q-learning setting. In fact, it incorporates temporal difference (TD) error that represents the loss function of the network (that the optimizer will try to minimize):

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(\gamma_i^{DQN} - Q(s,a;\theta_i))^2] \tag{3.17}$$

with γ_i^{DQN} being the target value of the estimation computed with the next state-action pair. Now that each agent can individually learn to estimate the Q-values and update their parameters, they can communicate with other agents in the environment in order to create a coordination that improves with learning.

Multi-Agent Learning with Communication (CommNet)

To facilitate coordination, each agent transmits a continuous communication vector c_j, allowing message aggregation across agents:

$$c_j^{t+1} = \frac{1}{N-1}\sum_{i \neq j}^{N} h_i^t \tag{3.18}$$

where N is the total number of agents in the system, and h_i^t is the hidden state of agent i at time t. The CommNet-based Q-function is then updated taking into account both the communication vector, the hidden state, and the agents observations:

$$h_i^{t+1} = f(h_i^t, c_i^t, o_i^t) \tag{3.19}$$

with o_i^t being the agent's observation and f a neural network. I won't delve into the details of how CommNet works or the specific architecture of the strategy as it strays from the topic of this thesis and won't bring any additional value to my work.

With this communication-enhanced version of DQN, during training all agents share their experiences, optimizing the joint policy:

$$\pi(a|s,c) = arg\max_a Q(s,a,c;\theta) \tag{3.20}$$

Similarly to multi-agent reinforcement learning, the joint policy is an aggregation of all the agent's policies created during a *centralized learning* phase. During execution, each agent selects actions independently in a *decentralized* manner, relying only on its learned Q-network and local observations.

3.3.3 Task Allocation Process using CQDL

After specifying the underlying processes of their strategy, the authors define the task allocation process using the CQDL strategy. They make the assumption of full

cooperation between agents. They define the following elements when formulating the social task allocation problem :

- $A = \{a_1, ..., a_m\}$ a set of agents.
- $R = \{r_1, ..., r_k\}$ the collection of resources available to A.
- $T = \{t_1, ..., t_n\}$ the set of tasks where each task t_i is defined by the tuple $\{u(t_i), rsc(t_i), loc(t_i)\}$ with:
  - $u(t_i)$ is the utility gained if task t_i is accomplished, they assume it is identical to the reward R_t in Equation 3.16.
  - $rsc(t_i)$ is a function that specifies the amount of resources required to complete t_i.
  - $loc(t_i)$ is a function that specifies the location at which tasks arrive in the social network SN. An agent a on which the task arrives (i.e. $loc(t_i) = a$ is called the manager of the task t_i.

In this problem, each agent $a \in A$ controls a fixed amount of resources for each resource type in R, which is defined by a resource function $rsc : A \times R \longrightarrow N$. Agents are connected via a social network, which is an undirected graph SN.

More specifically, each agent is defined by a 5-tuple:

$$\{AgentId(a), Neig(a), Resource(a), State(a), Q^a(o_t^a, c_{t-1}^{a'}, h_{t-1}^a, a^a)\}$$

where $AgentId(a)$ is the identity of the agent, $Neig(a)$ is the set which indicates the neighbors of agent a, $Resource(a)$ is the resource which agent a contains, $State(a)$ demonstrates the state of agent a, and $Q^a(o_t^a, c_{t-1}^{a'}, h_{t-1}^a, a^a)$ is the Q-network of agent a (h_{t-1}^a is the individual hidden state of agent a, o_t^a is the observation, $c_{t-1}^{a'}$ the messages from other agents during the communication, and a^a the action of agent a. Each of these at time t).

Finally the authors define a task allocation, which is mapping $\phi : T \times A \times R \longrightarrow N$ (i.e. a mapping of a set of tasks assigned to agents with resources). In addition to finding a mapping, where no tasks are left undone and the resource constraints are respected, the agents maximize the reward in the environment to achieve an overall goal.

Algorithm Detail

The article describes the execution algorithm as a decentralized approach where agents communicate and allocate tasks efficiently. The authors define 3 type of agents for their MAS:

- Manager : the agent which requests help for its task when it is not able to complete it himself.
- Participant : the agent which accepts and performs the announced task.
- Mediator : the agent that receives another agent's commitments for assistance to find Participants.

At all time, and agent is either **Busy** (when the agent is a Manager or a Participant), **Committed** (when an agent is a Mediator), **Idle** (when an agent is not assigned or committed to any task). Logically, only idle agents can be assigned to a new task as a Manager or a Participant, or committed as a mediator. I detailed the general idea of the process in the algorithm presented in appendix 6.

The function `ResAnnounceMess` is essentially a call to all neighbors asking for participation to complete the task the manager agent has just received. Upon receiving the result of function `ProposeMess`, the manager agent utilizes CommNet to process the state-view s of all neighboring agents, encoding and iterating over hidden states h and communication vectors c to obtain h_k defined at 3.19. It then samples actions a for its neighbors based on the learned Q-values $Q(a, s)$ computed with 3.16. If the manager finds multiple suitable agents based on resource proposals, it applies a roulette selection to pick an agent based on its utility (reward), choosing the one with the highest utility and marking this selected agent as busy. If multiple agents share the same utility, is selects the one with the shortest execution time (information relayed by the agent, the authors do not specify when or how it is initialized). However, if only one agent is available, the manager assigns the task in a greedy way not considering utility. Finally, the function `RefuseMess` is simply a message stating that a neighboring agent isn't idle.

Performance Evaluation

The experiments in this paper evaluate the effectiveness of the process introduced in the previous section, by comparing it against Greedy Distributed Allocation Protocol (GDAP) [68] and a previous task allocation algorithm developed by the authors (TAP) [66] that does not use cooperative deep q-learning. The evaluation focuses on utility ratio and execution time in different network settings to assess the efficiency, scalability, and communication overhead of the proposed approach.

The authors specify the GDAP is a distributed task allocation where manager agents try to find neighboring contractors to complete their tasks. Then, contractors bid (similarly to 3.2.2) on tasks based on available resources, and the manager selects the best offer. However, if no contractor is found, the task is removed from the system (leading to task failures). The authors have repeated the experiment described in the article introducing GDAP for proper performance evaluation. In this experiment there are two settings:

Table 3.1: Experimental Settings for Small-World and Scale-Free Networks

Net. Type	**Agents**	**Tasks**	**Res. Types**	**Avg. Res/Task**	**Avg. Neighbors**
Small-World	40	20	5	30	Variable
Scale-Free	100–2000	5:3 ratio	20	100	Fixed (10)

In both of these settings, the algorithms have been evaluated according to two criteria:

- Utility Ratio: percentage of tasks successfully allocated. The higher the percentage, the better the performance.
- Execution Time: the performing time of each algorithm in each network under different situations. The lower the time, the better the performance.

The first experiment evaluates how the number of neighbors influences task allocation performance. The proposed method TAP CQDL achieves the highest utility ratio, reaching 1.2, significantly outperforming TAP at 0.6 and GDAP at 0.3. GDAP struggles in scale-free networks due to its inability to reallocate tasks, whereas small-world networks provide better results by facilitating more direct connections among agents.

For all methods, the utility ratio improves as the number of neighbors increases, demonstrating the effect of connectivity on task allocation efficiency. However, execution time remains relatively stable ($\pm$0.5ms), indicating that while more neighbors enhance success rates, they do not significantly impact speed. GDAP is the fastest due to its reliance on direct neighbors, whereas TAP CQDL incurs higher computational costs due to deep Q-learning and communication-based decision-making.

Table 3.2: Comparison of Utility Ratio and Execution Time in Setting 1

Algorithm	**Utility Ratio (Best Case)**	**Execution Time (Worst Case, ms)**
TAP CDQL	1.2	7
TAP	0.6	5
GDAP	0.3	2

The second experiment examines the scalability of the approaches in networks ranging from 100 to 2000 agents, simulating real-world applications such as large-scale distributed systems or fleets of autonomous robots. TAP CQDL and TAP maintain a high task success rate, though performance gradually declines as the network size increases. TAP CQDL shows a decrease from 0.8 to 0.7, while TAP drops from 0.6 to 0.5. In contrast, GDAP's performance deteriorates significantly going from 0.2 to 0.1. I assume it is due to its reliance on direct neighbors, making it ineffective in large-scale networks.

As expected, small-world networks perform better than scale-free networks, as they offer higher connectivity per agent, facilitating task completion. In terms of execution time, GDAP remains the fastest (2ms), while TAP CQDL and TAP displays higher time complexity due to reinforcement learning techniques.

Table 3.3: Comparison of Utility Ratio and Execution Time in Setting 2

Algorithm	**Utility Ratio (100 to 2000 Agents)**	**Execution Time (ms)**
TAP CDQL	$0.8 \rightarrow 0.7$	7
TAP	$0.6 \rightarrow 0.5$	5
GDAP	$0.2 \rightarrow 0.1$	2

Overall, the results highlight the advantages of TAP CQDL in terms of efficiency and adaptability, particularly in larger networks. However, this comes at the cost of higher execution time, which remains a key factor to optimize in future work.

In conclusion, approach proposed by the authors achieves the highest task allocation efficiency, but at the cost of longer execution time. The baseline GDAP is the fastest but fails to allocate many tasks, especially in large networks as it considers only direct neighbors. We also see that the new approach outperforms the former one proposed by the authors showing the benefits of deep reinforcement learning and communication-aware allocation. Finally, the impact of network topology is significant: small-world networks are more favorable for task allocation than scale-free networks. The authors showed that introducing deep Q-learning improves the system performance by means of past task allocation performance at the cost of a slightly more time consuming execution.

3.3.4 Conclusion on CTDE

In this section I have presented methods that illustrate that the CTDE paradigm is a powerful framework for task allocation and load balancing in Multi-Agent Systems, combining global coordination during training with autonomous execution for scalability in dynamic environments. The first approach, Centralized Learning Distributed Scheduling, presented at 3.3.1 employs centralized reinforcement learning with a modular auctioneer role, ensuring robustness while maintaining good time complexity, though it only achieves near-optimal policies in dynamic settings. On another hand, the approach presented in Section 3.3.2 TAP CQDL enhances adaptability by integrating DQN and communication-driven learning, enabling agents to cooperatively allocate resources. It highlighted the benefit of communication in contrast to the conclusions that were drawn in decentralized methods in Section 3.2.2 at the cost of employing more advanced technologies making the execution time slightly higher. Then again, what trade-off will the user take given the application

of the model ? All of these approaches demonstrate the Centralized Training, Decentralized Execution paradigm's effectiveness in balancing optimization, scalability, and autonomy in MAS.

Chapter 4

Discussion

In this chapter we will orient our discussion on highlighting the key similarities and differences between the presented methods and how a few elements of each could constitute a good solution for load balancing and task assignment in multi-agent systems. Each of these parts are either purely critical on the work I have analyzed or supported by the authors of the articles. I will subsequently compare and criticize each approach under a different angle, starting with algorithm complexity and implementation complexity. I will then discuss the limitations of each methods and how certain aspects of other methods can be a solution.

4.1 About Complexities

Task allocation methods in multi-agent systems exhibit a wide spectrum of complexity. On the simpler end, centralized approaches like Fisher market-based and auction-based methods offer conceptually straightforward solutions by leveraging global optimization to achieve envy-free and Pareto-optimal allocations. Algorithms such as FMC_TA [10] presented in Section 3.1.1 or CBBA [29] presented in Section 3.2.2, have explicitly defined exhaustive rule-sets that need to be met otherwise the system will not function properly. The action policies in themselves depend on predefined actions with temporal and spatial constraints that need to be respected. Alternatively the SCA algorithm in Section 3.1.2 [11] does not proceed to checking if constraints are respected, it is less demanding on that aspect. This method employs a heuristic method to explore the space of possible actions. The hill-climbing process

helps find the best possible allocation. However, SCA does not take into account temporal or spatial constraints. Without the predefined rule-sets it proposes a simpler implementation than the centralized FMC_TA algorithm or the decentralized CBBA algorithm, and the stochastic nature makes it more robust to unusual events in the environment. The question is, which is the most suited for which application ? Is it preferable to have an algorithm that implements constraints at the cost of specifying each of them ? Or is it better to have a simpler system that assumes no constraints be met, straying from real life scenarios, but has a strong adaptability to uncertain environment ? Either way, these methods do not use the "intelligence" other methods use. Each agent does not keep track of its history. In a centralized setup each agent has global knowledge of the environment and they do not communicate with each other. In addition to this, centralized methods often suffer from scalability issues and the risk of a single point of failure. Although, one could speculate that that the modular *manager* presented for the CLDS Algorithm [64] in Section 3.3.1 could be implemented on a centralized system, where agents can assume a role when a problem occurs.

In contrast, decentralized methods propose a more scalable, robust and dynamic approach to load balancing or task allocation in MAS, at the cost of a much more complex implementation. Either by the fact that the operations occur simultaneously. Or by the underlying concepts such as game theory-based methods presented in Section 3.2.1, for example, that model inter-agent interactions as strategic games, which allow for nuanced decision-making but increase computational complexity due to the need to resolve Nash equilibria. Or the distributed optimization-based approaches like CBAA-CBBA (Section 3.2.2) whose determinism and dependence on strictly defined environments and exhaustive rule-set is similar to the centralized methods, the key difference is that agents share information that improves their decision and it proposes single and multi-task assignment with a guarantee of at least 50% of the optimal solution.

From the underlying processes perspective, maybe the most complex approaches and the ones that propose a real "intelligence" and potentially emerging behaviors from agent interactions, are the approaches relying on a reinforcement learning framework. In the decentralized methods, although not specified by the authors or explicitly stated in Section 3.2.2, the *multi-agent multi-resource stochastic system* uses the reinforcement learning framework. The RL framework is illustrated by the use of dynamic parameters that are updated over time. The exploration / exploitation factor as well as the learning rate, or the discount for rewards are all parameters in the RL framework. The paper presented by Schaerf et al. [36] also illustrates the complexity of RL with the conclusions they draw. In fact, using the most efficient resource over the other ones is said to be a poor policy by the authors. Which makes sense, when fully exploiting a resource, other resources are underused

or neglected, and the system finds itself in a local minima. The authors also draw the conclusion that adaptive selection rules are the best, validating my assumption that approaches where system parameters and agent information are updated over time and improved with previous iterations are better than the static ones. The method that, in my opinion, brings a very interesting contribution is the one presented at Section 3.3.1, where a central *manager* can be replaced during training by any other scheduler agent assuming the new role. This mechanism could be implemented on various other systems.

Pushing complexity even further may come with a computing cost, but with the improvement of GPU accelerated computing or faster CPUs, methods like Multi-Agent Deep Reinforcement Learning presented at Section 3.3.2 propose a communication system augmented with a neural architecture: CommNet. While leveraging complex concepts, this approach has shown that a previous statement stated by Schaerf et al. [36] saying that communication among agents might not be fully beneficial was, in fact, not right. This complex system architecture is able to aggregate the information of surroundings neighbors with an awareness brought by the back-propagation mechanism to understand the agents condition in the current state. In the experiments, the authors show that the novel, more complex method has a slightly higher time of execution that can be neglected at the scale at which it was tested for real-time decision making compared to its faster, but less developed competitors.

We can summarize these reflections saying that this contrast in complexity highlights a key trade-off: simpler algorithms benefit from speed and reduced overhead, whereas complex, learning-based approaches, despite their higher computational cost, can better tackle the challenges of task allocation and load balancing in unpredictable, real-world settings.

4.2 About Limitations

As stated in Section 2.5, the approaches presented in this work are able to solve or handle few challenges among so many other. In the design of MAS, selecting the appropriate methods involves considering and weighting trade-offs between optimality, scalability, adaptability, and communication overhead. The approaches we have presented so far carries inherently limitations that not only restrict their individual performance but also affect how these strategies might be combined or extended in more complex environments.

Centralized approaches provide a global view and control over the environments which can be used to compute envy-free and Pareto optimal task assignments (e.g., via the FMC_TA or SCA Algorithms). However these methods struggle with scalability because a single controller must process information from every agent, which

potentially hold information that also need to be aggregated. This creates computational bottlenecks as the amount of information carried and the number of agents grows. In addition to this, centralized approaches suffer from the single-point-of-failure, making the system vulnerable to a lot of external factors hardware-wise or software-wise. The solution to scalability would be to utilize the local aspect of decentralized approaches. In fact each agent would only need to aggregate a few information from its neighbors with its own (e.g., the CBBA algorithm allows the agent to iteratively understand the global view of the system by sharing a winning bid list). Whereas the single-point-of-failure could be solved with the CLDS approach, where the general *manager* can be replaced at any time by any other agent if it fails. Such improvement would open up a lot of application to centralized approaches, but then, would they be called centralized if any agent can assume the role of global *manager*, and would their decisions be impacted by their former experience as a scheduler or the information they gathered at a given point in the execution ? This role changing system can also be a solution for communication overhead limitations. If we assume that regularly in time the central *manager* is changed with an agent that has a lot of neighbors, the information of all agents wouldn't have to be relayed to the central *manager* since the agent assuming the role has already aggregated the information of it's neighbors, thus gaining time and computational power ? This would be a modular approach to centralized methods worth exploring.

On the other hand, decentralized methods enhance the system robustness by distributing decision-making among agents, eliminating strictly the single-point-of-failure issue. However, the lack of central coordination means that agent base themselves solely on local observations. Given how the system is defined, this limited view can lead to local minima and suboptimal global outcomes. And this is specifically the case when agents pursue individual objectives, what is the global benefit when the only benefit they know is their own ? The other issue about decentralized approaches is that each agent must evolve with a non stationary environment. That is, an environment that is modified due to the actions of the other agents. A dynamic environment makes it challenging to achieve stable convergence during learning and can result in unpredictable behavior. To solve this issue, communication among the agents is used to approximate the environment via an aggregation of the local observations (e.g., CBBA, MAMRSS), or heuristics are used to explore the action space stochastically and improve over time (e.g., DSA). In order to improve decentralized methods, research should be made on how agents can cooperate with each other and which piece of information they should be sharing with one another. One can imagine a system where agents are of different nature with a global role at the beginning and according to what local observation the agent perceives, it can assume a specific role (explorer, exploiter, messenger etc.) ? An interesting piece of work was provided by Suarez et al. on agent interaction and cooperation/competition in

MAS [69].

Finally the approaches that make use of the CTDE paradigm who leverage the advantages of both centralized and decentralized methods face what is called a training-execution mismatch. That is, the mismatch between global information used during training to learn a coordination among agents, and the information that are used in a local way during the execution. This can particularly happen when the training environment does not capture the complexity and dynamics of the operational setting. They remain, however, the most used and developed techniques as it tackles multiple limitations at once. The cost at which they come is on two sides; the first being the computational cost mentioned in the previous section, the second is the fact that they rely on techniques employing deep learning (e.g., CQDL) and that for any application using deep learning, tuning and overfitting is a major issue. This can have an important effect on the decisions the agent make on unforeseen or rapidly changing real-world scenarios. For CQDL we could imagine an improvement of attention-based graph networks to estimate the environment and surroundings of each of our networks and also estimate the implicit interactions an agent can have with another by interacting with a resource or a task its neighbor has seen before.

Overall, our critical analysis displays that no single approach is universally superior. And while the ones using CTDE are the most promising, each method embodies trade-offs that must be balanced according to the specific requirements of the application domain. We have summarized our understandings in a comparative table in Appendix 7.

Chapter 5

Conclusion

The aim of this work was to present various approaches that solve task allocation and load balancing in multi-agent systems. First we proposed a review of the underlying theory behind the approaches we presented, citing the foundational work of distributed artificial intelligence and how software agents are defined. Then we detailed what a multi-agent system is and specified the Markov decision process framework and its variations to reinforcement learning for the single agent framework, as well as multi-agent reinforcement learning for multiple agents After that we proposed a formal definition of the two problems our work is about : task allocation and load balancing in MAS. We then presented the major challenges the community was facing when trying to solve the problems of load balancing and task allocation by illustrating existing methods and their impact. Finally we briefly detailed how these algorithms were evaluated in a multi-agent setup.

In chapter 3, we have chosen to present the existing methods under 3 panels. The centralized methods, the decentralized methods, and the centralized-training decentralized-execution paradigm. Throughout the analysis of these methods we have highlighted their limitations, advantages, and hinted how they could be complementary with one another. The aim was to make each approach as clear as possible for the reader to understand the underlying processes behind game-theoretic based approaches or distributed optimization approach for example. Throughout our work, the approaches have gained in complexity, ranging from a simple, deterministic Fisher Market-based approach to a multi-agent deep reinforcement learning with communication capabilities approach.

Finally, in chapter 4, we have put all these methods in parallel and have discussed their complexity and limitations by comparing them, being critical on the way they were built, and proposing questions for further research. We have drawn the conclusion that each of the presented methods contribute to advancing the field of load balancing and task allocation in MAS by tackling one or multiple challenged wether it is being scalable, optimal, failure-proof, or communication enhanced. The research in multi-agent systems is very active and there is still much to be done to improve performance and find methods that are applicable to real-life scenarios.

Chapter 6

Appendix A: CLDS Algorithm

Algorithm 5 Task Allocation using Cooperative Deep Q-Learning (TAP CDQL)

Require: Task t_{Mn}, Manager Agent AM_n, Maximum retries $N_{\max}$
if AM_n cannot complete t_{Mn} **then**
Broadcast **ResAnnounceMess** $\langle AM_n.ID, t_{Mn}.ID, t_{Mn}.required\_res \rangle$ to neighbors
Collect responses from neighboring agents
for each Agent A_j in responses **do**
if $A_j.state = Idle$ **then**
Send **ProposeMess** $\langle A_j.ID, A_j.res, A_j.exec\_time, A_j.util, Q(A_j) \rangle$ to AM_n
else
Send **RefuseMess** $\langle A_j.ID \rangle$ to AM_n
end if
end for
Initialize $selected\_agent \leftarrow$ None, $best\_util \leftarrow -\infty$
for each response in responses **do**
if response.utility $>$ $best\_utility$ **then**
$selected\_agent \leftarrow response.agent$
$best\_util \leftarrow response.util$
else if response.util = best_util$\wedge$response.exec_time<selected_agent.exec_time **then**
$selected\_agent \leftarrow response.agent$
end if
end for
if $selected\_agent \neq$ None **then**
Send **Contract** $\langle AM_n.ID, selected\_agent.ID, t_{Mn}.ID, selected\_agent.res \rangle$
Set $selected\_agent.state \leftarrow Busy$
else
$AM_d \leftarrow$ select_mediator$(AM_n.neighbors)$
if $AM_d \neq$ None **then**
Send **Commitment** $\langle AM_n.ID, AM_d.ID, t_{Mn}.ID, t_{Mn}.remaining\_res \rangle$
Set $AM_d.state \leftarrow Committed$
Call TAP_CDQL(t_{Mn}, AM_d) ▷ Recursive call for reallocation
else
Revert all allocations
return "Task Allocation Failed"
end if
end if
end if
return "Task Successfully Allocated"

Chapter 7

Appendix B: Comparative Table

Method	Nature	Underlying Approach	Tackled Challenges	Key Strengths	Key Limitations
FMC_TA	Centralized	Market-based Optimization	Fairness Task Scheduling	Envy-free Pareto-optimal Multiple constraints	Scalability issues Single point of failure Lacks robustness
SCA	Centralized	Markov Chain Monte Carlo	Task Allocation Adaptability	Robust to uncertainty Heuristic-based	Ignores constraints Computational bottlenecks Single point of failure
CBBA	Decentralized	Distributed Auction	Scalability Partial Observability	Near-optimal ($\geq$50%) Local information sharing	Rule-based Communication overhead
DSA	Decentralized	Game-Theoretic Optimization	Dynamic Task Assignment	Handles real-time changes Scalable	Local minima risk No central coordination
MAMRSS	Decentralized	Reinforcement Learning	Dynamic Adaptation	Learns from experience Adaptive policies	Extensive training required Suboptimal convergence risk
CLDS	Hybrid	CTDE + Role Adaptation	Fault Tolerance Scalability	Role switching mechanism Increased robustness	Training-execution mismatch
CQDL	Hybrid	Multi-Agent Deep RL	Task Allocation Communication Efficiency	Uses CommNet Aggregates neighbor info	High computational cost Risk of overfitting

Table 7.1: Comparison of Centralized, Decentralized, and Hybrid Load Balancing Methods

References

[1] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," 2020. [Cited on pages vii, 3, and 4]

[2] G. Weiss, ed., *Multiagent Systems. 2nd edition.* MIT Press, 2013. [Cited on page 3]

[3] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 1–1, 01 2015. [Cited on page 4]

[4] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-Agent Systems: A Survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018. [Cited on pages 4 and 9]

[5] A. Bond and L. Gasser, *Readings in Distributed Artificial Intelligence.* Erscheinungsort nicht ermittelbar: Morgan Kaufmann, 1st edition ed., 2014. [Cited on page 7]

[6] G. M. P. O'Hare and N. R. Jennings, eds., *Foundations of distributed artificial intelligence.* Sixth-generation computer technology series, New York: Wiley, 1996. [Cited on page 8]

[7] S. Franklin and A. Graesser, "Is It an agent, or just a program?: A taxonomy for autonomous agents," in *Intelligent Agents III Agent Theories, Architectures, and Languages* (J. G. Carbonell, J. Siekmann, G. Goos, J. Hartmanis, J. Van Leeuwen, J. P. Müller, M. J. Wooldridge, and N. R. Jennings, eds.), vol. 1193, pp. 21–35, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. Series Title: Lecture Notes in Computer Science. [Cited on pages 8 and 9]

[8] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, vol. 10, pp. 115–152, June 1995. [Cited on pages 8 and 9]

[9] H. S. Nwana, "Software agents: an overview," *The Knowledge Engineering Review*, vol. 11, pp. 205–244, Sept. 1996. [Cited on page 9]

[10] S. Amador, S. Okamoto, and R. Zivan, "Dynamic Multi-Agent Task Allocation with Spatial and Temporal Constraints," *AAAI*, vol. 28, June 2014. [Cited on pages 9, 15, 22, and 59]

[11] K. Zhang, E. G. Collins, and D. Shi, "Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, July 2012. [Cited on pages 9, 18, 25, and 59]

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016. [Cited on pages 10 and 11]

[13] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, Nov. 2019. [Cited on page 10]

[14] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990. [Cited on page 10]

[15] R. A. Howard, "Dynamic programming and markov processes," *MIT Press google schola*, vol. 2, pp. 39–47, 1960. [Cited on page 10]

[16] L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," *IEEE Trans. Syst., Man, Cybern. C*, vol. 38, pp. 156–172, Mar. 2008. [Cited on pages 10 and 12]

[17] K. Zhang, Z. Yang, and T. Başar, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," in *Handbook of Reinforcement Learning and Control* (K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, eds.), vol. 325, pp. 321–384, Cham: Springer International Publishing, 2021. Series Title: Studies in Systems, Decision and Control. [Cited on pages 10 and 12]

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach Learn*, vol. 8, pp. 279–292, May 1992. [Cited on pages 11, 49, and 52]

[19] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE control systems magazine*, vol. 12, no. 2, pp. 19–22, 1992. [Cited on page 11]

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015. [Cited on pages 11 and 52]

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach Learn*, vol. 8, pp. 229–256, May 1992. [Cited on page 11]

[22] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014. [Cited on page 11]

[23] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, pp. 1291–1307, Nov. 2012. [Cited on page 11]

[24] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020. [Cited on page 12]

[25] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artificial Intelligence*, vol. 101, pp. 165–200, May 1998. [Cited on page 13]

[26] M. U. Arif and S. Haider, "A Flexible Framework for Diverse Multi-Robot Task Allocation Scenarios Including Multi-Tasking," *ACM Trans. Auton. Adapt. Syst.*, vol. 16, pp. 1–23, Mar. 2021. [Cited on page 13]

[27] K. Macarthur, R. Stranders, S. Ramchurn, and N. Jennings, "A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems," *AAAI*, vol. 25, pp. 701–706, Aug. 2011. [Cited on page 13]

[28] S. S. Fatima and M. Wooldridge, "Adaptive task resources allocation in multi-agent systems," in *Proceedings of the fifth international conference on Autonomous agents*, (Montreal Quebec Canada), pp. 537–544, ACM, May 2001. [Cited on page 13]

[29] Han-Lim Choi, L. Brunet, and J. How, "Consensus-Based Decentralized Auctions for Robust Task Allocation," *IEEE Trans. Robot.*, vol. 25, pp. 912–926, Aug. 2009. [Cited on pages 14, 29, 33, 34, 39, and 59]

[30] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, "New challenges in dynamic load balancing," *Applied Numerical Mathematics*, vol. 52, pp. 133–152, Feb. 2005. [Cited on page 14]

[31] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load Balancing for Ultradense Networks: A Deep Reinforcement Learning-Based Approach," *IEEE Internet Things J.*, vol. 6, pp. 9399–9412, Dec. 2019. [Cited on pages 14 and 15]

[32] E. Jafarnejad Ghomi, A. Masoud Rahmani, and N. Nasih Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, June 2017. [Cited on page 14]

[33] J. Watts and S. Taylor, "A practical approach to dynamic load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, pp. 235–248, Mar. 1998. [Cited on page 14]

[34] S. Choudhury, J. K. Gupta, M. J. Kochenderfer, D. Sadigh, and J. Bohg, "Dynamic Multi-Robot Task Allocation under Uncertainty and Temporal Constraints," 2020. Version Number: 3. [Cited on page 15]

[35] Y. Shoham and M. Tennenholtz, "On the synthesis of useful social laws for artificial agent societies," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, p. 276–281, AAAI Press, 1992. [Cited on page 16]

[36] A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive Load Balancing: A Study in Multi-Agent Learning," *jair*, vol. 2, pp. 475–500, May 1995. [Cited on pages 16, 17, 18, 29, 33, 40, 42, 60, and 61]

[37] J. Lifflander, P. P. Pebay, N. L. Slattengren, P. L. Pebay, R. A. Pfeiffer, J. D. Kotulski, and S. T. McGovern, "A Communication- and Memory-Aware Model for Load Balancing Tasks," 2024. Version Number: 1. [Cited on page 16]

[38] P. Skowron and K. Rzadca, "We Are Impatient: Algorithms for Geographically Distributed Load Balancing with (Almost) Arbitrary Load Functions," 2014. Version Number: 1. [Cited on page 16]

[39] D. Rutten and D. Mukherjee, "Mean-field Analysis for Load Balancing on Spatial Graphs," 2023. Version Number: 1. [Cited on page 17]

[40] K. Zhang, T. SUN, Y. Tao, S. Genc, S. Mallya, and T. Basar, "Robust multi-agent reinforcement learning with model uncertainty," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 10571–10583, Curran Associates, Inc., 2020. [Cited on page 17]

[41] S. He, S. Han, S. Su, S. Han, S. Zou, and F. Miao, "Robust multi-agent reinforcement learning with state uncertainty," 2023. [Cited on page 17]

[42] L. Shi, E. Mazumdar, Y. Chi, and A. Wierman, "Sample-efficient robust multi-agent reinforcement learning in the face of environmental uncertainty," 2024. [Cited on page 17]

[43] J. B. Fernandes, Ítalo A. S. de Assis, I. M. S. Martins, T. Barros, and S. X. de Souza, "Adaptive asynchronous work-stealing for distributed load-balancing in heterogeneous systems," 2024. [Cited on page 18]

[44] E. Beauprez, *Système multi-agents adaptatif pour l'équilibrage de charge centré utilisateur*. PhD thesis, 2024. Thèse de doctorat dirigée par Morge, Maxime Informatique et applications Université de Lille (2022-....) 2024. [Cited on page 18]

[45] G. M. Skaltsis, H.-S. Shin, and A. Tsourdos, "A survey of task allocation techniques in mas," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 488–497, IEEE, 2021. [Cited on page 19]

[46] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani, "Market equilibrium via a primal–dual algorithm for a convex program," *J. ACM*, vol. 55, pp. 1–18, Oct. 2008. [Cited on page 23]

[47] K. Binmore, A. Rubinstein, and A. Wolinsky, "The nash bargaining solution in economic modelling," *The RAND Journal of Economics*, pp. 176–188, 1986. [Cited on page 29]

[48] D. Grosu, A. T. Chronopoulos, and M.-Y. Leung, "Load balancing in distributed systems: An approach using cooperative games," in *Proceedings 16th International Parallel and Distributed Processing Symposium*, pp. 10–pp, IEEE, 2002. [Cited on page 29]

[49] A. Chapman, R. A. Micillo, R. Kota, and N. Jennings, "Decentralised dynamic task allocation: A practical game–theoretic approach," 2009. [Cited on pages 29, 30, and 32]

[50] D. Grosu and A. T. Chronopoulos, "Noncooperative load balancing in distributed systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1022–1034, 2005. [Cited on page 29]

[51] W. Zhang and Z. Xing, "Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling," in *AAMAS-02 Third International Workshop on Distributed Constraint Reasoning*, pp. 192–201, Citeseer, 2002. [Cited on page 32]

[52] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, "Multi-task allocation and path planning for cooperating uavs," *Cooperative control: models, applications and algorithms*, pp. 23–41, 2003. [Cited on page 39]

[53] C. Schumacher, P. R. Chandler, and S. R. Rasmussen, "Task allocation for wide area search munitions," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 3, pp. 1917–1922, IEEE, 2002. [Cited on page 39]

[54] D. Bernhardt and D. Scoones, "A note on sequential auctions," *The American economic review*, vol. 84, no. 3, pp. 653–657, 1994. [Cited on page 39]

[55] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt, "Simple auctions with performance guarantees for multi-robot task allocation," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 1, pp. 698–705, IEEE, 2004. [Cited on page 39]

[56] R. Rico, M. Sánchez-Manzanares, F. Gil, and C. Gibson, "Team implicit coordination processes: A team knowledge–based approach," *Academy of management review*, vol. 33, no. 1, pp. 163–184, 2008. [Cited on page 39]

[57] S. L. Smith and F. Bullo, "Monotonic target assignment for robotic networks," *IEEE Transactions on Automatic Control*, vol. 54, no. 9, pp. 2042–2057, 2009. [Cited on page 39]

[58] V. Jain and I. E. Grossmann, "Algorithms for hybrid milp/cp models for a class of optimization problems," *INFORMS Journal on computing*, vol. 13, no. 4, pp. 258–276, 2001. [Cited on page 40]

[59] W. B. Arthur, "Inductive reasoning and bounded rationality," *The American economic review*, vol. 84, no. 2, pp. 406–411, 1994. [Cited on page 41]

[60] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017. [Cited on pages 47 and 52]

[61] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, 2016. [Cited on page 47]

[62] A. Galindo-Serrano and L. Giupponi, "Distributed q-learning for aggregated interference control in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010. [Cited on page 47]

[63] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 4213–4220, 2019. [Cited on page 47]

[64] M. Moradi, "A centralized reinforcement learning method for multi-agent job scheduling in grid," in *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 171–176, 2016. [Cited on pages 48 and 60]

[65] G. Chen, "A new framework for multi-agent reinforcement learning – centralized training and exploration with decentralized execution via policy distillation," 2019. [Cited on page 48]

[66] D. B. Noureddine, A. Gharbi, and S. B. Ahmed, "Multi-agent deep reinforcement learning for task allocation in dynamic environment.," in *ICSOFT*, pp. 17–26, 2017. [Cited on pages 52 and 55]

[67] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, 2016. [Cited on page 52]

[68] M. De Weerdt, Y. Zhang, and T. Klos, "Distributed task allocation in social networks," in *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*, pp. 1–8, 2007. [Cited on page 55]

[69] J. Suarez, Y. Du, P. Isola, and I. Mordatch, "Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents," *arXiv preprint arXiv:1903.00784*, 2019. [Cited on page 63]