Adaptive Resource Allocation in Multi-Agent Social Networks

by

Mojtaba Malek Akhlagh
B.Sc., Physics, University of Guilan, Iran, 2007
Master of Information Technology, Multimedia University, Malaysia, 2010
M.Sc., Computer Science, University of Northern British Columbia, Canada, 2015

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Adaptive Resource Allocation in Multi-Agent Social Networks

by

Mojtaba Malek Akhlagh
B.Sc., Physics, University of Guilan, Iran, 2007
Master of Information Technology, Multimedia University, Malaysia, 2010
M.Sc., Computer Science, University of Northern British Columbia, Canada, 2015

Supervisory Committee

Dr. Jens Weber, Supervisor
(Department of Computer Sience)

Dr. Kui Wu, Departmental Member
(Department of Computer Sience)

Dr. Kin Fun Li, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Distributing resources among agents in social networks is an important and challenging problem. It involves deciding on assignment of a subset of resources to each agent based on the system objectives. Various instances of this problem can be observed in healthcare resource distribution, disaster management, cloud resource optimization, etc. In collaborative systems, different coordination techniques have been introduced in order to maximize overall social welfare. However, this problem becomes more complex in large-scale networks with limited connectivity among the agents. Moreover, in dynamic environments, where the set of tasks or resources change over time, an effective system needs to adapt to changes in the environment. Existing mechanisms fall short in addressing the social network constraints, and do not present efficient solutions when dealing with dynamic changes of supply and demand quantities. In this thesis, we view this social resource allocation problem (SRAP) as a multi-agent coordination problem. In a centralized approach, we consider a master agent with global knowledge, which makes decisions for all the agents. We present a greedy mechanism using an efficiency heuristic, and a learning-based mechanism by formulating the SRAP as a Markov Decision Process (MDP), and incorporating deep Q-learning. On the other hand, in a decentralized approach, we present a multi-agent protocol, which relies on local interactions among agents and their local knowledge only. The protocol enables the agents to negotiate with each other on allocation of resources to their tasks. It allows an agent in need of resources to concurrently negotiate with multiple providers and combine their resource contributions. We present greedy and learning-based mechanisms by integrating deep Q-learning into the negotiation process. In addition, the agents are able to cascade their corresponding information along the network, and apply timeouts in their messages. Hence, the decentralized protocol enables the multi-agent system to be self-organized, without relying on any central entity. We evaluate our approaches by developing simulation models of agents, tasks, and resources. We perform experiments on three main types of social networks, namely small-world, scale-free, and random networks. We conduct an empirical study of the performance of these approaches under varying conditions, such as resource availability, resource types, task requirements, etc. Our simulation results present a comprehensive analysis of various approaches across different types of social networks, by highlighting the strengths and limitations of centralized versus decentralized, as well as greedy versus learning-based approaches.

Contents

List of Tables

List of Figures

List of Algorithms

ACKNOWLEDGEMENTS

I would like to sincerely express my gratitude to my supervisor, Dr. Jens Weber, for his guidance and support throughout all stages of my PhD studies. This dissertation would not have been completed without his advice, encouragement, and patience.

I am grateful to Dr. Kui Wu for serving as a member of my supervisory committee and for his insightful advice and encouragement, which helped me complete this study.

I would like to thank Dr. Kin Fun Li for his involvement as a supervisory committee member and for his comments and encouragement, which have improved the quality of my research.

I am thankful to Dr. Alex Thomo and Dr. Venkatesh Srinivasan for the courses they taught during my candidacy. Their lectures and advice were invaluable to my research.

I am also thankful to Dr. Glen Berseth for agreeing to serve as the external examiner for my defence.

In the end, I want to dedicate this thesis to my family: my wife, Mahboubeh, for her support and all the beautiful moments in Victoria; my parents, Nahid and Esmaeil, for their endless love, support, and encouragement; and Moein for being an amazing brother in all stages of my life.

Chapter 1

Introduction

Resource allocation problems can be observed in various real-world applications, from healthcare and disaster management to manufacturing and cloud computing. It involves distributing a set of resources among a set of agents in order to maximize a certain objective, such as social welfare, or fairness. An optimal solution to a resource allocation problem is an allocation assigning a subset of resources to each agent such that an objective function is optimized. This problem becomes more challenging when the agents act in a dynamic environment, where the set of tasks or resources may change over time. As the environment evolves, a mechanism which finds the best possible allocation needs to adapt to the changes in the environment by adjusting the solution, i.e. reallocating the resources.

1.1 Social Resource Allocation Problem

In this thesis, we investigate a variant of the resource allocation problems, in which the agents require resources in order to perform their delegated tasks which arrive over time. In particular, we consider dynamic environments in which supply and demand of resources may change over time. In our model, the individual agents have a domain specific motivation to collaborate, and share a common goal to maximize the total utilities gained by all agents over time. In addition, we consider the agents to be connected in a social network, which constrains their interactive behavior, inspired by business relations among organizations in real-world scenarios. In the context of social networks, a resource allocation problem becomes even more complex due to the connectivity and proximity constraints determined by the network structure. An

instance of such problems can be seen in allocating resources to hospitals distributed in a region, in particular, in a pandemic scenario, where the dynamism comes from sudden changes in demand e.g. an unforeseen number of patients in emergency department, and hence a shortage of critical resources e.g. ICU beds. This is more challenging as the hospitals in different regions may not be directly collaborating, but connected through a higher-level organization; i.e. there exists a hierarchy in the network. Hence, an effective mechanism needs to consider the social network constraints, and recognize the most in-demand hospitals. Another instance can be found in the allocation of required resources during natural disasters like earthquakes or hurricanes. In such scenarios, the allocation of food, water, and medical supplies to affected regions is challenging since the resources may need to be transported from unaffected regions. Optimizing the distribution of resources to reach the most impacted regions is crucial, and needs to consider changes in supply and demand quantities over time.

1.2 Motivation

Dynamic allocation of resources in a social network can be observed in many real-world scenarios. For instance, in the healthcare domain, challenges such as distribution of vaccines or blood products require effective resource allocation strategies to ensure timely and equitable access [7, 43]. Similarly, in cloud computing, the effective allocation of computing resources to jobs is crucial for load balancing and providing online services to a large number of user requests [23]. The current state-of-the-art solutions have shortcomings when addressing the social resource allocation problem (SRAP). In particular, they often fail to address the requirements related to the social network, which restricts interaction among the agents [14]. An effective approach needs to model the proximity and hierarchy among the agents. It should consider social networks with limited connectivity, and allow distribution of resources along the network among agents who may not be directly connected. In addition, a realistic approach needs to consider the global transfer cost of resources along the network. Furthermore, in dynamic environments, where the set of tasks and resources constantly change over time, the approach needs to adapt to the changes and adjust the allocations in a timely manner. In this thesis, we aim to address these limitations by presenting new centralized and decentralized approaches in order to solve the SRAP. Our research objectives are to develop novel mechanisms which are adaptive

in dynamic environments, and ensure efficiency and scalability in large-scale settings.

1.3 Centralized and Decentralized Approaches

We model the social resource allocation problem (SRAP) using the multi-agent paradigm [56]. Distributed intelligent systems or multi-agent systems are increasingly employed in practical solutions, in which automated coordination among the participants is crucial in achieving the system objectives. In our model, each agent is an autonomous software system, representing an entity or an organization, and it is responsible for allocation of required resources to its delegated tasks. In this thesis, we present different centralized and decentralized approaches in order to solve the SRAP. In the centralized approach, a single external entity such as a higher-level organization, who has complete knowledge of the social network including all agents' tasks and resources, decides on allocation of required resources in order to perform all the tasks. On the other hand, in a decentralized, self-organizing approach [59], we develop an interaction protocol, which allows individual agents to negotiate with each other on allocation of resources to their tasks on demand; i.e. when an agent recognizes lack of resources to perform its current tasks, it initiates a negotiation process with its neighbors in the social network. As a concrete example for the decentralized approach, imagine a network of hospitals across a metropolitan area, each operating autonomously but collaboratively to optimize the allocation of both human and material resources. Each hospital employs an autonomous software agent, which is responsible for allocation of required resources for its scheduled operations. Consider the scenario when the hospitals are experiencing a surge in patient admissions due to a pandemic. The hospitals are already operating near capacity, and face shortage of resources. Each hospital's agent constantly monitors its current supply and demand quantities for upcoming operations. Hospital A determines a critical shortage of ventilators and ICU staff, and initiates the negotiation process by sending out request messages to neighboring hospitals B and C in the network. The corresponding agents B and C evaluate their own supply, and determine whether they have a surplus of requested resources. Agent B offers to lend ventilators, but not ICU staff due to its own requirements. Agent C has a balanced resource pool and offers both ventilators and ICU staff. Then, agent A confirms the offers and the resources are transported to hospital A. Therefore, the hospital A is able to manage a sudden surge in the number of patients without compromising patient care quality. At the same time, the hospi-

tals B and C, while helping the hospital A, maintain a sufficient level of supply to continue their operations.

1.4 Contributions

In this thesis, we investigate the SRAP by focusing on social networks with partial connectivity in dynamic environments. We present novel mechanisms in both centralized and decentralized approaches by incorporating greedy and learning-based algorithms. In particular, we integrate deep reinforcement learning methods into the negotiation process in order to enable the agents to make better decisions when interacting with their neighbors. Moreover, the protocol enables the agents to cascade their messages along the network in order to interact and coordinate with each other in incomplete networks. In addition, we formulate the SRAP as an integer linear programming (ILP) to serve as a baseline centralized solution for small-scale settings. In summary, our main contributions are as follows:

- A decentralized negotiation-based protocol with cascading behavior using greedy methods: *DEC-GRD*
- A decentralized negotiation-based protocol with cascading behavior using learning-based methods: *DEC-RL*
- A centralized greedy approach: *CEN-GRD*
- A centralized learning-based approach: *CEN-RL*
- A centralized optimal solution: *OPT*

We evaluate the performance of our greedy and learning-based approaches using three prominent types of social networks, namely small-world, scale-free, and random networks. We develop simulation models of agents with their tasks and resources in dynamic environments, where the tasks and resources arrive over time. We perform an empirical evaluation of our approaches by comparing their simulation results. In our simulation experiments, we vary different parameters including available resources, resource types, number of agents, number of tasks, resource transfer cost, and the network degree. Our simulation results provide comprehensive analysis of different approaches across different types of social networks, highlighting their strengths and

limitations in various parameter spaces. The findings of this thesis offer valuable insights into the efficiency and scalability of different centralized and decentralized approaches. Thus, they have practical implications for a wide range of real-world applications, such as healthcare resource management, disaster management, cloud resource management, etc.

1.5 Thesis Outline

The rest of this thesis is structured in the following manner. In Chapter 2, we review the background and existing work related to the SRAP. Chapter 3 states our research objectives and methods. In Chapter 4, we formally define the SRAP. Chapter 5 presents our centralized approaches to solve the SRAP, including optimal, greedy, and learning-based. Chapter 6 presents our decentralized negotiation-based protocol with greedy and learning-based variations. In Chapter 7, first we describe our simulation models, learning models, experimental setup, and parameter settings used for evaluation. Then we compare and discuss our simulation results for different approaches. Finally, Chapter 8 concludes the thesis, summarizing our findings and outlining directions for future work; and Appendix A provides an index of all the notational conventions used in our algorithms.

Chapter 2

Background and Related Work

In this chapter, we review the background definitions and context related to our work. We also review existing task/resource allocation mechanisms using centralized and decentralized approaches. Then we discuss their limitations when addressing the key characteristics of the social resource allocation problem (SRAP) described in Chapter 1 and formalized in Chapter 4.

2.1 Multi-Agent Systems

The most widely acceptable definition of an agent is the following.

"An agent is an encapsulated computational system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [57]."

A multi-agent system is composed of multiple intelligent agents interacting with each other in an environment. The individual agents may have their own individual goals or share common goals. They may collaborate or compete with each other in order to achieve their goals. They can also share their knowledge by an agreed language, and a communication protocol [56].

Multi-agent systems can incorporate self-organisation mechanisms [59], which may lead to a common improvement. Self-organisation is a localized and decentralized operation, which allows a distributed system to be more scalable and highly adaptive to dynamic requirements [17]. Serugendo et al. [46] defined *self-organization* and *strong self-organizing systems* as follows.

Definition 1. (Self-Organization)*: "is the mechanism or the process enabling a sys-*

tem to change its organization without explicit external command during its execution time [46]."

Definition 2. (Strong Self-Organizing Systems)*: "are those systems where there is no explicit central control either internal or external [46]."*

Ye et al. [59] present a survey of self-organization mechanisms in multi-agent systems. They classify the mechanisms based on their objectives; i.e. what the self-organization mechanism is designed for. They describe task and resource allocation problems as one of the main important research issues in multi-agent systems, which use self-organization techniques.

2.2 Task/Resource Allocation in Social Networks

Many real-world issues can be modeled as task allocation or resource allocation problems in multi-agent social networks. Task allocation mechanisms deal with the problem of how to efficiently allocate tasks among multiple agents considering their capabilities and constraints. Similarly, resource allocation mechanisms focuses on how to efficiently distribute the resources required by agents to achieve their individual or common goals. The social network constraints determine the social relationships among the agents by specifying the agents' interaction topology, or task/resource allocation rules among the neighboring agents. Many researchers have investigated task/resource allocation problems in the last two decades with different assumptions and emphases. The related literature ranges from centralized to decentralized approaches, assuming fully collaborative to self-interested agents, considering partially or fully connected networks.

2.3 Centralized Approach

In a centralized approach, a single agent controls certain actions of all other agents in the system. In other words, the coordination among the agents is done by a central entity. Thus, in centralized task/resource allocation mechanisms, all the decisions regarding task and resource allocations are made by such single entity and communicated to all the other agents. This approach simplifies the system design and implementation. The interaction protocol is simple as all the agents only communicate with the center, and the message complexity is low. Moreover, the central

entity has a global knowledge of the system, potentially capable of finding the optimal solution. However, there are limitations with regards to the system's reliability and scalability. It is obvious that the central entity becomes a single point of failure; i.e. if it fails, the multi-agent system fails to operate. In addition, centralized control mechanisms tend to perform with limited scalability as the number of agents, number of tasks, or complexity of the tasks increases.

2.4 Decentralized Approach

In a decentralized approach, there is no single entity that has complete control; instead individual agents locally make decisions based on their local knowledge. They interact with each other using a protocol in order to coordinate their activities. A decentralized multi-agent system has better scalability as the number of agents or tasks increases. It can also be more robust and fault-tolerant since there is no single point of failure; i.e. if one or more agents fail to operate, the rest of the agents may be able to resume their activities. Moreover, it is easier to add new agents to the system in open environments. However, there are limitations with regards to the message complexity and system's efficiency. The interaction protocols are more complex to ensure effective coordination among the agents, and consequently the number of messages communicated among the agents is high. Moreover, since agents make decisions based on their local knowledge only, the multi-agent system may only achieve sub-optimal results. The self-organizing mechanisms are decentralized as well, and have been developed for task and resource allocations in dynamic and open environments [59]. In such environments, the set of tasks and agents may change dynamically over time. In a self-organized allocation process, there is no global, computationally complex mechanism. Instead, the individual agents decide on the allocations locally. Hence, the self-organized allocation is decentralized and robust to failures in agents' performance and communication, and works well in large-scale multi-agent systems.

2.5 Existing Mechanisms

In this section, we review existing centralized and decentralized mechanisms for task and resource allocation problems. The Contract Net Protocol (CNP) [50] introduced in 1980 is one of the first and most well-known multi-agent protocols. It presents an auction-based approach for decentralized task allocation problem in a multi-agent

system. As described by the Foundation for Intelligent Physical Agents (FIPA) [19], CNP allows a manager and participants to interact through a four-phase negotiation process. Initially, the manager initiates the process by sending a Call For Proposals (CFP) for a task specification. The participants who receive the CFP, evaluate it and decide whether to send a proposal or a refuse message. The CFP includes a deadline by which the manager starts evaluating all the received proposals and then awards a contract to the participant with the most appropriate proposal, and rejects all the other proposals. The participant who received the contract starts working on the task and eventually sends a completion message or a failure message to the manager. Many researchers have proposed extensions to the original CNP as follows [8].

The original CNP was designed for cooperative agents only, and formal decision models for individual agents were undefined. Sandholm [44] developed a formalization based on local cost calculations in all stages of announcing, bidding, and awarding. Their formal mechanism generalized the CNP to be applicable to both cooperative and competitive scenarios. In addition, their protocol allows negotiation over a set of tasks, which is awarded to the agent with the most inexpensive bid. Sandholm and Lesser [45] also introduced unilateral decommitting mechanisms with penalties applied for self-interested agents.

Knabe et al. [29] noted that early commitment in original CNP leads to suboptimal outcomes when there are several tasks to be allocated to agents concurrently running the protocol. They proposed the Contract Net With Confirmation Protocol (CNCP), which avoids the problems of early commitment by introducing two more interaction phases: a request for confirmation and a reply to it. Hence, a participant can reply to all incoming call for proposals without committing to any proposal and allocating the resources early. In a similar work, Aknine et al. [3] extended CNP by introducing two more interaction phases; they replaced the bidding phase and assignment phase of original CNP with four phases of PreBidding, DefinitiveBidding, PreAssignment and DefinitiveAssignment. Their enhancement allowed multiple managers and participants negotiating concurrently, and reduced decommitment situations by allowing the participants to propose temporary bids without having to commit early.

Fatima and Wooldridge [18] developed a self-organizing multi-agent system called TRACE, which allows a collection of organizations to cooperate in dynamic environments. TRACE includes a task allocation protocol (TAC) and a resource allocation protocol (RAP). TAP is a contract net-based protocol, which enables the individual

agents in an organization to allocate tasks among themselves based on their capabilities and local schedules. In a dynamic environment, each organization arbitrarily receives requests for different tasks, so that one organization could have shortage of resources, while another one could have extra resources. RAP allows the organizations to reallocate resources between themselves in accordance with their demands. RAP uses ideas from computational market systems, by calculating the total supply and demand for resources in the system and determining the equilibrium price.

An et al. [4] proposed a negotiation-based resource allocation mechanism for self-interested agents in electronic commerce markets. In their model, multiple buyer and seller agents are allowed to negotiate with each other concurrently for trading multiple resources. Agents make tentative agreements for each resource before committing to final agreements. They determine a reserve price for each resource and adjust the tentative agreements dynamically based on a time-dependent strategy in response to the market situation. In their model, agents have incomplete information about each other. Each agent knows the number of trading partners and competitors, and the trading partners' reserve price distributions. For example, a buyer agent knows the reserve price distribution of a seller agent. However, the negotiation strategy and status of agents are their private information.

Polajnar et. al. [41] proposed the Mutual Assistance Protocol (MAP), which enables an agent team to incorporate helpful behavior in performing actions (Action MAP) and providing resources (Resource MAP). The interaction sequence in MAP is similar to the one in the Contract Net Protocol. The deliberation on helpful behavior is based on the interest of the team, and is jointly determined by two agents through a three-phase negotiation process. A team is given a task, with each agent addressing a subtask. An agent performs actions in order to accomplish its subtask. Individual agents have different abilities with respect to performing actions, hence they can help each other when need arises. Each individual agent estimates the team impact of helpful behavior to its own individual plan. Further extensions of Action MAP allowed both requester-initiated and helper-initiated versions [37], and combining the two approaches in one bidirectional protocol [33].

Kash et al. [28] proposed a centralized resource allocation mechanism based on fair division theory in dynamic settings where agents arrive over time. Their objective is to find an equilibrium among all agents such that no one has an incentive to change the resource allocations. Their method requires a central controller and global information. Georgara et al. [21] studied the problem of allocating teams of agents to

a collection of tasks, where each team is in charge of a specific task. They proposed an anytime heuristic algorithm, which first finds an initial feasible allocation, then improves it by swapping agents among teams.

Macarthur et al. [32] proposed a decentralized anytime algorithm for task allocation in dynamic environments, where the set of agents and tasks may change over time. They improved the efficiency and scalability of an existing fast-max-sum algorithm by introducing an online pruning procedure that simplifies the problem, and a branch-and-bound technique that reduces the search space. However, they do not address the task allocation problem in a social network, which restricts the agents' interactions to their direct neighbors. Delgado-Roman et al. [15] proposed the Coalition Oriented Sensing Algorithm (COSA) for coordination among nodes in wireless sensor networks. COSA allows a network to save energy and extend its useful battery life by avoiding redundant sensing in dynamic environments. It is fully decentralized by making the nodes autonomous and relies on peer to peer negotiation among neighboring nodes in order to reallocate sensing tasks over time.

Nongaillard et al. [38] proposes a decentralized negotiation-based approach for the multi-agent resource allocation problem. They consider different social welfare definitions including utilitarian, egalitarian and nash product. In further study [39], they focus on maximizing nash product, i.e. the product of agents' utilities, in order to promote fairness in resource allocation. In their model, only bilateral transactions involving only two agents at a time, such as gifts and swaps, are allowed. Zhang et al. [60] presents a multi-agent learning approach for online resource allocation in computing cluster networks. There is no central controller in their approach and agents do not have access to the global knowledge. While the computing resources are stationary, the tasks can be moved along the cluster network. Each agent representing a cluster uses local knowledge to make decisions on local task allocation, and also task routing to its unsaturated neighboring clusters. In this approach, each agent learns its own policy, and hence the whole system can be load-balanced in a self-organized manner.

De Weerdt et al. [13, 14] studied a variant of the task allocation problem, where the agents are connected in a social network. In their model, the agents are only allowed to interact with their direct neighbors in the network. Each agent is given a set of tasks to perform, and has a set of resources of different types. Each task requires some resources, and has a fixed benefit. Each agent is only allowed to use resources provided by its neighbors. The problem is to find out which tasks to perform, and

which resources of which neighbors to use in order to maximize the total benefit for all agents. They proved that the problem is NP-hard for an arbitrary graph, and proposed a greedy distributed protocol based on the Contract-net protocol, in which an agent in charge of a task acts like a manager and its neighbors act like participants.

Li et al. [31] present a group-oriented approach to solve a task allocation problem in large-scale multi-agent systems. The agents relationships are structured through social networks. They focus on allocating tasks to pre-existing agent groups based on their social value, which is modeled based on their skills, reputation, communication costs, and social contexts. In contrast with other works that involve team formation, their method does not require forming new agent groups for each task, instead selecting the most suitable group based on a calculated comprehensive social value for each candidate group. If the selected agent group is not able to perform the task independently, it can receive help from its neighbor groups in the social network. Once a task is accomplished, the agents gain reputation reward based on their contribution and performance, which promotes their social values.

Gao et al. [20] present a hierarchical multi-agent optimization method, combining a genetic algorithm with multi-agent optimization, specifically designed for resource allocation in cloud computing. Their objective is to maximize resource utilization of CPU, GPU, and memory, and minimize the bandwidth cost. They aim to address this problem for large-scale cloud computing environments. Their approach includes two processes. First, they present an improved genetic algorithm used for identifying an optimal set of service agents for performing tasks given resource utilization. Then, they use a multi-agent optimization algorithm in order to minimize bandwidth costs by swapping the sub-tasks among the service agents. This work can be classified as a hybrid approach including both centralized planning and decentralized optimization processes.

Chevaleyre et al. [11] focuses on the problem of fairness in multi-agent resource allocation. Their work provides a theoretical framework for fair allocation of resources in a decentralized approach. Agents are allowed to negotiate with each other and exchange resources. The objective is to find a proportionally fair and envy-free allocation. They study the convergence properties of their decentralized approach, and identify certain conditions under which the local negotiations will converge to envy-freeness. They also extend their work to scenarios where agents are connected in a social network, constraining their interactions.

Rahimzadeh et al. [42] addresses the problem of reliable task allocation in social

networked multi-agent systems. They focus on scenarios where agents may fail to perform their delegated tasks, leading to task rescheduling and increased execution time. Their objective is to decrease the task execution time by minimizing communication and waiting times, while enhancing the task execution success rate. They consider agents' talents in terms of resource availability, and distances between them. Their approach involves a decentralized negotiation-based protocol which takes the reliability of agents into account in the task allocation process. The protocol allows the agents to obtain information about each other and expand a reliability tree of all agents. The tree is utilized to identify the most reliable paths and agents, when selecting manager and resource provider agents.

Sless et al. [49] studies the problem of coalition formation in multi-agent systems where the values of coalitions are based on the social network connecting the agents. They present a centralized game-theoretic approach in which a central organizer is able to form coalitions and create new connections among the agents at a certain cost. The objective of the organizer is to maximize social welfare by building effective and core-stable coalition structures to perform a given set of tasks. In their later work [48], they consider the formation of exactly k coalitions in the social network. They present polynomial-time heuristic algorithms to find the coalition structure that offer near-optimal social welfare. They also show that finding core-stable coalitions is intractable even for a fixed k, but checking if a coalition structure is core-stable can be done in polynomial time.

Doucette et al. [16] presents a decentralized approach for allocating resources to tasks in multi-agent systems. They focus on scenarios in which tasks arrive dynamically over time. Their algorithm design leverages preemption, i.e. revoking allocation of resources to tasks in progress, and reallocating them to newly arrived tasks that have higher utilities. In their model, task agents represent tasks and proxy agents represent resources. The task agents request resources from proxy agents. They are also able to learn about system congestion and churn in order to make better requests. If a request made by a task agent demonstrates a higher utility than the current task, the proxy agent preempts the resource allocation, and reallocates the resource to the task with higher utility.

Wang and Jiang [53] present a community-aware task allocation model for multi-agent social networks. In contrast with global-aware task allocation approaches, their model reduces the overall communication cost by constraining collaboration in agent local communities, i.e. each agent is only allowed to negotiate within its intracommu-

nity. Their approach includes a heuristic algorithm for task selection and allocation to communities, and a decentralized negotiation process for resource contributions by agents in each community. Their model is suitable for real-world scenarios where agents are organized in communities. In another work [54], they present a decentralized approach for allocating complex tasks to individuals in social networks. They focus on agent mobility and teamwork in order to enhance load balancing. The tasks include multiple subtasks, hence require cooperation among the individuals. The mobile agents are able to transport themselves to each subtask of each complex task, and form cooperative teams to maximize social effectiveness by minimizing waiting and communication costs.

Yang and Danos [58] present a learn-and-adapt model for team formation and task allocation in social networks. The model allows decentralized reconfiguration and neighborhood adjustment in response to dynamic task demands, agent failures, and new agents. In their model, an agent continuously adjusts its neighbourhood based on local metrics such as task success rate and team waste rate. Therefore, the multi-agent network as a whole can cope with changes in a self-organized approach, involving local role changes and rewiring of network connections.

Jiang et al. [27] address undependable multi-agent systems in social networks, in which agents may be deceptive. They focus on scenarios where agents may fabricate their resources during task allocation, but not really contribute resources when performing the task. They present a task allocation mechanism which allows negotiation among agents based on their reputation in task allocation; i.e. an agent's past behavior impacts the probability of its involvement in task allocation. An agent is rewarded, when it provides dependable resources and the task execution is successful; and it is penalized when its allocated resources are undependable and the task fails. This mechanism encourages truthful resource contribution and improves the reliability of resources. Thus, it aims to maximize task success rate and minimize resource access time.

2.6 Limitations in Existing Mechanisms

In this section, we review shortcomings in the existing centralized and decentralized approaches, when addressing resource allocation problems in multi-agent social networks.

Proximity and hierarchy among agents - A resource allocation problem becomes more complex in multi-agent social networks, when the network is incomplete and each agent is only allowed to interact with its direct neighbors. The proximity and hierarchy among the agents impact interaction cost in terms of number of messages, and the waiting time for receiving messages. In particular, this is more critical in real-time systems with strict timing constraints. In addition, the social network constraints impact cost of transferring tasks or resources among the agent locations in the network. The existing centralized and decentralized approaches do not model the proximity among the agents, and do not consider the global transfer cost of tasks and resources. They only allow swapping tasks or allocating resources among neighboring agents or local communities. Moreover, the existing decentralized approaches do not model cascading of messages along the network, in order to allow forming global coalitions.

Dynamic environments - In environments with dynamic changes of supply and demand quantities, the existing approaches are not effective in large-scale multi-agent social networks. In other words, when the set of tasks and resources change over time, the multi-agent system needs to adapt to the dynamics by constantly making decisions on allocating currently available resources to updated set of tasks. This process becomes more complex when the social network graph is incomplete, i.e. each agent is only connected to a limited number of other agents in the network. In dealing with this problem, existing centralized approaches are not scalable, and need to recompute the solution for the whole network every time there is a change. On the other hand, the existing decentralized approaches have good scalability, however, they do not present efficient solutions when dealing with the social network constraints.

Tasks with urgencies and perishable resources - Existing centralized and decentralized approaches are inefficient when dealing with tasks that have deadlines; in particular, in environments with unreliable communication channel and possibility of agent failures. The centralized approaches have single point of failure and scalability issues in large-scale environments, and the decentralized approaches miss temporal constraints in the allocation mechanism or the negotiation process. Likewise, existing solutions are inefficient when dealing with perishable resources as they do not include any temporal constraint related to expiry date of resources in the decision making process. For instance, in negotiation protocols, the initial call or request may include

a deadline constraint based on the urgency of its tasks. Then, a potential provider also needs to determine a deadline for its offer based on the expiry dates of its resources. Without a deadline for the offer, the provider might be blocked if it waits indefinitely for a confirmation. Given all this, one needs to find a way to determine such timing constraints, which can be challenging in different environments.

Chapter 3

Research Objectives and Methods

In this chapter, we present our research objectives and methods for solving a variant of resource allocation problems in multi-agent social networks, called the social resource allocation problem (SRAP) described in chapter 1. The formal definitions and functions are presented in chapter 4.

3.1 Research Objectives

We investigate different centralized and decentralized approaches, which enable the agents in a social network to optimize their resource allocation processes. We aim to design mechanisms for dynamic environments, where tasks and resources arrive over time. The mechanisms need to deal with multiple items of different resource types, and multiple tasks with different resource requirements. In addition, we consider the social network constraints including the proximity and hierarchy among the agents in our mechanisms.

We have reviewed the existing mechanisms addressing these problems and their limitations in chapter 2. The current state-of-the-art solutions have shortcomings when addressing the key characteristics of these problems described in chapter 1. Against this background, we aim to develop a coordination mechanism which satisfies the following set of requirements:

1. **Adaptive behavior** - A mechanism should operate in dynamic environments, and be able to adapt to changes in supply and demand quantities. For example, it must be adaptive to a surge in incoming tasks or shortage of required resources.

2. **Efficiency** - A mechanism should be efficient in terms of minimizing transfer cost and maximizing gained utilities; in particular, when addressing the proximity and hierarchy among the agents, and the complex resource requirements.

3. **Scalability** - A mechanism should find a solution efficiently even in large-scale environments, and be scalable with increasing number of agents, tasks, resource types, and in different social network structures.

4. **Robustness** - A mechanism should be robust when an agent is suddenly unresponsive in an interaction with others. In addition, the mechanism should be tolerant to unreliable communication channels with unpredictable transmission delay.

5. **Self-organization** - A mechanism should enable the multi-agent system to be strongly self-organized without relying on a central entity either internal or external. Hence, the system can avoid a single point of failure. This requirement can be met only if the mechanism is completely decentralized. Thus, we disregard it when developing centralized approaches.

3.2 Research Methods

To achieve our research objectives, we develop a decentralized multi-agent interaction protocol which allows the agents to coordinate with each other on allocation of resources to their tasks. Using this protocol, if an agent recognizes lack of resources to perform its current tasks, it initializes a negotiation process with its direct neighbors in the social network. In particular, the protocol allows an agent in need to receive partial quantities of different resources types, from multiple provider agents; i.e. each provider is allowed to contribute partially in order to help the agent in need of resources. Moreover, in a social network that restricts agents to only interact with their direct neighbors, the protocol allows the agents to cascade requests and combine offered resources along the network by incorporating timeouts in their interactions. Hence, it allows the multi-agent system to be self-organized, as all decisions are made by individual agents and there is no central controller either internal or external.

In our decentralized protocol design, we incorporate different techniques including greedy and reinforcement learning algorithms in the negotiation process. In addition, we develop centralized greedy and reinforcement learning algorithms as alternative

solutions for the SRAP, and compare the centralized results with the ones for the decentralized protocol. Furthermore, we compute the centralized optimal solution using integer linear programming (ILP) for small-scale settings in order to measure the quality of our greedy and learning-based results.

In our formulations, we define utility and cost functions used in agents' decision making processes for expressing their benefit and cost of resource allocation. We model the proximity and hierarchy among the agents using an undirected weighted graph which determines the cost it takes for each resource type to be transferred along the network. We define different types of resources and tasks, and apply combinatorial optimization techniques to deal with resource quantities in agents' decision making processes. The agents share a common goal of increasing a utilitarian social welfare, defined as the sum of individual utilities minus all transfer costs for all agents.

We evaluate our centralized and decentralized approaches based on our research objectives given in Section 3.1, by developing simulation models of tasks, resources, agents and their interactions in a dynamic environment. Our evaluation involves simulation experiments in different types of social networks, namely small-world, scale-free, and random networks. We investigate whether our solutions adapt to changes in dynamic environments, exhibit efficiency and robustness, and scale well to large-scale applications. We perform an empirical evaluation by varying different parameters including available resources, resource types, task requirements, number of tasks, transfer cost, number of agents, and social network degree. The simulation studies of our multi-agent mechanisms in such scenarios reveal the effectiveness of employing such solutions in real-world applications.

Chapter 4

Social Resource Allocation Problem

In this chapter, we describe and formally define the problem of resource allocation in a social network. Consider an environment, in which a group of agents are delegated with tasks which arrive over time. Each task is assigned to a specific agent and has a specific resource requirements. There are different types of resources. A task can be accomplished only if all the required resources are allocated to that task. Each agent also procures resources over time. The task assignment and resource procurement frequencies can be different for each individual agent, and it can change unpredictably over time.

In addition, the agents are distributed over a social network, which allows them to interact and help each other by providing resources. The environment in which the agents are operating is considered to be dynamic in two ways: the tasks may arrive at different frequencies over time; also, the resources may be procured at different frequencies over time. The knowledge regarding tasks and resources is local, i.e. an agent only knows its own current tasks and resources; hence each agent individually decides on which tasks to perform given its available resources at a time. In our model, the individual agents are self-interested, but may have a common motivation, mandated at the multi-agent system level to collaborate, such as sharing common goals in completing their tasks.

Real-world applications can be seen in different domains. For example, in the healthcare environment, autonomous agents could represent hospitals, and tasks could be the patients who arrive unpredictably, in particular in emergency cases. The

resources required for each task could include medical supplies, hospital beds and equipment, or specific expertise. The social network enables the different hospitals in a geographic region to interact and collaborate on demand. Another example can be seen in cloud computing, where agents could represent data centers distributed across multiple locations; and tasks could be computational jobs arriving at these centers with unknown frequencies. The required resources for the jobs could be CPU time, memory, storage, and network bandwidth. The data centers are allowed to interact using the social network in the cloud environment to improve load-balancing and optimize the overall system performance.

Formally, we consider a set of m agents, $A = \{a_1, \ldots, a_m\}$. Each agent $a \in A$ receives tasks over time from a domain, $T = \{t_1, \ldots, t_n\}$. Each task $t \in T$ requires resources defined by a function: *rqr*: $T \times R \to \mathbb{N}$, where $R = \{r_1, \ldots, r_l\}$ is set of resource types. Each task has a *utility* indicating its importance. We denote the location of a task by a function: *loc*: $T \to A$, which is the agent owning the task, called the task manager. The set of tasks currently assigned to an agent is defined by a function: *tsk*: $A \to P(T)$, where $P(T)$ denotes the power set of the task domain. Each agent $a \in A$ also receives resources over time. The number of resource items currently available to an agent is defined by a function: *rsc*: $A \times R \to \mathbb{N}$. Moreover, we assume agents are connected by a social network defined as follows.

Definition 3. (Social Network)*: An agent social network $SN = (A, E)$ is an undirected weighted connected graph, where vertices $a \in A$ are agents, and each edge $e_{i,j} \in E$ indicates a social connection between agents a_i and a_j, and the weight of $e_{i,j}$ represents the distance between a_i and a_j.*

The social network constrains the agents' interactions so that they can only communicate and share information with their direct neighbors. Resource items can be allocated by an agent to any other agent in the network even if they are not neighbors, considering the cost of transfer between the two agents, defined as follows.

Definition 4. (Transfer Cost)*: The cost of transferring resources between two agents is determined by a function:* tcs*: $\mathbb{N} \times \mathbb{N} \times R \to \mathbb{N}$. The first argument is the distance between them and the second is the number of items; i.e. $tcs(d, n, r)$ returns the transfer cost for moving n resource items of type r across a distance of d between two agents.*

We denote the distance between any pair of agents in the network by a function: *dst*: $A \times A \to \mathbb{N}$. If two agents are not connected, the distance is determined as the

total weight of a shortest path between them in the network. The agents are not able to swap their tasks, but they are allowed to allocate resources to each other along the network. A resource item can be reallocated multiple times along the network until it is consumed in performing a task. The allocation of resources to agents' tasks is defined as follows.

Definition 5. (Resource Allocation)*: Given a set of agents A, a set of tasks T, and a set of resource types R, a resource allocation is a function: $o : T \times A \times R \to \mathbb{N}$. The value of $o(t, a, r)$ specifies the number of resources of type r allocated by agent a to task t; subject to the following constraints:*

- $\forall t \in T, a \in A, r \in R : 0 \leq o(t, a, r) \leq rqr(t, r)$*; i.e. each agent may allocate resources to each task only up to the required quantity for each resource type.*
- $\forall a \in A, r \in R : \sum_{t \in T} o(t, a, r) \leq rsc(a, r)$*; i.e. each agent can only allocate resources to each task up to its available quantity for each resource type.*

Once resources are allocated, each agent gains utilities by performing its own tasks and incurs costs of transfers along the network as a result of the allocation. We define the value of an allocation as follows.

Definition 6. (Social Welfare)*: The social welfare of a resource allocation o is defined by the function* swf*: $O \to \mathbb{N}$, where O is the set of all possible resource allocation functions. The function* swf(o) *calculates the sum of the differences in utilities gained and transfer costs incurred by all agents: $\sum_{a \in A}(util_a - cost_a)$, where $util_a$ and $cost_a$ represent the total utilities gained and total transfer costs incurred by agent a, respectively.*

Now, given sets of all tasks and resources assigned to the agents, the problem is to find out which tasks of which agents to perform, and which resources of which agents to use for these tasks, in order to maximize the social welfare, defined as follows.

Definition 7. (Social Resource Allocation Problem (SRAP))*: Given a set of agents A, connected by a social network $SN = (A, E)$, and the current sets of tasks and resources assigned to them, the problem is to find a resource allocation function, $o^* \in O$, that maximizes the social welfare; i.e., $o^* = \arg\max_{o \in O}$* swf(o).

The complexity of finding the optimal solution for the resource allocation problem without the condition of the social network is NP-hard [47], which comes from the

exponential number of subsets of the set of tasks assigned to the agents. De Weerdt et al. [13, 14] showed that this problem with an arbitrary unweighted social network is also NP-complete, even when the utility of each task is 1, and the quantity of all required and available resources is 1.

Proposition 1. *The complexity of finding the optimal solution for* SRAP *is NP-hard*

Proof. The weighted version of social network in *SRAP* can be reduced to the unweighted one by assigning a weight of 1 to all the edges in the social network.

□

Chapter 5

Centralized Approaches

In this chapter, we introduce centralized approaches to solve the social resource allocation problem (SRAP). We consider a centralized agent called the master agent that receives all the tasks and resources information from all the individual agents in SRAP and knows the social network constraints. Then, it decides on how to allocate required resources to the tasks for all the agents in the network. First, we present an optimal, exact solution by defining SRAP as an integer linear program (ILP). Since the complexity of SRAP is NP-complete, the optimal solution runs in exponential time. Hence, we present a heuristic approach and further a reinforcement learning approach in order to solve large-scale problems in polynomial time.

5.1 Optimal Solution

We can formulate SRAP as an integer linear programming (ILP) problem P_{opt} as follows. We introduce two types of variables: a binary variable $y_j \in 0, 1$ for $1 \leq j \leq n$ describes whether or not all required resources for task j are allocated, and an integer variable $x_{ijk}(\forall i, j, k$, where $1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq l)$ denotes the amount of resource type r_k agent a_i allocates to task t_j. Then, we define the objective function as follows.

$$\max \quad \sum_{j=1}^{n}(y_j * util_j - \sum_{i=1}^{m}\sum_{k=1}^{l} tcs(dst(a_i, loc(t_j)), x_{ijk}, r_k) \tag{5.1}$$

$$\text{s.t.} \quad \forall j,k(1 \leq j \leq n, 1 \leq k \leq l) \sum_{i=1}^{m} x_{ijk} \geq y_j * rqr(t_j, r_k) \tag{5.2}$$

$$\text{and} \quad \forall i,k(1 \leq i \leq m, 1 \leq k \leq l) \sum_{j=1}^{n} x_{ijk} \leq rsc(a_i, r_k) \tag{5.3}$$

The constraint (5.2) ensures that there are sufficient resources of all types from all agents for each selected task; and the constraint (5.3) ensures that the total allocated resources of all types are not more than available to each agent.

Note that in order for this formulation to qualify as an ILP, the transfer cost function *tcs* needs to be linear. Solving this ILP finds an optimal solution for the SRAP. However, its run time is exponential in the number of tasks, agents, and resource types. It is not practical to use it for large-scale problems, or in dynamic environments in which the input variables change over time. In this thesis, we use the optimal result as a basis for evaluating the quality of our greedy and reinforcement learning approaches.

5.2 Centralized Greedy Approach

In a centralized approach, a master agent uses a greedy algorithm by first ranking the tasks with respect to a heuristic measure defined as follows. It then tries to allocate required resources to the tasks in the order of their ranking.

Definition 8. (Task Efficiency)*: the ratio of the net utility of a task t (task utility minus the total transfer costs for all required resources computed by Algo. 2), to its total number of required resources for all types:* $(util_t - \text{cost}_{trf}) / \sum_{r \in R} rqr(t, r)$

The rationale for the efficiency heuristic is based on the idea of a greedy approximation for 0-1 knapsack problem, in which the items are ranked based on their relative value [12]. Using this heuristic, the master agent ranks the tasks in order of descending efficiency.

The master agent does not reconsider its decision once a task is selected. If the selected task is feasible based on the available resources of all agents in the social

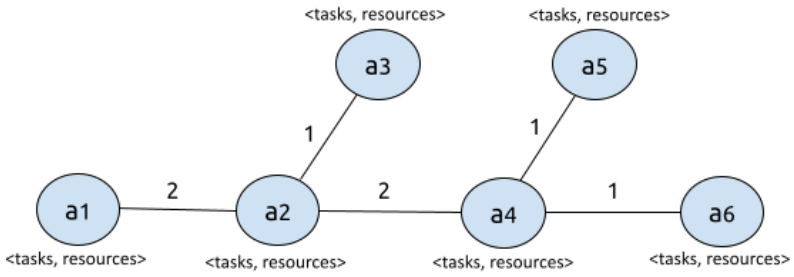

Figure 5.1: Example for the centralized greedy approach

network, it allocates the required resources in order to accomplish the task. To allocate required resources to each task, the master agent sorts the potential resource providers by their shortest distance to the task location. It further sorts the ones with the same distance by their degree in the network; i.e. the ones with more neighbors are ranked higher. Then, it first uses the resources of the task manager and then other agents in the sorted set, until all the required resources are allocated for each resource type.

The rationale for selecting the resource provider with the shortest distance to the task manager is to minimize the transfer cost. Furthermore, selecting the one with higher degree is based on the idea that later on if it lacks resources for its own selected task, it may be less costly to transfer required resources since it has more direct neighbors. Figure 5.1 illustrates a small social network including six agents. Each individual agent has its own set of tasks and resources which arrive over time. The edge weights represent distances among the agents which determine the transfer costs of resources. In this example, consider that the master agent selects a task of agent a_2 as the current best feasible task based on the task efficiency ranking. Then, the master agent allocates required resources to the a_2's selected task in the following order. First, a_2's own resources are allocated. If more resources are required, it allocates a_3's resources since it is closer to the task location, then a_4's resources are allocated because it has more neighbors than a_1, and so on.

We denote total number of resource items currently available to all the agents for each resource type by a function: trc: $R \rightarrow \mathbb{N}$. Algorithm 1 and 2 formalize the centralized greedy task selection, and the allocation of required resources to the selected task processes, respectively. Once a task is accomplished, its utility $util_t$ is added to the social welfare SW (Algo. 1, line 11); and if there are resources

transferred from other agents in the network, the cost of transfer $cost_{trf}$ for all the providers is subtracted from SW (Algo. 1, line 12). As can be seen in Algo. 2, line 12, the transfer cost tcs for each potential provider is a function of the distance dst between the task manager a_t and the provider a_p, the number of transferred resource items q_{trf}, and the resource type r.

Algorithm 1: Centralized greedy task selection process

Input : current set of all agents' tasks: T, resources types: R
Output: social welfare: SW

```
1  SW ← 0
2  sort T by task efficiency in descending order
3  foreach t in T do
4      feasible ← true
5      foreach r in R do
6          if trc(r) < rqr(t, r) then
7              feasible ← false
8              break
9      if feasible then
10         cost_trf ← allocate all required resources to t using Algo. 2
11         SW += util_t
12         SW -= cost_trf
```

Algorithm 2: Centralized greedy resource allocation process

Input : agents: A, selected task: t, resources types: R

Output: transfer cost: $cost_{trf}$

$cost_{trf} \leftarrow 0$
$a_t \leftarrow loc(t)$
$providers \leftarrow$ sort $\{a \in A | a \neq a_t\}$ by $dst(a, a_t)$ in ascending and degree of a in descending order
foreach r *in* R **do**
    $q_{rqr} \leftarrow rqr(t, r)$
    $q_{alc} \leftarrow \min(q_{rqr}, rsc(a_t, r))$
    update $rsc(a_t, r)$
    while $q_{alc} < q_{rqr}$ **do**
        $a_p \leftarrow$ next agent in *providers*
        $q_{trf} \leftarrow \min((q_{rqr} - q_{alc}), rsc(a_p, r))$
        q_{alc} += q_{trf}
        update $rsc(a_p, r)$
        $cost_{trf}$ += $tcs(dst(a_p, a_t), q_{trf}, r)$
    allocate q_{alc} resource items of type r to task t

Proposition 2. *For m agents, n tasks, and l resource types, the overall time complexity of the centralized greedy task selection process (Algo. 1), including the centralized greedy resource allocation process (Algo. 2) is $O(nm \log m + nlm)$.*

Proof. At first, in Algo. 1, the efficiency of each task is computed, which requires sorting all potential providers, and finding total transfer cost for all resource types, which takes $O(nm \log m) + O(nlm)$. In addition, sorting the tasks takes $O(n \log n)$. Then, for each task, it iterates through each resource type to check if the task is still feasible, which takes $O(nl)$. Next, in Algo. 2, again sorting the agents for each feasible task takes $O(nm \log m)$. And, in the worst case, resource allocation requires iterating over all potential providers per resource type, again taking $O(nlm)$. Combining them, the overall time complexity is $O(2nm \log m) + O(2nlm) + O(n \log n) + O(nl)$. Hence, it simplifies to $O(nm \log m + nlm)$.

□

5.3 Centralized Learning Approach

In a reinforcement learning approach, a master agent learns a policy in order to solve the social resource allocation problem (SRAP). In this approach, first we formulate SRAP as a Markov Decision Process (MDP). Then, we incorporate deep Q-learning [51] with experience replay [35, 36] and double learning [52] in order to train the master agent to better select tasks over time described as follows.

5.3.1 Markov Decision Process (MDP)

A Markov Decision Process (MDP) is a mathematical formalism used in modeling of an agent situated in an environment. In the context of reinforcement learning, at each time step, called an *episode*, the agent observes current state of the environment and selects an action based on its observation. As a result of performing the action, the agent ends up in a new state, and gains a reward. The transition model from the current state to another state might be deterministic or stochastic depending on the type of environment. In summary, an MDP is made up of four components:

- *States* - denoted by $\mathtt{s}, \mathtt{s}', ...$ are the set of predefined dimensions that represents the status of agent within its environment.
- *Actions* - denoted by $\mathtt{a}, \mathtt{a}', ...$ are the set of available decisions that an agent can make in each state.
- *Transition model* - denoted by $\mathtt{P}(\mathtt{s}'|\mathtt{s}, \mathtt{a})$, gives the probability of reaching state $\mathtt{s}'$ when performing action $\mathtt{a}$ in state $\mathtt{s}$.
- *Reward function* - denoted by $\mathtt{R}(\mathtt{s}, \mathtt{a})$, returns a value for taking action $\mathtt{a}$ in state $\mathtt{s}$.

In this thesis, in order to apply our reinforcement learning approach for the SRAP, we formulate our MDP as follows.

- *State*: a matrix that represents sets of all agents' tasks with their utilities and required resources, and all agents' available resources for each resource type. The order of agents in the state is fixed.
- *Action*: to select an agent (task manager) and allocate required resources to its best feasible task.

- *Transition Function*: the master agent allocates required resources to the selected task with a probability of 1, since situated in a deterministic environment.
- *Reward*: the task utility minus the total transfer costs for all the allocated resources.

A policy is a function from states to actions, $\pi : States \rightarrow Actions$. It specifies the best action that the agent can do in each state, e.g. for state $\mathtt{s}$, policy π returns action $\mathtt{a}$, i.e. $\pi(\mathtt{s}) = \mathtt{a}$. In reinforcement learning, the goal is to learn an optimal or near-optimal policy. Using a reinforcement learning algorithm, the policy is updated over time as the agent gains more experience by interacting with its environment.

5.3.2 Deep Q-learning

Q-learning is a model-free off-policy reinforcement learning algorithm [51]. It is model-free as it does not estimate the transition function and the reward function of the MDP. It is off-policy as it directly approximates the optimal policy, while allowing continuous exploration by following a different policy for action selection, typically using an epsilon-greedy approach. The Q-learning algorithm learns the quality of actions in each visited state; i.e. an action-value function $\mathtt{Q}(\mathtt{s}, \mathtt{a})$ that gives the expected utility of taking action $\mathtt{a}$ in state $\mathtt{s}$ and following the policy thereafter. The Q-values for each state-action pair are stored in a table called Q-table. Then, at each iteration, Q-values are adjusted using the following update rule, where γ is the discount factor, $0 \leq \gamma \leq 1$, which determines the present value of future rewards; and α is the learning rate, $0 \leq \alpha \leq 1$, which determines to what extent new information overrides old information.

$$\mathtt{Q}(\mathtt{s}, \mathtt{a}) \leftarrow \mathtt{Q}(\mathtt{s}, \mathtt{a}) + \alpha[\mathtt{R}(\mathtt{s}, \mathtt{a}) + \gamma \max_{\mathtt{a}'} \mathtt{Q}(\mathtt{s}', \mathtt{a}') - \mathtt{Q}(\mathtt{s}, \mathtt{a})] \tag{5.4}$$

In the above formula, $\mathtt{R}(\mathtt{s}, \mathtt{a}) + \gamma \max_{\mathtt{a}'} \mathtt{Q}(\mathtt{s}', \mathtt{a}') - \mathtt{Q}(\mathtt{s}, \mathtt{a})$ is the Bellman error (also known as the temporal difference error), which is the difference between the current Q-value and the target Q-value: $\mathtt{R}(\mathtt{s}, \mathtt{a}) + \gamma \max_{\mathtt{a}'} \mathtt{Q}(\mathtt{s}', \mathtt{a}')$. The Q-learning algorithm aims to minimize the Bellman error over time, converging to an optimal policy.

However, standard Q-learning is not scalable for large-scale problems, since it must compute Q-values for each state-action pair. To overcome this problem, in recent works, a function approximator has been used to estimate the Q-values. In deep

Q-learning, a deep neural network called deep Q-network (DQN) is incorporated to approximate the Q-values. Then, at each iteration i, DQN uses the backpropagation algorithm in order to adjust the network weights θ; i.e. using stochastic gradient descent in order to minimize the loss function $\mathtt{L_i}(\theta_\mathtt{i})$:

$$\mathtt{L_i}(\theta_\mathtt{i}) = [\mathtt{y_i} - \mathtt{Q}(\mathtt{s}, \mathtt{a}; \theta)]^2 \tag{5.5}$$

where $\mathtt{y_i}$ is the target:

$$\mathtt{y_i} = \mathtt{R}(\mathtt{s}, \mathtt{a}) + \gamma \max_{\mathtt{a'}} \mathtt{Q}(\mathtt{s'}, \mathtt{a'}; \theta) \tag{5.6}$$

Moreover, standard Q-learning suffers from instability and even divergence from the optimal policy, in particular when combined with a non-linear function approximator such as a neural network. The instability is due to several reasons including the correlations present in the sequence of observations, and the correlations between current network parameters and recent observations. These issues have been addressed by two techniques. First, using a mechanism called experience replay, a memory is used to store the observed transitions, and then randomly select a number of samples for batch training, therefore removing the correlation among them. Second, by incorporating another neural network, called target network $\mathtt{Q'}$, which is only periodically updated, the correlations with the target is reduced.

Furthermore, based on the idea of double learning [25], Double DQN [52] uses the target network $\mathtt{Q'}$ in order to reduce the overstimation problem in DQN, by partially decoupling action selection from the action evaluation in the target as follows, where θ and θ' are the parameters of the networks $\mathtt{Q}$ and $\mathtt{Q'}$, respectively.

$$\mathtt{y_i} = \mathtt{R}(\mathtt{s}, \mathtt{a}) + \gamma \mathtt{Q'}(\mathtt{s'}, \arg\max_{\mathtt{a'}} \mathtt{Q}(\mathtt{s'}, \mathtt{a'}; \theta); \theta') \tag{5.7}$$

5.3.3 Centralized Learning Algorithm

Based on the background in the previous section, we present an algorithm which learns a policy over many episodes in order to solve the SRAP. Our centralized learning process incorporates two neural networks, called policy and target networks. While the policy network is updated in every step, the target network is only updated every C steps [36]. Here, C represents a predetermined update frequency, determining the

intervals at which the target network is reset with the policy network to stabilize the learning process. In addition, we incorporate experience replay to enable random minibatch updates [35], which enhances the learning stability; and we apply double learning when setting the target value (Equation 5.7), which mitigates overestimation in the learning process [52].

Algorithm 3: Centralized learning process

Input : agents: A, tasks for n episodes: T_1, T_2, ..., T_n

Output: trained target action-value function: $\mathtt{Q}'$

initialize replay memory D to capacity N

initialize policy action-value function $\mathtt{Q}$ with random weights θ

initialize target action-value function $\mathtt{Q}'$ with random weights $\theta' = \theta$

for $e = 1..n$ **do**

  $T_e \leftarrow$ tasks in this episode

  $\mathtt{t} \leftarrow 1$

  while $T_e \neq \emptyset$ **do**

    $\mathtt{s_t} \leftarrow$ generate the current state

    if $\mathtt{s_t}$ *is terminal* **then**

      break *while*

    agent $\leftarrow$ with probability ϵ select a random action $\mathtt{a_t}$ otherwise select $\mathtt{a_t} = \arg\max_{\mathtt{a}} \mathtt{Q}(\mathtt{s_t}, \mathtt{a}; \theta)$

    task $\leftarrow$ select the best feasible task of *agent* with highest efficiency

    $cost_{trf} \leftarrow$ allocate required resources to *task* using Algo. 2

    $\mathtt{r_t} \leftarrow util_{task}$ - $cost_{trf}$

    $\mathtt{s_{t+1}} \leftarrow$ generate the next state

    store transition $(\mathtt{s_t}, \mathtt{a_t}, \mathtt{r_t}, \mathtt{s_{t+1}})$ in D

    sample random minibatch of transitions $(\mathtt{s_j}, \mathtt{a_j}, \mathtt{r_j}, \mathtt{s_{j+1}})$ from D

    $$\mathtt{y_j} \leftarrow \begin{cases} \mathtt{r_j} & \text{for terminal } \mathtt{s_{j+1}} \\ \mathtt{r_j} + \gamma \mathtt{Q}'(\mathtt{s_{j+1}}, \arg\max_{\mathtt{a}'} \mathtt{Q}(\mathtt{s_{j+1}}, \mathtt{a}'; \theta); \theta') & \text{otherwise} \end{cases}$$

    perform a gradient descent step on $[\mathtt{y_j} - \mathtt{Q}(\mathtt{s_j}, \mathtt{a_j}; \theta)]^2$ with respect to θ

    every C steps update the target network $\mathtt{Q}' = \mathtt{Q}$

    remove *task* from T_e

    $\mathtt{t} \mathrel{+}= 1$

Algorithm 3 formalizes this process. It uses the epsilon-greedy strategy by balancing between exploration and exploitation. With probability $1 - \epsilon$, it selects the best action based on the policy, and with probability ϵ, it selects a random action. Initially, the exploration rate ϵ is set to one. Over time, ϵ is decayed in order to further exploit the learned policy. Once the training process is done and a policy is learned, it can be used to solve the problem in one episode. The ϵ is then set to zero in order to fully exploit the acquired knowledge. Algorithm 4 formalizes this learning-based task selection process, which selects tasks based on a learned policy. We will outline the neural network architecture and other hyperparameters used in the learning process in Chapter 7: Evaluation.

Proposition 3. *For m agents, n tasks, and l resource types, the overall time complexity of the centralized learning process (Algo. 3) in each episode is $O(n^2(ml + \log n) + n(NW + m \log m))$, where N is the number of neurons, and W is the number of weights in the target neural network.*

Proof. In each episode, generating the state takes $O(mnl)$. Selecting a task manager requires a forward pass (prediction) in the target network, which generally takes $O(NW)$. Selecting the best feasible task involves sorting and checking feasibility which takes $O(n \log n) + O(nl)$. Allocating resources to the selected task takes $O(m \log m) + O(lm)$. The sampling from the replay memory can be done in a constant time $O(1)$. The backward pass (backpropagation) has a similar complexity. Combining them, the overall complexity for all tasks is $O(mln^2) + O(nNW) + O(n^2 \log n) + O(n^2 l) + O(nm \log m) + O(nlm)$. Hence, it simplifies to $O(mln^2) + O(nNW) + O(n^2 \log n) + O(nm \log m) = O(n^2(ml + \log n) + n(NW + m \log m))$

□

Algorithm 4: Centralized learning-based task selection process

Input : all agents' tasks: T, trained target action-value function: $\mathtt{Q}'$

Output: social welfare: SW

$SW \leftarrow 0$
while $T \neq \emptyset$ **do**
  $\mathtt{s} \leftarrow$ generate the current state
  if $\mathtt{s}$ *is terminal* **then**
    break *while*
  agent $\leftarrow$ select $\mathtt{a}^* = \arg\max_{\mathtt{a}} \mathtt{Q}'(\mathtt{s}, \mathtt{a}; \theta')$
  task $\leftarrow$ select the best feasible task of *agent* with highest efficiency
  $cost_{trf} \leftarrow$ allocate required resources to *task* using Algo. 2
  $SW \mathrel{+}= util_{task}$
  $SW \mathrel{-}= cost_{trf}$
  remove *task* from T

Chapter 6

A Decentralized Protocol

In this chapter, we present a negotiation-based protocol in order to solve the social resource allocation problem (SRAP) in a decentralized approach. The protocol enables the multi-agent system to be self-organized as all the decisions are made by individual agents and there is no central controller. It allows real-time concurrent negotiations in which an agent in need of resources is allowed to select multiple providers and combine their resource contributions. It also allows the agents to collaborate in a social network by cascading information along the network. In the following sections, we first present a high-level description of the protocol, and then specifications of its main interaction phases.

6.1 Protocol Description

When an agent a_i has a set of of tasks to do, it checks if it can locally allocate the required resources to fully accomplish them, otherwise it considers creating a request based on the missing quantity for each resource type and sends them to its neighbors in the social network. A neighboring agent a_j who receives a request decides whether it can fully provide the requested quantity. In that case, it considers sending an offer to a_i; otherwise, it considers cascading this request based on the new missing quantity and sends it to its own neighboring agents. In such case, a_j waits for offers from its neighbors before sending an offer to a_i. If a_j receives offers to its cascaded request, it combines them with its own offer and cascades it to a_i. Since multiple neighboring agents can offer to a request, a_i may receive multiple offers. In such case, it confirms a combination of them which maximizes the social welfare. Finally, the corresponding

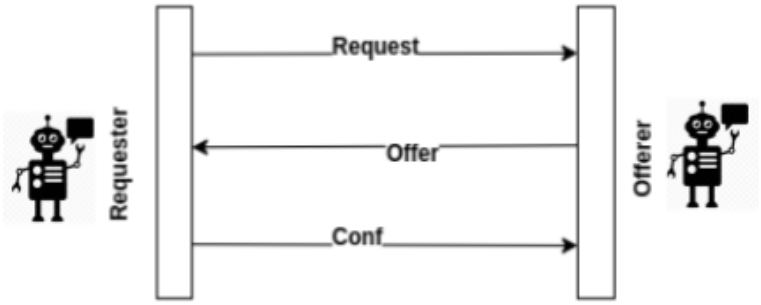

Figure 6.1: Main interaction phases in DARAP

partial confirmations are cascaded by a_j back to the original offerers.

Figure 6.1 illustrates the main interaction phases of the negotiation process assuming there are only two agents. The process is initiated by a request. After receiving an offer, the requester agent finalizes the process by confirming the offer. Figure 6.2 illustrates an example of the cascading behavior in a social network with multiple agents. In this scenario, a_1 sends a request to its only neighbor a_2; and waits for an offer. Then, a_2 decides to cascade this request to its own neighbors, a_3 and a_4. Similarly, a_4 cascades the request to a_5 and a_6. Consequently, offers made by a_5 and a_6 are combined with a_4's offer and cascaded to a_2. Similarly, a_2 cascades a combined offer to the original requester a_1. Then, the confirmation from a_1 is cascaded along the network back to all the offerers involved in this transaction. We call this negotiation process the *Decentralized Adaptive Resource Allocation Protocol (DARAP)*. Its main steps for one full negotiation round are formalized in Algorithm 5. In the next sections, we present algorithms used in the following phases of *DARAP*: requesting, cascading requests, offering, cascading offers, confirming, and cascading confirmations.

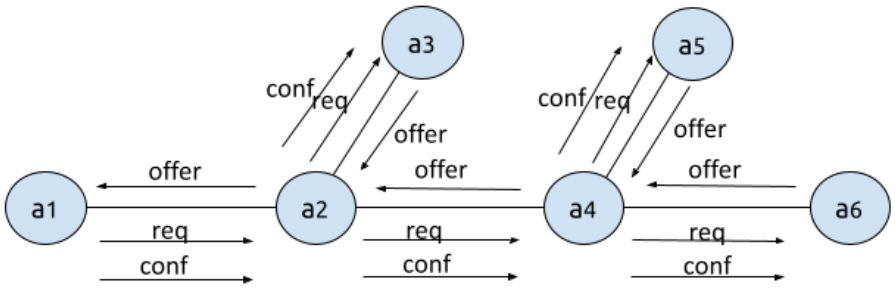

Figure 6.2: Cascading behavior in DARAP

Algorithm 5: Decentralized Adaptive Resource Allocation Protocol (DARAP)

Input : agent: a_i, agent neighbors: $\mathcal{N}_{a_i}$
Output: individual social welfare: SW_{a_i}

1 *sentReqs* ← create requests using Algo. 6
2 *receivedReqs* ← receive requests from $\mathcal{N}_{a_i}$
3 *cascadedReqs* ← cascade *receivedReqs* using Algo. 7
4 receive cascaded requests from $\mathcal{N}_{a_i}$; add to *receivedReqs*
5 *sentOffs* ← offer to *receivedReqs* using Algo. 8
6 *receivedOffs* ← receive offers from $\mathcal{N}_{a_i}$
7 *cascadedOffs* ← cascade *receivedOffs* for *cascadedReqs* using Algo. 12
8 receive cascaded offers from $\mathcal{N}_{a_i}$; add to *receivedOffs*
9 *sentConfs* ← confirm *receivedOffs* for *sentReqs* using Algo. 14
10 *receivedConfs* ← receive confirmations for *sentOffs*
11 allocate itm_{cnf} in *receivedConfs*; incur transfer costs to the requesters
12 *cascadedConfs* ← cascade confirmations in *receivedConfs* using Algo. 18
13 receive cascaded confirmations from $\mathcal{N}_{a_i}$; restore unconfirmed items
14 SW_{a_i} ← select tasks based on updated available resources

In *DARAP*, the agents interact in real time and the communication model is asynchronous; i.e. messages can be sent and received at any time, and the communication does not block other activities. The agents incorporate timeouts in their messages, and apply them in their deliberations, when sending and receiving the

messages. Applying timeouts allows the agents' communications to be asynchronous, while maintaining the reliability of their interactive behavior [10, 9]. Moreover, it allows the multi-agent system to be self-organized [59, 46], as all decisions are made by individual agents and there is no central controller either internal or external.

Proposition 4. *For m agents and l resource types, the overall message complexity of DARAP is $O(lm^2)$.*

Proof. In the worst-case scenario, suppose every agent lacks all types of resources. Moreover, suppose that each request is cascaded multiple times along the network to reach every other agent. Then, the total number of request messages would be $lm(m-1)$, as each of the m agents sends out a request for each resource type to the other $m-1$ agents. Furthermore, let's assume that for every received request, each agent (except for the one with the highest-priority request) responds with an offer. This leads to $l(m-1)(m-1)$ offer messages. The same number applies to confirmation messages, resulting in another $l(m-1)(m-1)$ messages. Thus, the total number of messages is $lm(m-1) + 2l(m-1)(m-1)$, which simplifies to $O(lm^2)$.

□

6.2 Requesting

Agent a_i evaluates its current set of tasks using a greedy approach. It sorts the tasks based on a their relative value determined as the ratio between task utility and its total number of required resources. Then it determines if it has shortage of resources given the requirements of the tasks, and creates a request based on missing quantity for each resource type. For each request, a_i computes a utility function: $utl_{req} : \mathbb{N} \longrightarrow \mathbb{N}$ computed as utility gains for partial quantities in $\{q : 0 < q \leq q_{req}\}$, where q_{req} is the missing quantity. utl_{req} indicates benefit of partial contributions, and allows the other agents to offer a partial amount, q. a_i sends each request to all of its neighbors and includes their ids in the request message, considering each agent in the social network has a unique id. A request message includes a timeout, indicating the latest time that the requester waits for potential offers. In case there are deadlines for the tasks, a request timeout can be determined by subtracting a predefined grace period from the earliest task deadline. We call this parameter *deadlineBuffer*, whose value depends on the application domain. This time interval ensures there is sufficient time to finalize the negotiation process, allocate required resources, and perform the tasks

in time. Then, a request message sent by a_i has the following format:

$$Request(r, q_{req}, utl_{req}, A_{req}, \tau_{req}, \tau^o_{req})$$

where q_{req} indicates quantity requested for resource type r, utl_{req} is the request utility function, A_{req} is the set of all agent ids that receive this request, τ_{req} is the request timeout, i.e. a_i waits for potential offers to its request until τ_{req} and then the request is expired; and τ^o_{req} is the original request timeout which initially is the same as τ_{req}, but it is different when the request is cascaded as described in the next section. Algorithm 6 formalizes the requesting process of a_i.

Algorithm 6: Requesting process

Input : agent: a_i, resource types: R

Output: requests sent by a_i: *sentReqs*

```
1  sentReqs ← ∅
2  tasks ← tsk(a_i)
3  τ_req ← (earliest deadline in tasks) - deadlineBuffer
4  foreach r in R do
5      q_rqr ← Σ_{t∈tasks} rqr(t, r)
6      q_avl ← rsc(a_i, r)
7      if q_avl < q_rqr then
8          A_req ← a_i's neighbors
9          q_req ← q_rqr - q_avl
10         utl_req ← compute request utility function for {q : 0 < q ≤ q_req}
11         τ^o_req ← τ_req
12         send request(r, q_req, utl_req, A_req, τ_req, τ^o_req) to A_req; add to sentReqs
```

Algorithm 7: Cascading requests process

Input : requests received by a_j: *receivedReqs*, resource types: R

Output: requests cascaded by a_j: *cascadedReqs*

```
1  cascadedReqs ← ∅
2  foreach r in R do
3      requestsForType ← requests in receivedReqs for r
4      sort requestsForType by efficiency in descending order
5      foreach request(r, q_req, utl_req, A_req, τ_req, τ^o_req) in requestsForType do
6          if currentTime < τ_req ∧ a_j has neighbor(s) ∉ A_req then
7              q_avl ← rsc(a_j, r)
8              A'_req ← A_req ∪ a_j's neighbor(s)
9              t'_req ← τ_req - timeoutReduction
10             if q_avl < q_req then
11                 q'_req ← q_req - q_avl
12                 utl'_req ← compute request utility function for {q : 0 < q ≤ q'_req}
13                 reserve q_avl of type r
14                 update rsc(a_j, r)
15                 send request(r, q'_req, utl', A'_req, t'_req, τ^o_req) to A'_req − A_req; add to
                     cascadedReqs
16             else if utl_req(q_avl) < offer cost of q_avl then
17                 send request(r, q_req, utl_req, A'_req, t'_req, τ^o_req) to A'_req − A_req; add to
                     cascadedReqs
```

6.3 Cascading Requests

When an agent a_j receives a request from one of its neighbors, it considers cascading it to its own neighbors, who are not given in the set of request recipients A_{req}. a_j may decide to cascade a request if it does not have enough resources to fully provide the requested quantity q_{req}, or if cost of offering is more that the benefit given in the utility function utl_{req}. In case a_j receives multiple requests from different agents for a resource type r, it sorts them in a greedy approach based on a heuristic measure defined as follows, and processes them in descending order.

Definition 9. (Request Efficiency)*: the ratio between the request utility,* $utl_{req}(q_{req})$*,*

and its requested quantity, q_{req}.

When cascading a request, a_j reserves an available amount of resources it can provide, and updates the request quantity and utility function, accordingly. In addition, it adds its own neighbors to A_{req}. Moreover, the cascaded request timeout must be sooner than the request timeout τ_{req}, in order for a_j to have enough time to receive potential offers from its neighbors and process them. Hence, τ_{req} is reduced by a parameter called *timeoutReduction*, which its value depends on the communication channel. The original request timeout τ^o_{req} is also included in the message and is used in the offering process described in the next section. Algorithm 7 formalizes the cascading requests process of a_j.

6.4 Offering

When an agent a_j receives a request from one of its neighbors, it considers offering to the request based on its available resources and cost of offering. The offering decision is more challenging when a_j receives requests from multiple neighbors for a resource type r with limited available quantity. Considering a_j is allowed to offer partially and distribute its available resources in multiple offers, each for a different request; finding the optimal offering strategy becomes computationally hard. Figure 6.3 illustrates an example of the problem of offering to multiple requests. In this scenario, a_1 receives requests from all of its neighbors a_2, a_3, and a_4, simultaneously. Suppose a_1 has 10 available resource items of type r; and req_2, req_3, and req_4 ask for seven, five, and three items of type r, respectively. What is the optimal offering strategy for a_1 (assuming it does not know about potential offers to a_2 from a_5, a_6, a_7, and a_8)?

In such a case that the number of available resource items is less than the total requested quantity by all the neighbors, then a_j optimizes the offering by selecting requests with maximum utility. Formally, suppose there are n requests and an available quantity of q_{avl}; then a_j selects an offering quantity q^k_{off} for each request $k \in \{1, ..., n\}$ with requested quantity q^k_{req} and utility function utl^k_{req}; in order to solve the integer programming (IP) problem P_{off} given below:

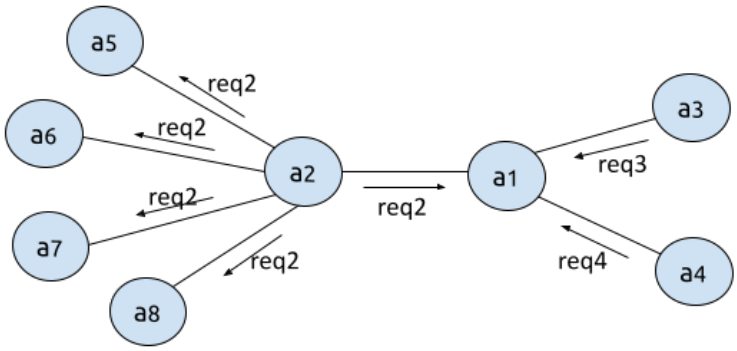

Figure 6.3: Offering to multiple requests

$$\max_{q^1_{off},\ldots,q^n_{off}} \quad \sum_{k=1}^{n} utl^k_{req}(q^k_{off}) \tag{6.1}$$

$$\text{s.t.} \quad 0 \leq q^k_{off} \leq q^k_{req} \tag{6.2}$$

$$\sum_{k=1}^{n} q^k_{off} \leq q_{avl} \tag{6.3}$$

Proposition 5. *For requests with non-linear utility functions, the complexity of finding the optimal solution for P_{off} is NP-hard.*

Proof. This can be converted to a 0-1 Knapsack problem: offering to each request with a quantity q_{off} from 1 to the requested quantity q_{req} is considered one Knapsack item with weight $= q^k_{off}$ and value $= utl^k_{req}(q^k_{off})$. The problem is to select items to maximize total utility, subject to the total weight $\leq$ available quantity and only one item can be selected from each requester.

□

Once the offer quantities are determined by solving P_{off}, a_j sends an offer to each requester by including the cost of offering, defined as a function: $cst_{off} : \mathbb{N} \longrightarrow \mathbb{N}$. The offer cost function is computed as utility losses (+ transfer costs) for partial quantities in $\{q : 0 < q \leq q_{off}\}$. It indicates cost of partial contributions by the offerer. In addition, an offer includes a timeout which is later than the original

request timeout in order for a_j to have enough time to receive potential confirmation from the original requester. Hence, the offer timeout is set by extending τ^o_{req} by a parameter called *timeoutExtension*, which its value depends on the communication channel. An offer message sent by a_j has the following format:

$$Offer(r, q_{off}, cst_{off}, itm_{off}, \tau_{off})$$

where q_{off} indicates maximum quantity offered for resource type r, $cst_{off} : \mathbb{N} \to \mathbb{N}$ is the offer cost function, itm_{off} is the set of offered resource items, and τ_{off} is the offer timeout; i.e. a_j waits for potential confirmation until τ_{off} and then the offer is expired. An offer message with $q_{off} = 0$ and $itm_{off} = \emptyset$ implies a rejection. The offering process is formalized in Algorithm 8.

Since solving P_{off} requires exponential running time, we introduce two polynomial-time approaches in order to deal with the problem of offering to multiple requests in large-scale settings. In the following sections, we present an approximate greedy approach by sorting the requests based on a heuristic measure; and a deep reinforcement learning approach in order to learn to better select requests over time based on the result of offering to the neighbors.

Algorithm 8: Offering process

Input : requests received by a_j: *receivedReqs*, resource types: R

Output: offers sent by a_j: *sentOffs*

```
sentOffs ← ∅
foreach r in R do
    requestsForType ← requests in receivedReqs for r
    offQuantities ← solve P_off for requestsForType using Algo. 9 or 11
    foreach (request, q_off) in offQuantities do
        cst_off ← compute offer cost function for {q : 0 < q ≤ q_off}
        itm_off ← reserve q_off resource items of type r
        τ_off ← τ^o_req + timeoutExtension
        send offer(r, q_off, cst_off, itm_off, τ_off) to the requester; add to sentOffs
```

6.4.1 Greedy Offering

In this approach, the received requests are sorted based on the request efficiency heuristic. Then, as long as resources are available, a_j considers offering to the request with the highest efficiency. Since a_j is allowed to make partial offers, it determines an available amount of resources it can provide q_{off}, not exceeding the requested quantity q_{req}. Moreover, it deliberates about offering to each request by comparing its offering cost, $cst_{off}(q_{off})$, against the request utility for the same quantity, $utl_{req}(q_{off})$. Then, a_j decides to offer only if $cst_{off}(q_{off}) < utl_{req}(q_{off})$. The greedy offering process is formalized in Algorithm 9.

Proposition 6. *For requests with linear utility functions, the greedy offering process in Algorithm 9 finds the optimal solution for P_{off}.*

Proof. Assuming linear functions for request utilities, P_{off} can be converted to a fractional Knapsack problem as follows. Each request is considered one Knapsack item with weight q^k_{req} and value $utl_{req}(q^k_{req})$. With linear utility functions, the benefit of partial contributions increases at a constant rate. Since Algorithm 9 allows offering a partial quantity (line 6), it results in the optimal solution.

□

Algorithm 9: Greedy offering process

Input : requests received by a_j for resource type r: *requestsForType*
Output: offered quantities for *requestsForType*: *offQuantities*

1 *offQuantities* ← (*request*, 0) for each *request* in *requestsForType*
2 sort *requestsForType* by efficiency in descending order
3 **foreach** *request*$(r, q_{req}, utl_{req}, A_{req}, \tau_{req}, \tau^o_{req})$ *in requestsForType* **do**
4 $q_{avl} \leftarrow rsc(a_j, r)$
5 **if** *currentTime* $< \tau_{req} \wedge q_{avl} > 0$ **then**
6 $q_{off} \leftarrow \min(q_{req}, q_{avl})$
7 $cost_{off} \leftarrow$ compute offer cost for q_{off}
8 **if** $cost_{off} < utl_{req}(q_{off})$ **then**
9 find and replace (*request*, q_{off}) in *offQuantities*
10 $rsc(a_j, r)$ $-= q_{off}$

6.4.2 Learning-based Offering

Using a deep reinforcement learning approach, each agent learns over time its own policy for deciding how to offer to its neighbors in the social network. The rationale is that in a random network, each agent may have a different set of neighbors. For example, in Figure 6.3, we can observe that a_2 has five neighbors, but a_3 and a_4 each has only one neighbor. Hence, it is more likely that req_2 receives an offer compared to req_3 and req_4. Then, a_1 may be able to learn over time that offering to req_3 or req_4 is a better action than offering to req_2; because it is more likely that a_3 or a_4 will confirm the offer.

Now in order to apply a reinforcement learning algorithm, first we define the Markov Decision Process (MDP) for deciding to offer when there are multiple requests as follows.

- *State*: a matrix that represents the requested resource type r, the available quantity q_{avl}, and set of net utilities (request utility - offer cost) for each possible offer quantity: $\{utl^k_{req}(q^k_{off}) - cst^k_{off}(q^k_{off})) : 0 < q^k_{off} \leq \min(q^k_{req}, q_{avl})\}$ for each neighbor's request. The order of agents in the state is fixed.

- *Action*: to offer to a request with a possible quantity q_{off}.

- *Transition Function*: the offerer agent sends an offer to the selected neighbor with a probability of 1, since situated in a deterministic environment.

- *Reward*: the net utility of confirmed quantity $q^k_{cnf} \leq q^k_{off}$: $utl^k_{req}(q^k_{cnf}) - cst^k_{off}(q^k_{cnf})$. The offering reward is determined once the offerer receives a confirmation from the selected neighbor (described in Section 4.6).

Similar to the centralized learning approach in Section 5.3, we can learn a policy over many episodes in order to solve the problem of offering to multiple requests, P_{off}. Hence, we apply a deep Q-learning algorithm with two neural networks: policy and target. The policy network is updated in every step, but the target network is only updated every C steps. Moreover, by incorporating experience replay (random minibatch updates) and double learning (decouple the selection from the evaluation of actions), we improve learning stability and reduce overestimation of the target.

Algorithm 10: Offering learning process

Input : received requests for n episodes: $requests_1$, $requests_2$, ..., $requests_n$
Output: trained target action-value function: $\mathtt{Q}'_{off}$

initialize replay memory D_{off} to capacity N_{off}
initialize policy action-value function $\mathtt{Q}_{off}$ with random weights θ_{off}
initialize target action-value function $\mathtt{Q}'_{off}$ with random weights $\theta'_{off} = \theta_{off}$
for $e = 1..n$ **do**
    $requests_e \leftarrow$ received requests in this episode
    $sentOffs \leftarrow \emptyset$
    foreach r *in* R **do**
        $requestsForType \leftarrow$ requests in $requests_e$ for r
        $\mathtt{t} \leftarrow 1$
        while $requestsForType \neq \emptyset$ **do**
            $\mathtt{s_t} \leftarrow$ generate the current state
            if $\mathtt{s_t}$ *is terminal* **then**
                break *while*
            ($request$, q_{off}) $\leftarrow$ with probability ϵ select a random action $\mathtt{a_t}$ otherwise select $\mathtt{a_t} = \arg\max_{\mathtt{a}} \mathtt{Q}_{off}(\mathtt{s_t}, \mathtt{a}; \theta_{off})$
            $\mathtt{s_{t+1}} \leftarrow$ generate the next state
            $cst_{off} \leftarrow$ compute offer cost function for $\{q : 0 < q \leq q_{off}\}$
            $itm_{off} \leftarrow$ reserve q_{off} resource items of type r
            $rsc(a_j, r)$ $-= q_{off}$
            $\tau_{off} \leftarrow \tau^o_{req} + timeoutExtension$
            send $offer(r, q_{off}, cst_{off}, itm_{off}, \tau_{off})$ to the requester; add to $sentOffs$
            remove $request$ from $requestsForType$
            $\mathtt{t}$ += 1
    while $sentOffs \neq \emptyset$ **do**
        check if any $offer$ in $sentOffs$ has been confirmed/rejected or expired
        $\mathtt{r_t} \leftarrow \begin{cases} utl_{req}(q_{cnf}) - cst_{off}(q_{cnf}) & \text{for any confirmed } offer \text{ in } sentOffs \\ 0 & \text{for any rejected or expired } offer \text{ in } sentOffs \end{cases}$
        store transition $(\mathtt{s_t}, \mathtt{a_t}, \mathtt{r_t}, \mathtt{s_{t+1}})$ in D_{off}
        sample random minibatch of transitions $(\mathtt{s_j}, \mathtt{a_j}, \mathtt{r_j}, \mathtt{s_{j+1}})$ from D_{off}
        $\mathtt{y_j} \leftarrow \begin{cases} \mathtt{r_j} & \text{for terminal } s_{j+1} \\ \mathtt{r_j} + \gamma \mathtt{Q}'_{off}(\mathtt{s_{j+1}}, \arg\max_{\mathtt{a}'} \mathtt{Q}_{off}(\mathtt{s_{j+1}}, \mathtt{a}'; \theta_{off}); \theta'_{off}) & \text{otherwise} \end{cases}$
        perform a gradient descent step on $[\mathtt{y_j} - \mathtt{Q}_{off}(\mathtt{s_j}, \mathtt{a_j}; \theta_{off})]^2$ with respect to θ_{off}
        every C_{off} steps update the target network $\mathtt{Q}'_{off} = \mathtt{Q}_{off}$
        remove $offer$ from $sentOffs$

Algorithm 10 formalizes the learning process. It uses the epsilon-greedy strategy, in which the exploration rate ϵ decays over time, balancing between exploration and exploitation. The *while* loop at line 22 indicates that there is a waiting period until reward(s) are determined for each sent offer. Hence, the policy updates are done asynchronously once a confirmation/rejection is received or the offer is expired. Once a policy is learned over many episodes, it can be fully exploited by setting $\epsilon = 0$ in one episode. Algorithm 11 formalizes this process called learning-based offering process in order to solve P_{off}. We will outline the neural network architecture and other hyperparameters used in the learning process in Chapter 7: Evaluation.

Proposition 7. *For an agent a_j with m neighbors, n tasks, l resource types, and maximum request quantity q_{req}^{max}, the overall time complexity of the offering learning process in each episode is $O(lm^2nq_{req}^{max} + lmNW)$, where N is the number of neurons, and W is the number of weights in the policy neural network.*

Proof. In each episode, generating the offering state takes $O(mnq_{req}^{max})$ because computing the offer cost takes $O(n)$ for each possible offer quantity $q_{off} \leq q_{req}^{max}$ and for each possible neighbor's request. Selecting an offering action requires a forward pass (prediction) in the target network, which generally takes $O(NW)$. All of the above is done for each resource type and each request. Then, the overall complexity of the *for* loop at line 7 is $O(lm^2nq_{req}^{max} + lmNW)$. The *while* loop at line 23 is repeated at most m times. In each iteration, the next state is generated which takes $O(mnq_{req}^{max})$; the sampling from the replay memory can be done in a constant time $O(1)$; the backward pass (backpropagation) has a complexity of $O(NW)$. The rest of operations can be done in a constant time $O(1)$. Then, the overall complexity of the *while* loop at line 23 is $O(m^2nq_{req}^{max} + mNW)$. Combining them, the overall complexity of the algorithm is $O(lm^2nq_{req}^{max} + lmNW + m^2nq_{req}^{max} + mNW)$. Hence, it simplifies to $O(lm^2nq_{req}^{max} + lmNW)$

□

Algorithm 11: Learning-based offering process

Input : requests received by a_j for resource type r: *requestsForType*, target action-value function: $\mathtt{Q}'_{off}$

Output: offered quantities for *requestsForType*: *offQuantities*

```
1 offQuantities ← (request, 0) for each request in requestsForType
2 while requestsForType ≠ ∅ do
3     s_t ← generate the current state
4     if s_t is terminal then
5         break while
6     (request, q_off) ← select a_t = arg max_a Q'_off(s_t, a; θ'_off)
7     find and replace (request, q_off) in offQuantities
8     rsc(a_j, r) −= q_off
9     remove request from requestsForType
```

6.5 Cascading Offers

For each cascaded request, the agent a_j waits for potential offers from its neighbors before sending an offer to the original requester. The rationale for waiting is to receive as many offers as possible. Eventually, a_j creates an offer, called cascaded offer, with a quantity up to the original requested quantity q_{req}. The offered items in the cascaded offer include all the items that have been reserved when cascading the original request, combined with any other item offered by a_j's neighbors. Similar to offering process in Section 6.4, the cascaded offer timeout is set by extending the original request timeout τ^o_{req} by *timeoutExtension*. The waiting period before cascading offers is determined by a parameter called *waitToCascadeOffers*, which its value depends on the application domain. Algorithm 12 formalizes the cascading offers process of a_j.

In case the total number of reserved items q_{res} plus all the offered items by the neighbors is greater than q_{req}, then a_j optimizes the offering cost by selecting offered items with minimum total cost. Formally, suppose there are n offers for a cascaded request with quantity $q'_{req} = q_{req} - q_{res}$; then a_i selects a quantity q'^k_{off} from each offer $k \in \{1, ..., n\}$ with offered quantity q^k_{off} and cost function cst^k_{off}; in order to solve the integer programming (IP) problem P_{csf} given below:

$$\min_{q'^1_{off},\ldots,q'^n_{off}} \quad \sum_{k=1}^{n} cst^k_{off}(q'^k_{off}) \tag{6.4}$$

$$\text{s.t.} \quad 0 \leq q'^k_{off} \leq q^k_{off} \tag{6.5}$$

$$\sum_{k=1}^{n} q'^k_{off} \leq q'_{req} \tag{6.6}$$

Proposition 8. *For offers with non-linear cost functions, the complexity of finding the optimal solution for P_{csf} is NP-hard.*

Proof. This can be converted to a 0-1 Knapsack problem: selecting a quantity from each offer is considered one Knapsack item with weight $= q'^k_{off}$ and value $= cst_{off}(q'^k_{off})$. □

We can solve P_{csf} in a greedy approach by sorting the offers in an ascending order based on a heuristic measure defined as follows:

Definition 10. (Offer Economy)*: the ratio between the offer cost, $cst_{off}(q_{off})$, and its offered quantity, q_{off}.*

Then, we select items from the offer with the minimum economy until we have selected q'_{req} items. Algorithm 13 formalizes this greedy process when cascading offers.

Proposition 9. *For offers with linear cost functions, the greedy cascading offers process in Algorithm 13 finds the optimal solution for P_{csf}.*

Proof. Assuming linear functions for offer costs, P_{csf} can be converted to a fractional Knapsack problem as follows. Each offer is considered one Knapsack item with weight q^k_{off} and value $cst_{off}(q^k_{off})$.

□

Algorithm 12: Cascading offers process

Input : requests cascaded by a_j: *cascadedReqs*, offers received by a_j: *receivedOffs*

Output: offers cascaded by a_j: *cascadedOffs*

$cascadedOffs \leftarrow \emptyset$

foreach $request(r, q'_{req}, utl_{req}, A_{req}, \tau_{req}, \tau^o_{req})$ *in cascadedReqs* **do**

  if $0 < \tau_{req}$ - $currentTime < waitToCascadeOffers$ **then**

    $offersForReq \leftarrow$ offers in *receivedOffs* for *request*

    $offItems \leftarrow \emptyset$

    foreach $offer(r, q_{off}, cst_{off}, itm_{off}, \tau_{off})$ *in offersForReq* **do**

      add itm_{off} to *offItems*

    if $|offItems| > q'_{req}$ **then**

      $itm'_{off} \leftarrow$ select q'_{req} from *offItems* by solving P_{csf} using Algo. 13

    else

      $itm'_{off} \leftarrow offItems$

    $resItems \leftarrow$ resource items reserved for *request* in Algo. 7

    add *resItems* to itm'_{off}

    $q'_{off} \leftarrow |itm'_{off}|$

    $cst'_{off} \leftarrow$ compute offer cost function for $\{q : 0 < q \leq q'_{off}\}$

    $t'_{off} \leftarrow \tau^o_{req} + timeoutExtension$

    send $offer(r, q'_{off}, cst'_{off}, itm'_{off}, \tau'_{off})$ to the requester; add to *cascadedOffs*

Algorithm 13: Greedy cascading offers process

Input : offers received for cascaded $request(q'_{req})$: *offersForReq*

Output: cascaded offered items: itm'_{off}

$itm'_{off} \leftarrow \emptyset$
sort *offersForReq* by economy in ascending order
$q_{total} \leftarrow 0$
while $q_{total} < q'_{req} \wedge offersForReq \neq \emptyset$ **do**
    (*offer*, q_{off}) $\leftarrow$ select next offer in *offersForReq*
    $q'_{off} \leftarrow \min(q_{off}, q'_{req} - q_{total})$
    add q'_{off} items in *offer* to itm'_{off}
    q_{total} += q'_{off}
    remove *offer* from *offersForReq*

6.6 Confirming

The requester a_i waits for potential offers to its request for a resource type r. Similar to waiting before cascading offers in Section 6.5, the rationale here is to receive as many offers as possible. Hence, a_i may receive multiple offers from multiple providers. In that case, it selects a combination of offers that maximizes the difference between utility of the request and total cost of all the selected offers. Note that a_i is allowed to take partial amounts of offered resources in multiple offers up to the requested amount. Figure 6.4 illustrates an example of the problem of confirming when there are multiple offers. In this scenario, a_2 has received offers from three of its neighbors a_1, a_5, and a_7, before its request's timeout. Suppose *off12*, *off52*, and *off72*, include 5, 4, and 3 offered items, respectively; while req_2 asked for seven items only. What is the optimal confirming strategy for a_2 (assuming it does not know about confirming decisions made by other agents)?

In case the number of requested resource items is less than the total offered quantity by all the neighbors, then a_i optimizes the confirming process by selecting offers with minimum cost. Formally, suppose there are n offers for a request with requested quantity q_{req} and utility function utl_{req}; then a_i selects a quantity q^k_{cnf} from each offer $k \in \{1, ..., n\}$ with offered quantity q^k_{off} and cost function cst^k_{off}; in order to solve the integer programming (IP) problem P_{cnf} given below:

$$\max_{q^1_{cnf},\ldots,q^n_{cnf}} \quad utl_{req}(\sum_{k=1}^{n} q^k_{cnf}) - \sum_{k=1}^{n} cst^k_{off}(q^k_{cnf}) \tag{6.7}$$

$$\text{s.t.} \quad 0 \leq q^k_{cnf} \leq q^k_{off} \tag{6.8}$$

$$\sum_{k=1}^{n} q^k_{cnf} \leq q_{req} \tag{6.9}$$

Proposition 10. *For requests and offers with non-linear utility and cost functions, the complexity of finding the optimal solution for P_{cnf} is NP-hard.*

Proof. This can be converted to a 0-1 Knapsack problem: selecting a quantity from each offer is considered one Knapsack item with weight $= q^k_{cnf}$ and value $= utl_{req}(q_{total} + q^k_{cnf}) - utl_{req}(q_{total}) - cst^k_{off}(q^k_{cnf})$, where q_{total} is the total confirmed quantity before adding this item.

□

The waiting period before confirming offers is determined by a parameter called *waitToConfirmOffers*, which its value depends on the application domain. A confirmation message sent by a_i for each selected offer has the following format:

$$Confirm(r, q_{cnf}, itm_{cnf})$$

where q_{cnf} indicates partial amount confirmed for resource type r, and itm_{cnf} is the set of confirmed resource items. A confirmation message with $q_{cnf} = 0$ and

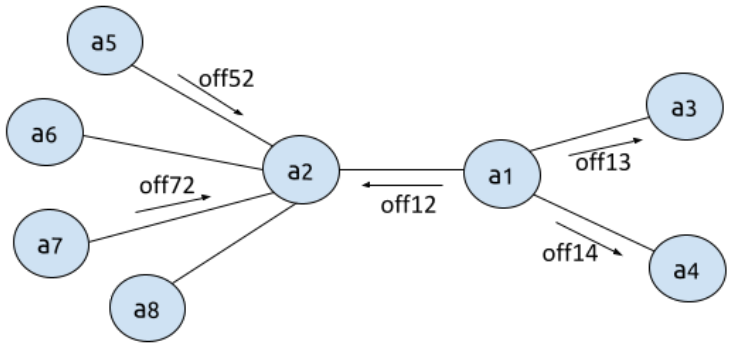

Figure 6.4: Confirming multiple offers

$itm_{cnf} = \emptyset$ implies a rejection. We formalize the confirming process in Algorithm 14. In the following, we present two polynomial-time approaches to solve P_{cnf}. A greedy approach by sorting the offers based on their costs; and a deep reinforcement learning approach in order to learn to better select offers over time based on the result of confirming.

Algorithm 14: Confirming process

Input : requests sent by a_i: *sentReqs*, offers received by a_i: *receivedOffs*
Output: confirmations sent by a_i: *sentConfs*

```
sentConfs ← ∅
foreach request(r, q_req, utl_req, A_req, τ_req, τ^o_req) in sentReqs do
    if 0 < τ_req - currentTime < waitToConfirmOffers then
        offersForReq ← offers in receivedOffs for request
        if offersForReq ≠ ∅ then
            confQuantities ← solve P_cnf for offersForReq using Algo. 15 or 17
            foreach (offer, q_cnf) in confQuantities do
                itm_cnf ← select q_cnf from itm_off in offer
                send confirm(r, q_cnf, itm_cnf) to the offerer; add to sentConfs
```

6.6.1 Greedy Confirming

Similar to solving P_{csf} in Section 6.5, we can solve P_{cnf} in a greedy approach by sorting the offers by the offer economy heuristic in an ascending order, and select items from the offer with the minimum economy until we have selected q_{req} items. Algorithm 15 formalizes the greedy confirming process.

Algorithm 15: Greedy confirming process

Input : offers received for $request(q_{req})$: *offersForReq*

Output: confirmed quantities for *offersForReq*: *confQuantities*

```
1 confQuantities ← (offer, 0) for each offer in offersForReq
2 sort offersForReq by economy in ascending order
3 q_total ← 0
4 while q_total < q_req ∧ offersForReq ≠ ∅ do
5     (offer, q_off) ← select next offer in offersForReq
6     q_cnf ← min(q_off, q_req − q_total)
7     find and replace (offer, q_cnf) in confQuantities
8     q_total += q_cnf
9     remove offer from offersForReq
```

Proposition 11. *For requests and offers with linear utility and cost functions, the greedy confirming process in Algorithm 15 finds the optimal solution for P_{cnf}.*

Proof. Assuming linear functions for request utilities and offer costs, P_{cnf} can be converted to a fractional Knapsack problem as follows. Each offer is considered one Knapsack item with weight q^k_{off} and value $utl_{req}(q^k_{off}) - cst_{off}(q^k_{off})$.

□

6.6.2 Learning-based Confirming

Using a deep reinforcement learning approach, each agent learns over time its own policy for deciding how to confirm offers from its neighbors in the social network. The rationale is the same as in the learning-based offering process. In a random network, each agent is situated in a different neighborhood, hence it learns to confirm offers based on interactions with its neighbors. For example, in Figure 6.4, we can observe that a_2 has received offers from a_1, a_5, and a_7. In deciding what to confirm among these offers, a_2 might learn over time that confirming the offers from a_5 or a_7 is a better action than confirming the offer from a_1. This is because a_1 has also offered to a_3 and a_4; hence, it is more likely that its offer to a_2 is more costly.

Similar to the learning-based offering in Section 6.4.2, first we define the Markov Decision Process (MDP) for deciding to confirm when there are multiple offers as follows.

- *State*: a matrix that represents the requested resource type r, the request utility function values utl_{req}, and set of offer costs for each possible confirmation quantity: $\{cst^k_{off}(q^k_{cnf}) : 0 < q^k_{cnf} \leq \min(q^k_{off}, q_{req} - q_{total})\}$ for each neighbor's offer, where q_{total} is the total confirmed quantity so far. The order of agents in the state is fixed.

- *Action*: to confirm an offer with a possible quantity q^k_{cnf}.

- *Transition Function*: the requester agent sends a confirmation to the selected neighbor with a probability of 1, since situated in a deterministic environment.

- *Reward*: the net utility of confirmed quantity q^k_{cnf}: $utl_{req}(q_{total}+q^k_{cnf})-utl_{req}(q_{total})$ $-cst^k_{off}(q^k_{cnf})$, where q_{total} is the total confirmed quantity before this action. The confirming reward is determined immediately.

Similar to the offering learning process, we can learn a policy over many episodes in order to solve the problem of confirming multiple offers. Hence, we apply a deep Q-learning algorithm with two neural networks: policy and target. The policy network is updated in every step, but the target network is only updated every C steps. Moreover, by incorporating experience replay (random minibatch updates) and double learning (decouple the selection from the evaluation of actions), we improve learning stability and reduce overestimation of the target. Algorithm 16 formalizes the learning process. It uses the epsilon-greedy strategy, in which the exploration rate ϵ decays over time, balancing between exploration and exploitation.

Algorithm 16: Confirming learning process

Input : received offers for n episodes: $offers_1$, $offers_2$, ..., $offers_n$

Output: trained target action-value function: $\mathtt{Q}'_{cnf}$

initialize replay memory D_{cnf} to capacity N_{cnf}
initialize policy action-value function $\mathtt{Q}_{cnf}$ with random weights θ_{cnf}
initialize target action-value function $\mathtt{Q}'_{cnf}$ with random weights $\theta'_{cnf} = \theta_{cnf}$
for $e = 1..n$ **do**
    $offers_e \leftarrow$ received offers in this episode
    foreach r *in* R **do**
        $offersForReq \leftarrow$ offers in $offers_e$ for sent request(r, q_{req})
        $q_{total} \leftarrow 0$
        $\mathtt{t} \leftarrow 1$
        while $offersForReq \neq \emptyset$ **do**
            $\mathtt{s_t} \leftarrow$ generate the current state
            if $\mathtt{s_t}$ *is terminal* **then**
                break *while*
            (*offer*, q_{cnf}) $\leftarrow$ with probability ϵ select a random action $\mathtt{a_t}$ otherwise select $\mathtt{a_t} = \arg\max_{\mathtt{a}} \mathtt{Q}_{cnf}(\mathtt{s_t}, \mathtt{a}; \theta_{cnf})$
            $\mathtt{r_t} \leftarrow utl_{req}(q_{total} + q_{cnf}) - utl_{req}(q_{total}) - cst_{off}(q_{cnf})$
            q_{total} += q_{cnf}
            $\mathtt{s_{t+1}} \leftarrow$ generate the next state
            store transition $(\mathtt{s_t}, \mathtt{a_t}, \mathtt{r_t}, \mathtt{s_{t+1}})$ in D_{cnf}
            sample random minibatch of transitions $(\mathtt{s_j}, \mathtt{a_j}, \mathtt{r_j}, \mathtt{s_{j+1}})$ from D_{cnf}
            $\mathtt{y_j} \leftarrow$

$$\begin{cases} \mathtt{r_j} & \text{for terminal } \mathtt{s_{j+1}} \\ \mathtt{r_j} + \gamma \mathtt{Q}'_{cnf}(\mathtt{s_{j+1}}, \arg\max_{\mathtt{a}'} \mathtt{Q}_{cnf}(\mathtt{s_{j+1}}, \mathtt{a}'; \theta_{cnf}); \theta'_{cnf}) & \text{otherwise} \end{cases}$$

            perform a gradient descent step on $[\mathtt{y_j} - \mathtt{Q}_{cnf}(\mathtt{s_j}, \mathtt{a_j}; \theta_{cnf})]^2$ with respect to θ_{cnf}
            every C_{cnf} steps update the target network $Q'_{cnf} = Q_{cnf}$
            remove *offer* from *offersForReq*
            $\mathtt{t}$ += 1

In contrast with the offering learning, there is no waiting period to determine the confirming rewards. Hence, the policy updates are done synchronously. Once a

policy is learned over many episodes, it can be fully exploited by setting $\epsilon = 0$ in one episode. Algorithm 17 formalizes this process called learning-based confirming process in order to solve the problem of confirming multiple offers, P_{cnf}. We will outline the neural network architecture and other hyperparameters used in the learning process in Chapter 7: Evaluation.

Proposition 12. *For an agent a_i with m neighbors, l resource types, and maximum request quantity q_{req}^{max}, the overall time complexity of the confirming learning process in each episode is $O(lm^2 q_{req}^{max} + lmNW)$, where N is the number of neurons, and W is the number of weights in the policy neural network.*

Proof. In each episode, the *while* loop at line 10 is repeated at most m times. In each iteration, generating the confirming state takes $O(mq_{req}^{max})$ for each possible offer quantity $q_{off} \leq q_{req}^{max}$ and for each possible neighbor's offer. Selecting an offering action requires a forward pass (prediction) in the target network, which generally takes $O(NW)$. The backward pass (backpropagation) has a similar complexity of $O(NW)$. The sampling from the replay memory can be done in a constant time $O(1)$. The rest of operations can be done in a constant time $O(1)$. All of the above is done for each resource type. Combining them, the overall complexity of the algorithm is $O(lm^2 q_{req}^{max} + lmNW)$.

□

Algorithm 17: Learning-based confirming process

Input : offers received by a_i for *request*(q_{req}): *offersForReq*, target action-value function: $\mathtt{Q}'_{cnf}$

Output: confirmed quantities for *offersForReq*: *confQuantities*

1 *confQuantities* ← (*offer*, 0) for each *offer* in *offersForReq*
2 **while** *offersForReq* ≠ ∅ **do**
3 $\mathtt{s_t}$ ← generate the current state
4 **if** $\mathtt{s_t}$ *is terminal* **then**
5 break *while*
6 (*offer*, q_{cnf}) ← select $\mathtt{a_t} = \arg\max_{\mathtt{a}} \mathtt{Q}'_{cnf}(\mathtt{s_t}, \mathtt{a}; \theta'_{cnf})$
7 find and replace (*offer*, q_{cnf}) in *confQuantities*
8 remove *offer* from *offersForReq*

6.7 Cascading Confirmations

When an offerer agent a_j receives a confirmation for its offer, it checks if the offer includes partial offers from its neighbors; i.e. it is a cascaded offer combining multiple offers. In that case, a_j cascades partial confirmations to the corresponding offerers. Since a confirmation message contains item ids, a_j is able to distinguish its local offered items from the items offered by its neighbors. Hence, a_j is able to restore any of its local offered items that has not been confirmed. Note that this process is repeated by the corresponding neighbors as long as there are partial offers along the network. We formalize the cascading confirmations process in Algorithm 18.

Algorithm 18: Cascading confirmations process

Input : confirmations received by a_j: *receivedConfs*, offers sent by a_j: *sentOffs*

Output: confirmations cascaded by a_j: *cascadedConfs*

1 $cascadedConfs \leftarrow \emptyset$
2 **foreach** $receivedConf(r, q_{cnf}, itm_{cnf})$ *in receivedConfs* **do**
3 $\quad sentOff\,(r, q_{off}, cst_{off}, itm_{off}, \tau_{off}) \leftarrow$ offer in *sentOffs* for *receivedConf*
4 $\quad neighborsOffs \leftarrow$ offers from neighbors included in *sentOff*
5 $\quad$ **foreach** $offer(r, q'_{off}, cst'_{off}, itm'_{off}, \tau'_{off})$ *in neighborsOffers* **do**
6 $\quad\quad itm'_{cnf} \leftarrow itm'_{off} \cap itm_{cnf}$
7 $\quad\quad q'_{cnf} \leftarrow |itm'_{cnf}|$
8 $\quad\quad$ send $confirm(r, q'_{cnf}, itm'_{cnf})$ to the offerer; add to *cascadedConfs*
9 $\quad$ **if** $q_{cnf} < q_{off}$ **then**
10 $\quad\quad$ restore local reserved items in $itm_{off} - itm_{cnf}$

Chapter 7

Evaluation

In this chapter, we evaluate the different approaches presented in this thesis in order to solve the social resource allocation problem (SRAP). First, we describe our simulation models, and different types of social networks that we use in our experiments. Then, we present our simulation results for the centralized and the decentralized approaches using greedy and learning-based solutions. Moreover, we measure the quality of our solutions by comparing their results to the optimal solution. We conclude the chapter by discussing our findings and presenting the insights derived from our experimental observations.

7.1 Simulation Models

We have developed an abstract simulator in Java using the JADE framework [6], which allows us to compare the performance of the different approaches in terms of efficiency and scalability. JADE complies with the FIPA (Foundation for Intelligent Physical Agents) specifications [2], ensuring interoperability and compliance with standardization in agent communication and behavior.

JADE is designed to support developing multi-agent systems and complex interaction protocols ranging from small to large-scale applications. Using JADE, each agent operates in its own thread, allowing for concurrent processing of agent behaviors. The agents communicate asynchronously using a message-passing mechanism, which allows processing of messages in a non-blocking manner. Hence, agents are able to perform multiple behaviors in parallel while waiting to receive messages.

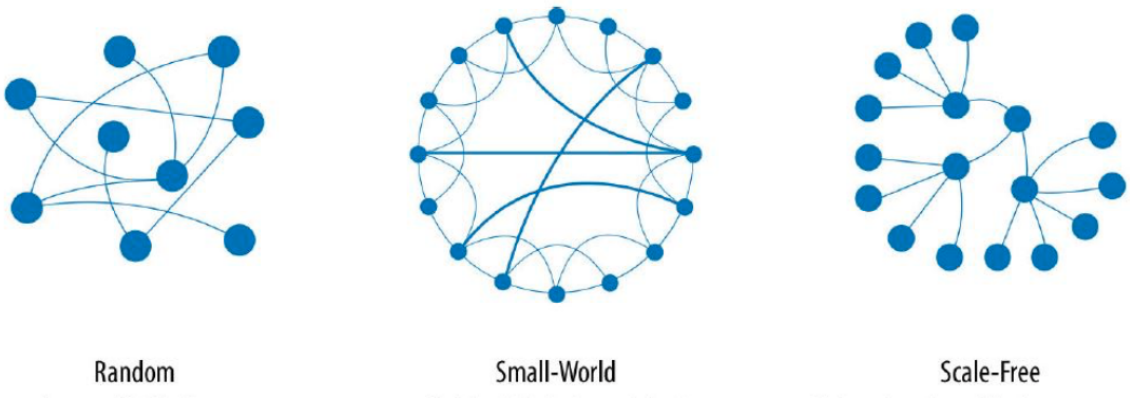

Figure 7.1: Different types of social networks

7.2 Social Networks

We perform experiments using three different types of social networks described as follows. We use the JGraphT Java library [34] in order to generate these networks. Figure 7.1 adopted from [26] shows one example for each type of these networks.

- *Small-world networks*: are networks in which most neighbors of an agent are also connected. We generate small-world graphs based on the Watts-Strogatz model [55] with a fixed rewiring probability of 0.05.
- *Scale-free networks*: are networks in which a small amount of nodes have many connections, while many nodes have a small number of connections [5]
- *Random networks*: we generate by first connecting each agent to another agent, and then randomly add more connections until an average degree has been reached [13, 14]

The small-world, scale-free, and random networks have distinct structural characteristics. Each network type represents a class of common network topologies observed in real-world environments. The small-world networks exhibit strong local clustering, i.e. larger neighborhoods which allow agents to interact more with their direct neighbors. They also provide a short chain of social connections between any two agents. The rewiring probability determines how much randomness is introduced into the initial regular lattice structure. This is the probability that each edge will be rewired to a randomly chosen node. Setting a low value (5%) of rewiring probability preserves

high local clustering, while introducing a small number of random connections, which reduces the average path length between agents. Real-world examples of small-world networks can be seen in hospital networks and electric power grids. The scale-free networks include central nodes or hubs that dominate the connectivity and influence within the network. They model many real-world examples, such as data center networks and transportation networks. The random networks include arbitrarily and uniformly random connections, without any inherent clustering or predefined rule. Real-world examples can be seen in peer-to-peer networks, and networks of IoT devices, given that the nodes might be randomly activated in the network at any given time. Therefore, by using these different social network types in our simulation experiments, we can perform a comprehensive assessment of our presented approaches to solve the SRAP. This allows us to draw conclusions about efficiency and scalability of our approaches, and to validate the applicability of our research findings in social network environments which resemble different real-world scenarios.

7.3 ILP Solver

In this thesis, we formulate the SRAP as an integer linear programming (ILP) problem P_{opt} in order to find the optimal solution (OPT) as a benchmark for assessing our greedy and learning-based approaches. The Gurobi optimizer [24] is employed for this purpose, as it is widely recognized for its efficiency in solving ILP problems. However, due to the exponential time complexity of solving this ILP, which scales with the number of agents, tasks, and resource types, finding the optimal solution (OPT) is computationally feasible only for small to moderate-sized problem instances. As the number of variables grows, the ILP solver requires significantly more computational resources, often making it impractical for larger networks with high agent-task-resource combinations. Consequently, for larger parameter values and also in dynamic environments, we exclude the OPT results from our experimental analysis and only rely on our greedy and learning-based methods for scalable and efficient solutions. These proposed methods, while not guaranteed to find the global optimum, offer practical performance and are designed to adapt to the changes in dynamic environments where computational efficiency is crucial.

7.4 Learning Models

In our experiments with the learning-based approach, we create custom neural networks for both centralized and decentralized learning algorithms (Alg. 3, Alg. 10, Alg. 16). We use the Deeplearning4j Java library [1] in order to implement these deep learning models used in our reinforcement learning algorithms.

7.4.1 Neural Network Architecture

In the centralized approach, as described in Section 5.3, the master state includes all the task utilities, task requirements for each resource type, and all the agents available resources for each type. For instance, with ten agents, two resource types, and maximum four tasks per agent, the master state size is calculated as follows.

$$masterStateSize = 10 * 4 + 10 * 4 * 2 + 10 * 2 = 140$$

In this case, we use the following configuration for the master network. The network architecture is illustrated in Fig. 7.2.

- *Layers and Neurons*: The input layer size is 140 and the output layer (equivalent to the master actions) size is equal to the number of agents, 10. There are three hidden layers of 100, 50, and 25 neurons. This hierarchical structure allows the network to learn intricate relationships in the data, which is essential for handling complex decision-making scenarios.

- *Activation Functions*: ReLU (rectified linear unit) activation function ($f(x) = max(0, x)$) is used for the input and hidden layers. ReLU allows deeper networks to learn effectively. It is computationally efficient and enables the network to model complex functions by introducing non-linearity. Identity Activation Function ($f(x) = x$) is used for the output layer. Identity is suitable for unbounded output values in a continuous space. In this case, the actions can be represented as real-valued numbers [22].

- *Weights Initialization*: Xavier technique is used for initializing network weights. It is particularly useful when ReLU activation function is used since it prevents neuron saturation issue at the beginning of the learning process [22].

- *Optimization Algorithm*: Stochastic Gradient Descent (SGD) is used to update the network parameters iteratively. SGD is suitable for large-scale optimization problems since it is based on the gradient of the loss function with respect to each parameter [22].

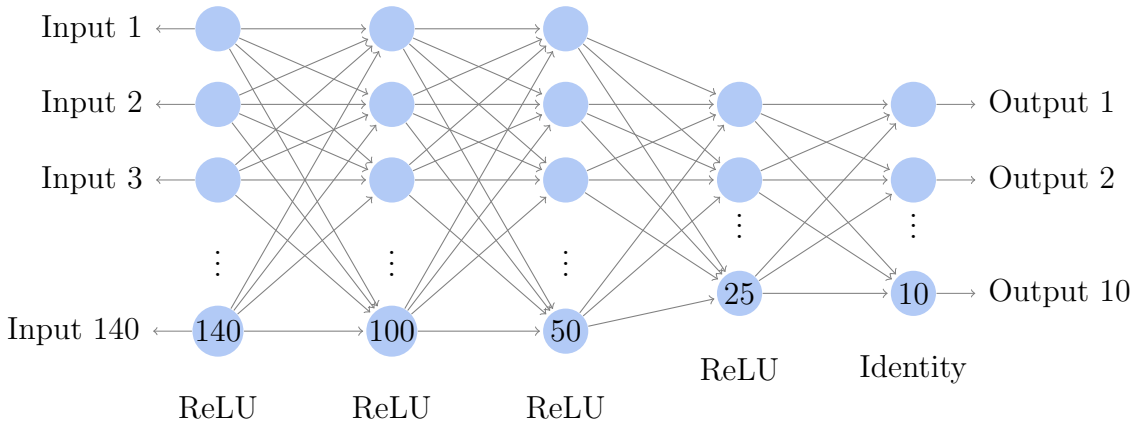

Figure 7.2: Deep neural network architecture for the master network

In the decentralized protocol, as described in Section 6.4.2, the offering state includes the requested resource type value, the available quantity, and the net utilities (request utility - offer cost) for each possible offer quantity for each neighbor's request. For instance, for a potential offerer with four neighbors and maximum request quantity of 10, the offering state size is calculated as follows.

$$offeringStateSize = 1 + 1 + 4 * 10 = 42$$

In this case, the offering network input layer size is 42 and the output layer (equivalent to the offering actions) size is equal to the number of neighbors multiplied by the maximum request quantity, $4 * 10 = 40$. Hence, we include two hidden layers of 70 and 35 neurons into the network architecture with the same configuration for the activation functions, weights initialization, and optimization algorithm as outlined for the master network. The offering network architecture is illustrated in Fig. 7.3.

Similarly, as described in Section 6.6.2, the confirming state includes the requested resource type value, the request utility function values, and offer costs for each possible confirmation quantity for each neighbor's offer. For instance, for a requester with four

neighbors and maximum request quantity of 10, the confirming state size is calculated as follows.

$$confirmingStateSize = 1 + 10 + 4 * 10 = 51$$

In this case, the confirming network input layer size is 51 and the output layer (equivalent to the confirming actions) size is equal to the number of neighbors multiplied by the maximum request quantity, $4 * 10 = 40$. Again, we include two hidden layers of 70 and 35 neurons into the network architecture with the same configuration for the activation functions, weights initialization, and optimization algorithm as outlined for the master network. The confirming network architecture is similar to the one for the offering network illustrated in Fig. 7.3.

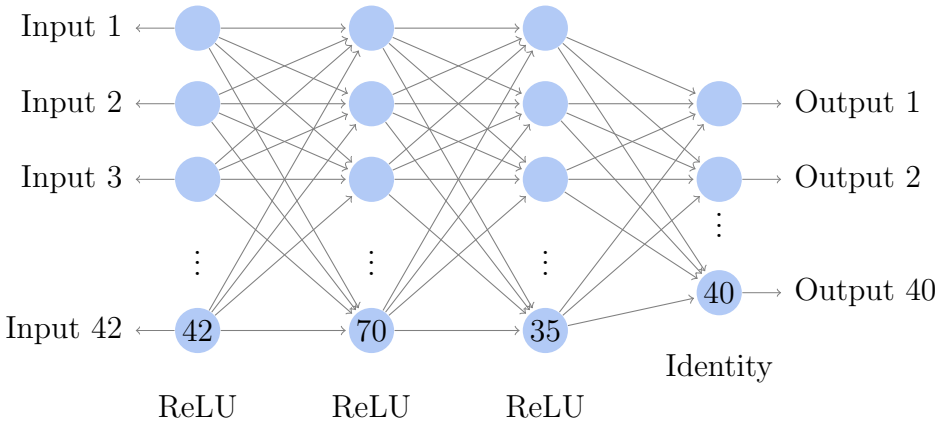

Figure 7.3: Deep neural network architecture for the offering network

7.4.2 Hyperparameters for Learning Processes

In deep reinforcement learning, there are numerous hyperparameters that require proper setting and fine-tuning [30]. These hyperparameters can significantly affect the quality of the learning processes. In our simulation experiments, the following hyperparameters are empirically set for both the centralized and decentralized approaches.

We employ the epsilon-greedy strategy by balancing between exploration and exploitation. In our learning processes, with probability of $1 - \epsilon$, the best action is selected based on the current policy values, and with probability of ϵ, a random

action is selected. We run the learning processes for 20000 episodes. Initially, the exploration rate ϵ is set to one. Over time, ϵ is decayed with a rate of 0.99982 to a minimum value of 0.1 in order to further exploit the learned policy. Then, in our evaluation runs, ϵ is set to *zero* in order to fully exploit the learned policy. We run the evaluation processes for one episode, when the algorithm is deterministic; or for 1000 episodes, when there is a randomness in the environment or there is a stochastic agent behavior, such as cascading requests in the decentralized protocol.

In addition, the discount factor, $0 \leq \gamma \leq 1$, and the learning rate, $0 \leq \alpha \leq 1$, used in computing the target values (Equation 5.7) are set as follows. γ determining the present value of future rewards is set to 0.95. α determining to what extent new information overrides old information is initially set to 0.001. We deploy a learning rate decay technique called *ReduceLROnPlateau*. Using this scheduling technique [22], α is reduced only when a specified metric does not improve for a number of episodes greater than a value called *patience*. Hence, α is kept the same as long as the metric is improving. We set the *patience* value to 500, and the reduction rate to 0.75. Hence, we reduce α with a factor of 0.75 when the metric does not improve for more that 500 episodes. This is done by monitoring the loss function (Equation 5.5); i.e., a lower score indicates better performance. We reduce α to a minimum value of 0.0000001. By using *ReduceLROnPlateau*, we can achieve better performance. In particular, when the neural network model reaches a plateau, *ReduceLROnPlateau* reduces the learning rate α, which is used to adjust the model parameters (Equation 5.7). This scheduling technique is useful when the model has largely converged to the optimal weights, and yet further smaller steps are required to fine-tune the network weights. Furthermore, in our learning processes, the experience replay buffer size is 100000, with a batch size of 32; and the target network update frequency is 200.

It is important to note that neural network design and setting hyperparameters for deep reinforcement learning is often an empirical process [22]. It involves experimentation with different network architectures and various configurations. Hence, each specific problem may require a different network design and configuration for the number of layers, number of neurons, activation functions, optimization algorithm, learning rate, discount factor, etc. In this thesis, our selected architectures and configurations are the result of a series of experiments in order to fine-tune the learning models and hyperparameters. While the selected settings may not be the optimal ones, they produce satisfactory results for our research objectives.

7.5 Experiments

In our simulation experiments, we evaluate the different approaches we presented in chapters 5 and 6 to solve the SRAP using scale-free, small-world, and random networks. For simplicity, we assign a weight of 1 to all the edges in the social networks. In each experiment, we observe and compare the social welfare results over a different parameter space. We run the experiments for one or many episodes depending on the deterministic or stochastic behavior of the approach; and the static or dynamic setting for the environment. In the decentralized protocol, in every episode, each agent: *i)* receives tasks and resources, *ii)* negotiates and allocates resources, *iii)* performs tasks given available resources. In the centralized approach, the master agent receives the same tasks and resources for all the individual agents in each episode in order to solve the same SRAP. We compare the social welfare result computed by the following approaches:

- Decentralized protocol with greedy offering and confirming (DEC-GRD)
- Decentralized protocol with learning-based offering and confirming (DEC-RL)
- Centralized greedy approach (CEN-GRD)
- Centralized learning approach (CEN-RL)
- Centralized optimal solution (OPT)

7.5.1 Resource Ratio

In our first experiment, we observe how different approaches behave in different social networks when we vary the available resource quantity. There are eight agents $(A1, A2, ..., A8)$. Each agent has four tasks to do with utilities of 4, 9, 16 and 25. There are two resource types. Each task requires two items per resource type. Four agents $(A1, A2, A3, A4)$ each have only one available resource item; hence, they act as requesters. The other four $(A5, A6, A7, A8)$ act as potential providers by having more available resources varying from 7 to 15. The small-world and random networks have an average degree of two; i.e. each agent has on average two direct neighbors. Since the social networks are randomly generated, our results are the average over 10 random instances for each social network type.

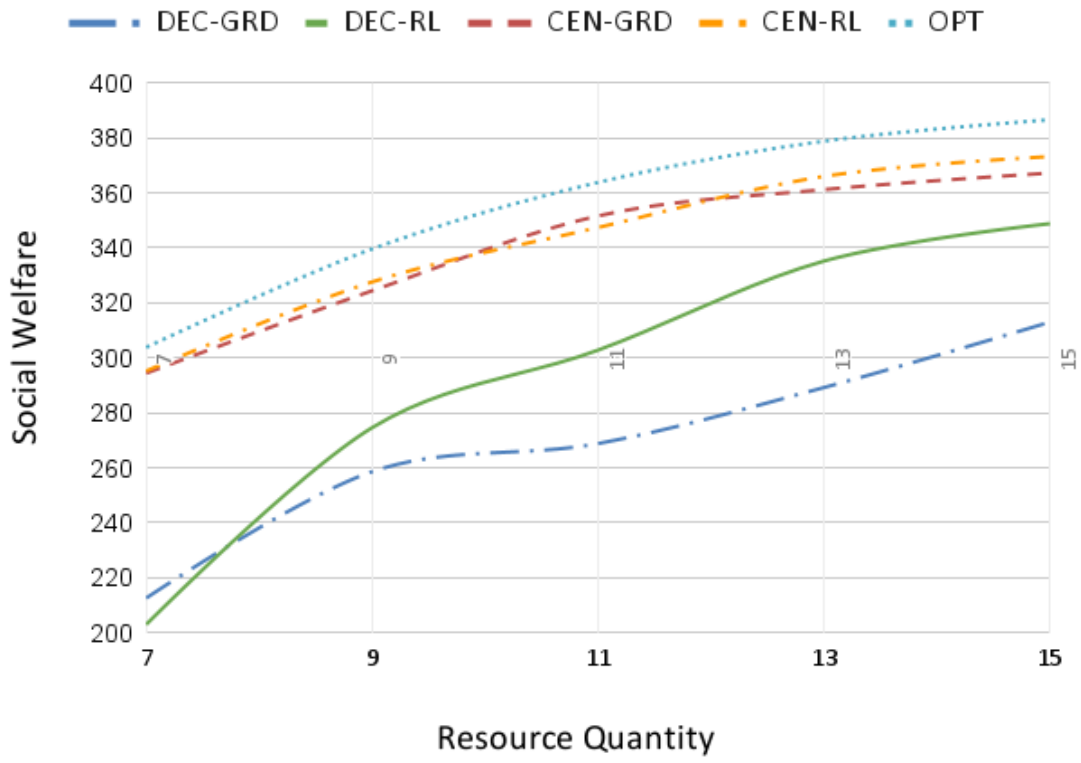

Figure 7.4: Results for varying available resource quantity in scale-free networks

Figure 7.4 illustrates our simulation results with scale-free networks. We observe that the centralized approaches, CEN-GRD and CEN-RL outperform the decentralized approaches, and they achieve social welfare results close to OPT by maintaining a consistent margin as the available resource quantity increases. Moreover, CEN-RL performs slightly better than CEN-GRD for higher values of resource quantity. On the other hand, the decentralized approaches, DEC-GRD and DEC-RL underperform for lower number of resource quantity, and perform better as more resources are available. We also observe that DEC-RL outperforms DEC-GRD increasingly as there are more available resources. In particular, we observe that DEC-RL achieves social welfare close to the centralized results at the higher end of parameter space, when the available resource quantity is high.

Figure 7.5 illustrates our simulation results with small-world networks. We observe similar results for the centralized approaches, CEN-GRD and CEN-RL. Notably, CEN-RL achieves the optimal result at the point where the resource quantity is

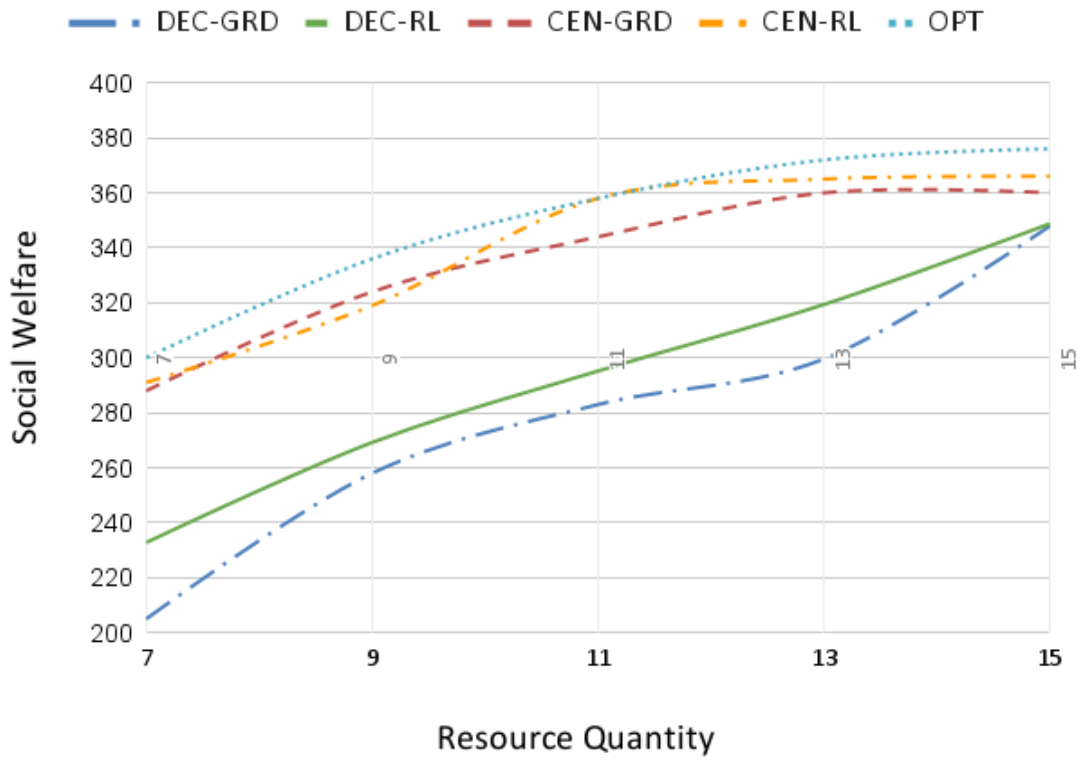

Figure 7.5: Results for varying available resource quantity in small-world networks

11. The decentralized approaches perform more closely over the parameter space, while DEC-RL always outperforming DEC-GRD. We also observe that the difference between centralized and decentralized results decreases as there are more available resources.

Figure 7.6 illustrates our simulation results with random networks. Similar to the results for other social network types, we observe that the centralized approaches, CEN-GRD and CEN-RL achieve social welfare results close to the optimal solution, OPT. Again, DEC-RL performs better than DEC-GRD by a little margin over all the parameter space. We also observe that while the centralized approaches decline at the higher end of parameter space, the decentralized approaches perform better and achieve results close to the centralized ones.

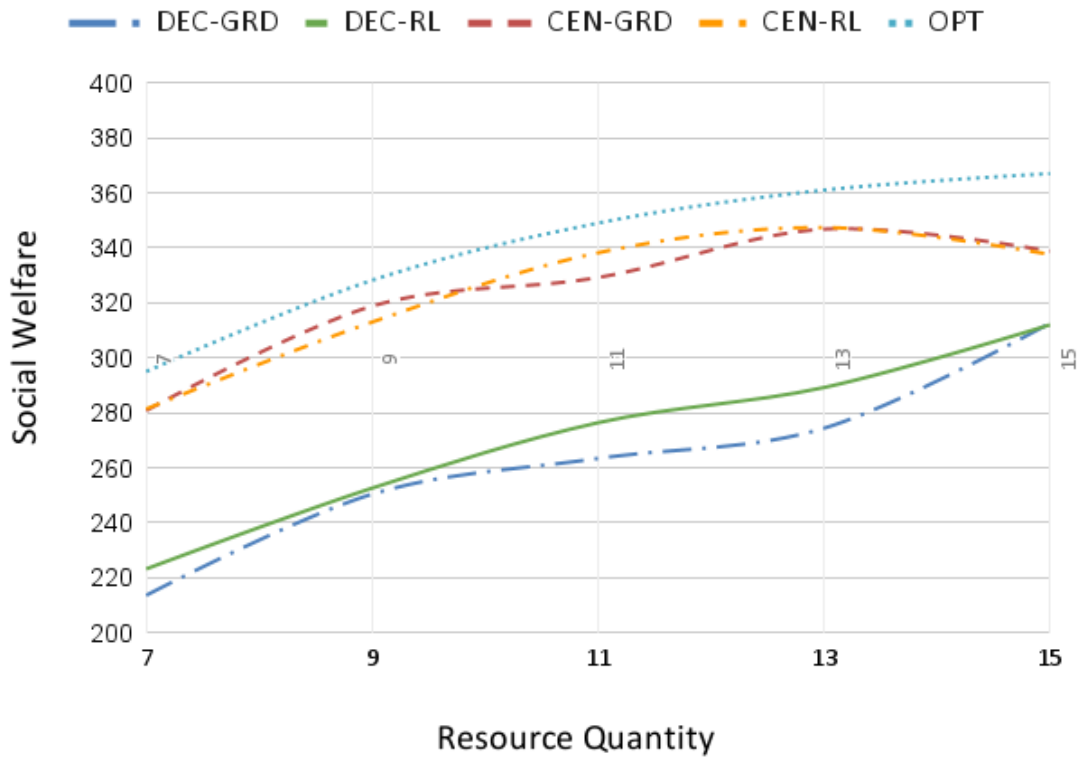

Figure 7.6: Results for varying available resource quantity in random networks

7.5.2 Resource Types

In this experiment, we study the impact of increasing the number of resource types. All the parameters are the same as in the previous experiment, except that we fix the available resource quantity and vary the number of resource types. There are eight agents. Each agent has four tasks to do with utilities of 4, 9, 16 and 25. Each task requires two items per resource type. Four agents each have only one available resource item; the other four agents with more available resources receive 11 items per resource type. We vary the number of resource types from 2 to 8. We observe how different approaches behave in different social networks. The small-world and random networks have an average degree of two. Our results are the average over 10 random instances for each social network type.

Figure 7.7 illustrates our simulation results with scale-free networks. We observe that all approaches achieve less social welfare as the number of resource types increases. This is expected as the SRAP becomes more complex. The centralized

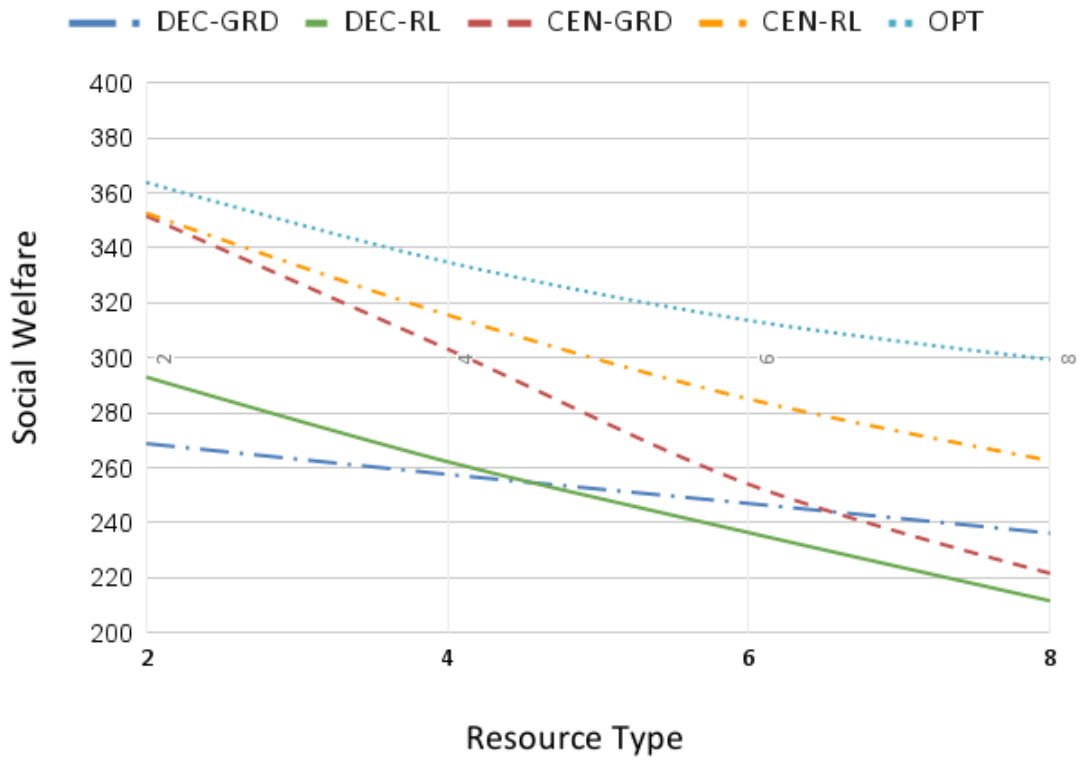

Figure 7.7: Results for varying number of resource types in scale-free networks

approaches, CEN-RL and CEN-GRD perform well compares to the decentralized approaches for lower number of resource types, and they achieve social welfare results close to the optimal solution, OPT. Notably, CEN-RL outperforms CEN-GRD and the decentralized approaches for any number of resource types. We also observe that CEN-GRD declines sharply at the higher end of parameter space. On the other hand, decentralized approaches decline more gracefully, and sustain a consistent gap relative to OPT. DEC-RL performs better than DEC-GRD for lower number of resource types. However, DEC-GRD is more stable, and even outperforms CEN-GRD when there are eight resource types.

Figure 7.8 illustrates our simulation results with small-world networks. Similar to the results for the scale-free networks, we observe that the centralized learning-based approach, CEN-RL exhibits a superior performance and achieves social welfare results close to the optimal solution, OPT, consistently as the number of resource types increases. However, CEN-GRD declines sharply and underperforms at the higher

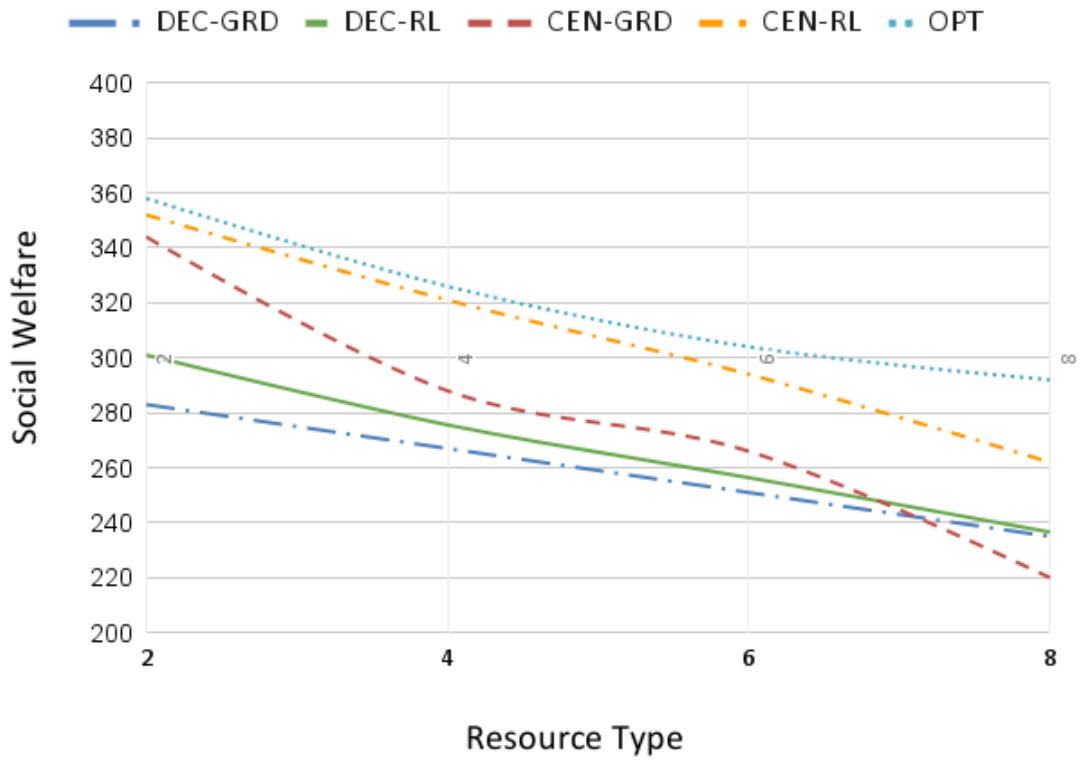

Figure 7.8: Results for varying number of resource types in small-world networks

end of parameter space. We also observe that DEC-RL outperforms DEC-GRD for any number of resource types. Notably, the learning-based approaches, CEN-RL and DEC-RL outperform the greedy approaches, CEN-DRD and DEC-GRD when there are eight resource types. Hence, while the centralized approaches, CEN-RL and CEN-GRD are superior at the lower end, the learning-based approaches, CEN-RL and DEC-RL are dominant at the higher end of the parameter space.

Figure 7.9 illustrates our simulation results with random networks. We observe that the results are similar to the ones for the other networks. The centralized approaches, CEN-RL and CEN-GRD achieve social welfare results close to the optimal solution, OPT, when there are two resource types. However, they decline even more sharply for higher numbers of resource types. Again, we observe that the decentralized approaches are more scalable and decline more gracefully as the number of resource types increases. This is noticeable as they outperform CEN-GRD, when there are eight resource types. It is also evident that CEN-RL outperforms all the other

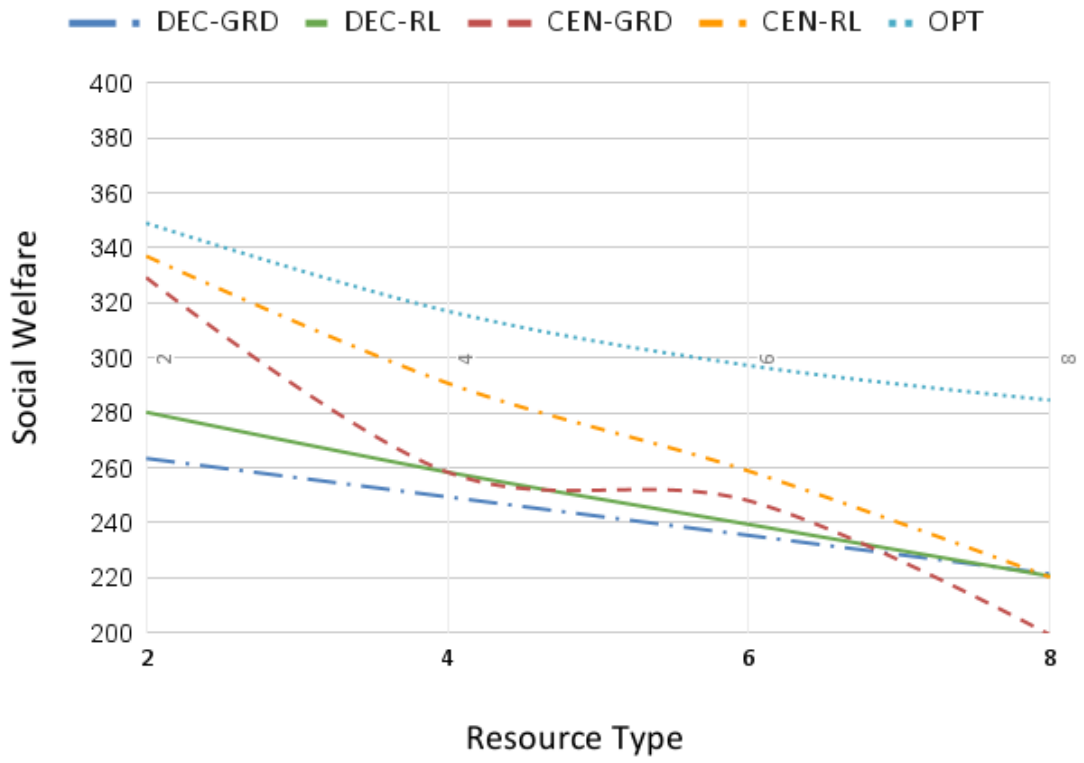

Figure 7.9: Results for varying number of resource types in random networks

approaches, and exhibits superior performance when the number of resource types is low. Notably, both learning-based approaches, CEN-RL and DEC-RL outperform their greedy counterparts, CEN-GRD and DEC-GRD for most of the parameter space.

7.5.3 Tasks

In this experiment, we study the impact of increasing the number of tasks per agent, while all the tasks have the same utility. Similar to the previous experiment setup, there are eight agents: the four agents with shortage of resources receive only one item, and the other four agents with more available resources receive 11 items per resource type. There are two resource types. All the tasks have the same utility of five, and require two resource items per resource type. We vary the number of tasks per agent from one to four in our simulation runs. We observe how different approaches behave in different types of social networks. The small-world and random

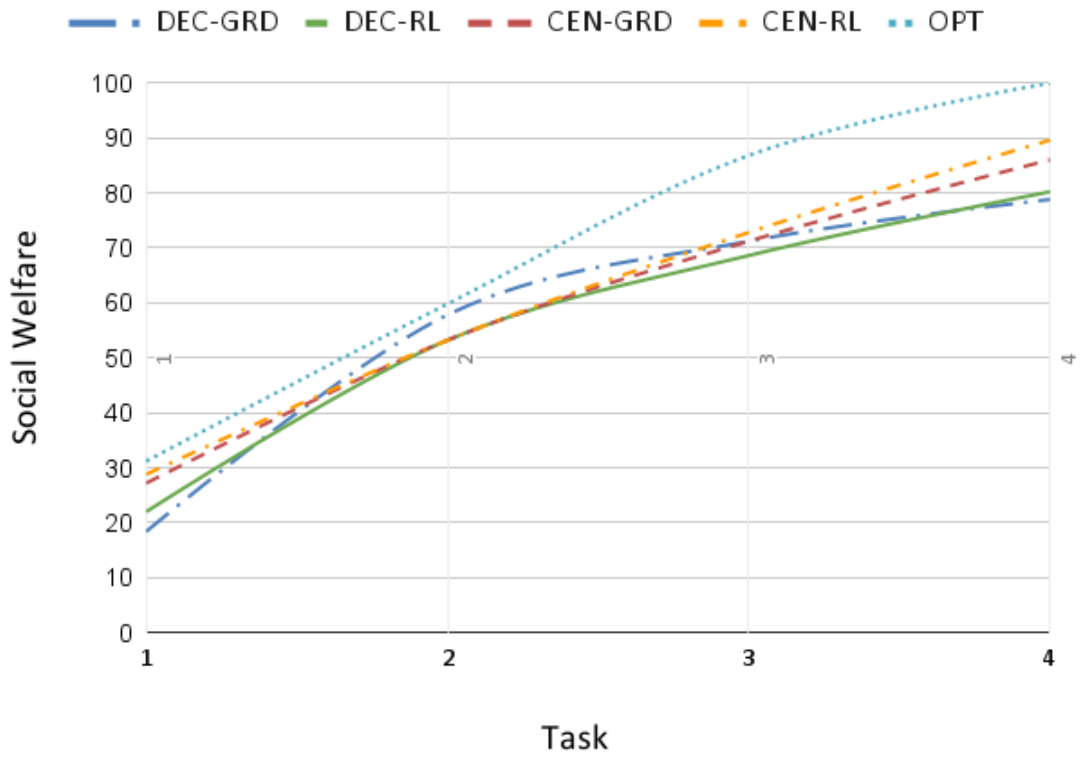

Figure 7.10: Results for varying number of tasks in scale-free networks

networks have an average degree of two. Our results are the average over 10 random instances for each social network type.

Figure 7.10 illustrates our simulation results with scale-free networks. We observe that for lower numbers of tasks, all the approaches perform well and with a little margin compared to the optimal solution, OPT. We also find that while the centralized approaches dominate for most of the parameter space, the decentralized greedy, DEC-GRD outperforms all the other approaches when there are two tasks per agent. Hence, the centralized approaches perform consistently, in this type of network, with minimal variance among them. Notably, while all the approaches achieve less social welfare compared to OPT for higher number of tasks, the centralized learning-based, CEN-RL surpass all the others when there are four tasks per agent. Moreover, CEN-RL outperforms CEN-GRD by a little margin for most of the parameter space.

Figure 7.11 illustrates our simulation results with small-world networks. In this type of network, we observe that the decentralized approaches, DEC-GRD and DEC-

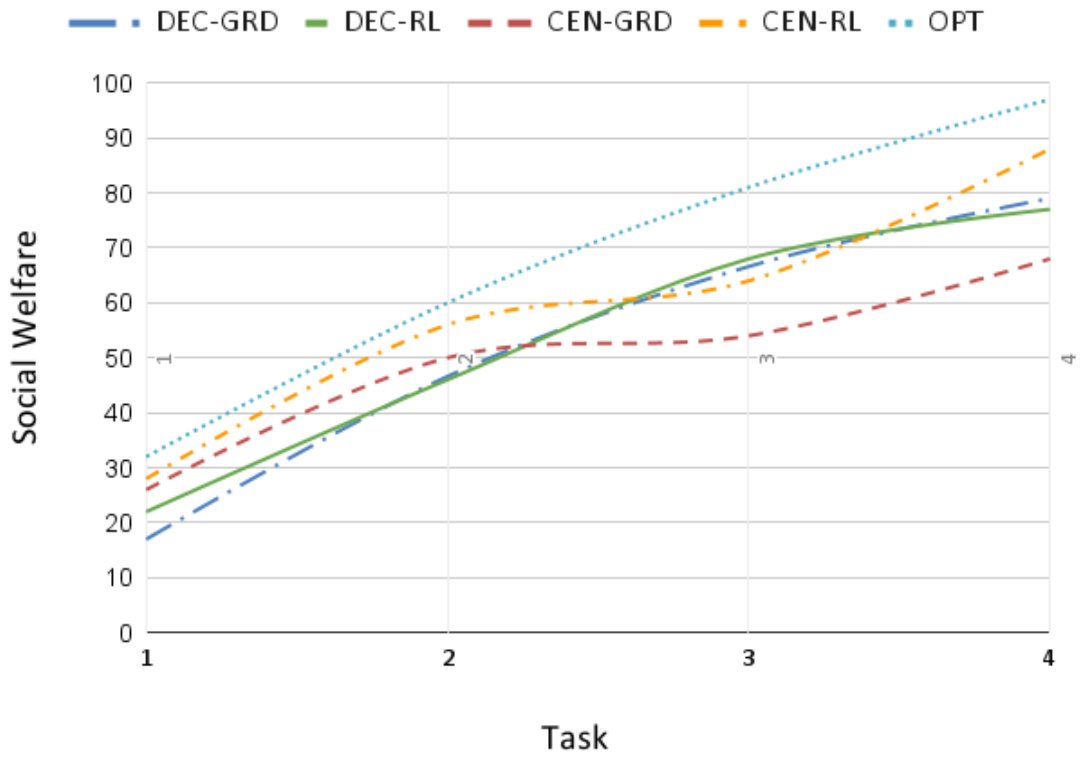

Figure 7.11: Results for varying number of tasks in small-world networks

RL perform consistently, and with a steady margin relative to OPT. On the other hand, in contrast with the results for scale-free networks, the centralized approaches exhibit more variance as the number of tasks increases. We observe that CEN-GRD performs well for lower numbers of tasks, but underperforms for higher numbers of tasks. Overall, similar to the scale-free results, CEN-RL achieves more social welfare compared to all the other approaches over the parameter space, in particular, when there are four tasks.

Figure 7.12 illustrates our simulation results with random networks. The social welfare results are similar to the ones for small-world networks. It is evident that the decentralized approaches, DEC-GRD and DEC-RL are more scalable than the centralized approaches, CEN-GRD and CEN-RL, as the number of tasks increases. In particular, we find that while CEN-GRD performs well when there are one and two tasks, it underperforms when there are three and four tasks. We also observe that similar to the results for scale-free and small-world networks, CEN-RL is dominant

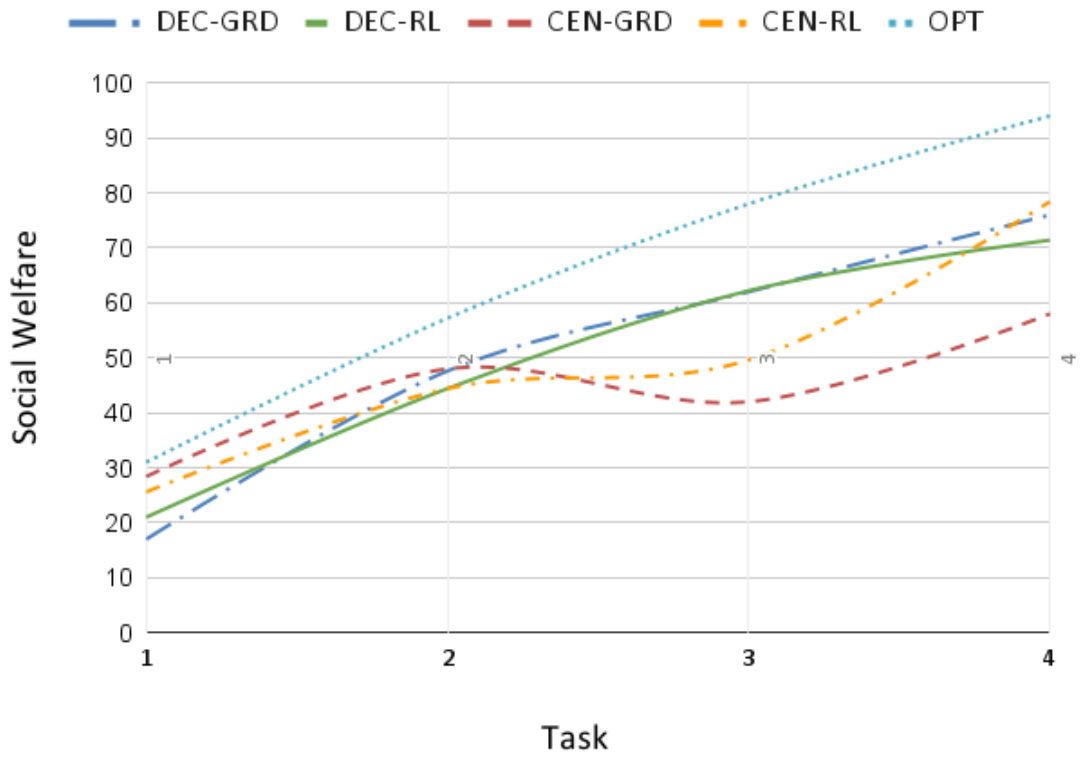

Figure 7.12: Results for varying number of tasks in random networks

at the higher end of parameter space, when there are four tasks per agent.

7.5.4 Dynamic Environment

In this experiment, we observe how the different approaches presented to solve SRAP perform in a dynamic environment. We model such environment by varying the available resource quantity and the number of tasks given to each agent; i.e. total supply and demand quantities in each episode. We also increase the number of agents to ten $(A1, A2, ..., A10)$ in order to experiment with a larger network. We simulate the dynamic environment by generating random parameters as follows. In each episode, each agent receives a number of tasks randomly chosen from the set $\{1, 2, 4\}$ with utilities randomly chosen from $\{4, 9, 16, 25\}$. There are two resource types, and each task requires two items for each resource type. Five agents $(A1, A2, ..., A5)$ act as requesters by receiving resources with a quantity randomly chosen from $\{0, 1, 2\}$ for

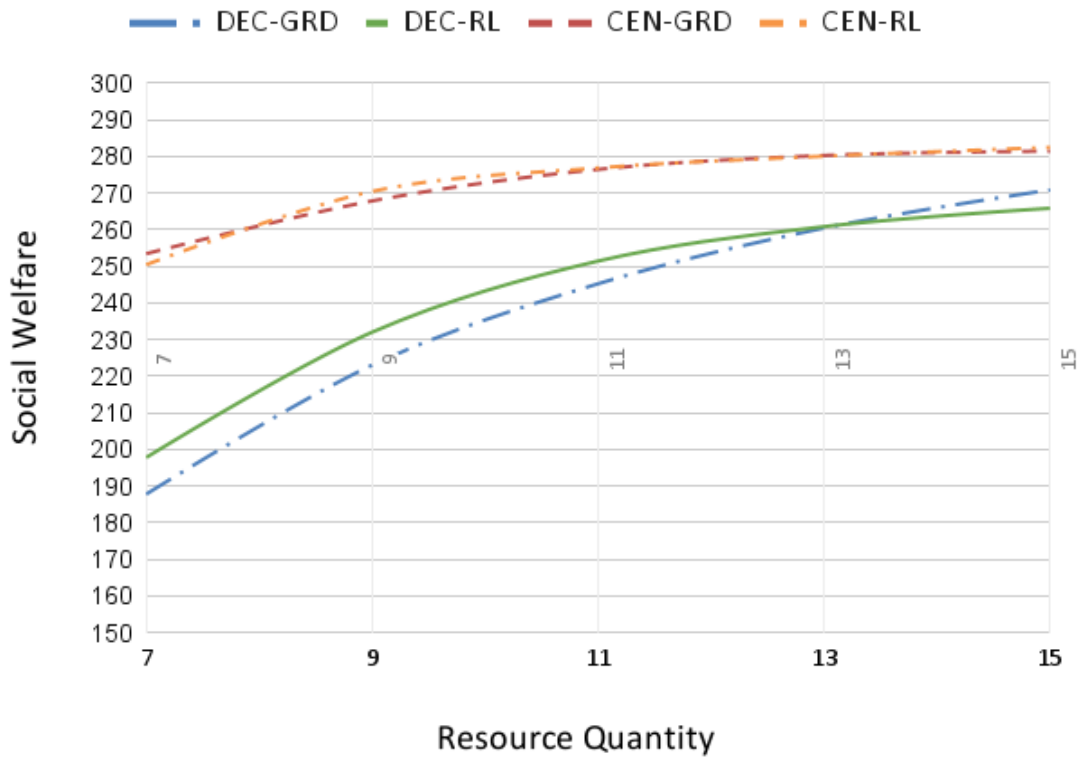

Figure 7.13: Results for varying available resource quantity in a dynamic environment in scale-free networks

each resource type. The other five agents ($A6, A7, ..., A10$) act as potential providers by receiving a fixed quantity for each resource type, varying from 7 to 15 in our experiment runs.

Similar to the previous experiments, we investigate the social welfare results in the three types of social networks. The small-world and random networks have an average degree of four; i.e. each agent has on average four direct neighbors. Since the social networks are randomly generated, our results are the average over 10 random instances for each social network type. The optimal results OPT are not computed in this experiment because of the random parameter settings. We run our experiments for 1000 episodes for each resource quantity value from 7 to 15.

Figure 7.13 illustrates our simulation results with scale-free networks. We observe that the centralized approaches outperform the decentralized approaches, in particular when the available resource quantity is low. The two centralized approaches,

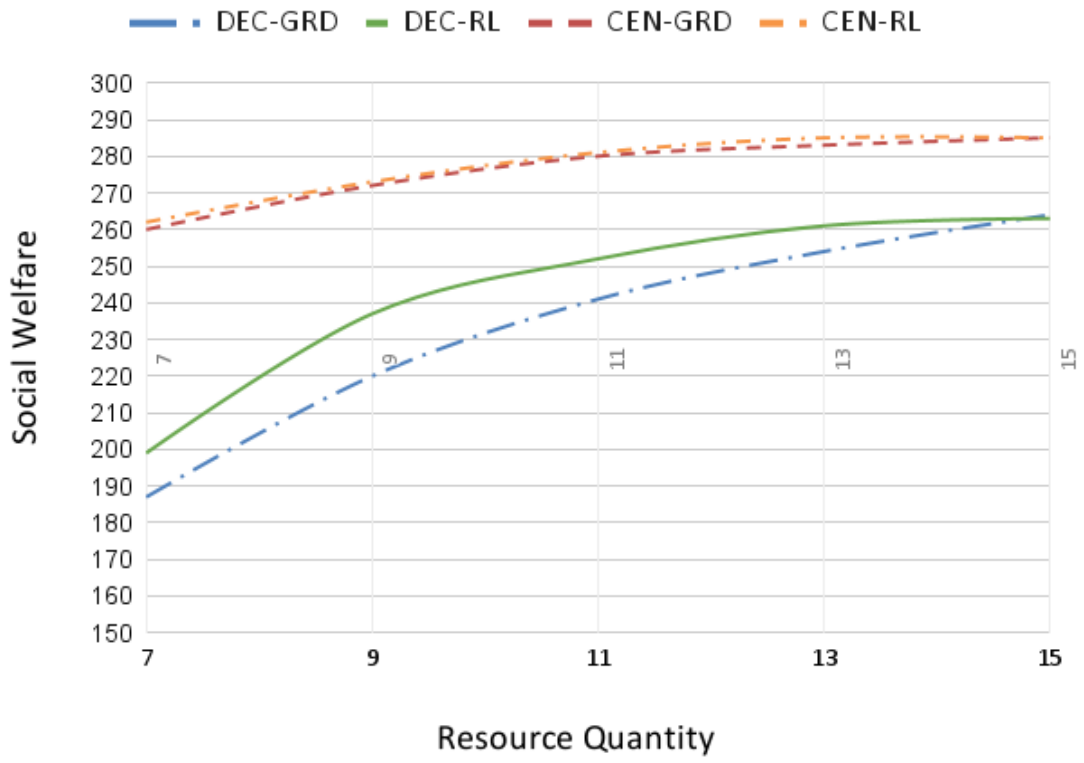

Figure 7.14: Results for varying available resource quantity in a dynamic environment in small-world networks

CEN-GRD and CEN-RL perform very closely and consistently exhibit robust performance over the parameter space. On the other hand, the decentralized approaches underperform, in particular, when there are less available resources. We also observe that DEC-RL outperforms DEC-GRD for most of the parameter space.

Figure 7.14 illustrates our simulation results with small-world networks. We observe similar results to the ones for scale-free networks. The centralized approaches, CEN-GRD and CEN-RL exhibit supervisor performance by achieving even better social welfare for lower numbers of resource quantity. We also observe that the decentralized learning-based approach, DEC-RL outperforms its greedy counterpart, DEC-GRD by a bigger margin for most of the parameter space. Figure 7.15 illustrates our simulation results with random networks. The social welfare results are very similar to the ones for small-world networks, with a little improvement for higher numbers of resource quantity.

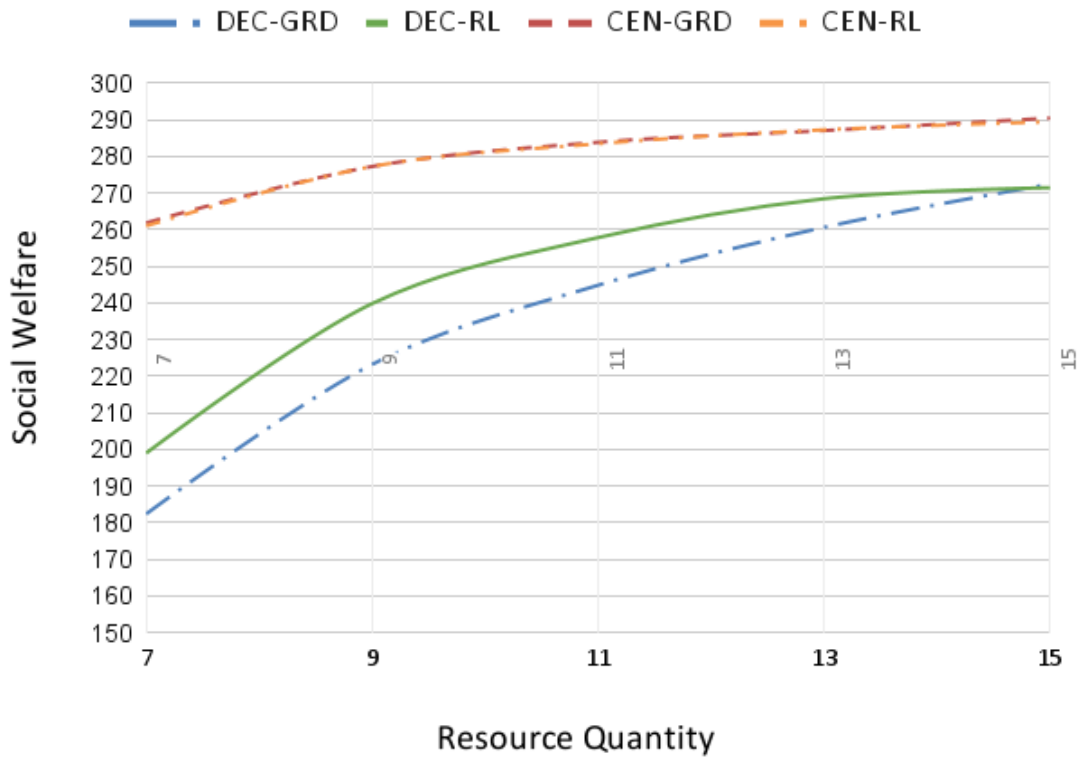

Figure 7.15: Results for varying available resource quantity in a dynamic environment in random networks

7.5.5 Transfer Cost

In this experiment, we study the impact of varying the cost incurred by transferring resources among agents. The transfer cost (Definition 4) is a function of the number of items being transferred between two agents. In this experiment, we consider a packaging policy that allows bundling resource items into packages given a predefined package size. Hence, the transfer cost becomes a function of the number of packages; and as the package size increases, the transfer cost decreases. Then, we observe how different approaches behave when we apply such packaging policy.

We use the following parameter settings. There are ten agents: five agents act as requesters by receiving three items, and the other four agents act as potential providers by receiving 11 items per resource type. There are two resource types, and two tasks per agent, with utilities of 9 and 16. Each task requires five items per resource type. Then, we vary the package size from one to five in our simulation runs

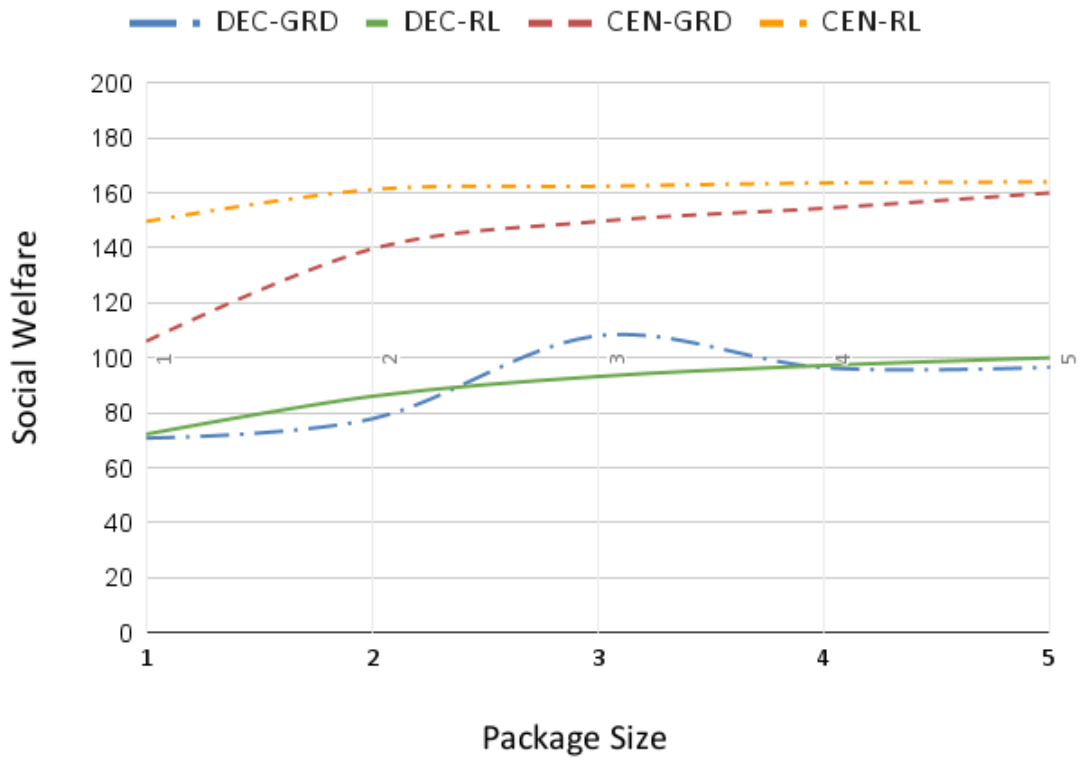

Figure 7.16: Results for varying package size in scale-free networks

with different types of social networks. The small-world and random networks have an average degree of four. Our results are the average over 10 random instances for each social network type. Given the packaging policy, we are not able to solve the centralized ILP formulation to find the optimal solution OPT, as the transfer cost definition with package size is not a linear function.

Figure 7.16 illustrates our simulation results with scale-free networks. We observe that the centralized approaches outperform the decentralized ones. Notably, the learning based approach, CEN-RL exhibits superior performance and outperforms its greedy counterpart, CEN-GRD for different values of package size. It is evident that the gap between CEN-RL and CEN-GRD is larger when package size is low and hence transfer cost is high, since CEN-GRD performs better as the package size increases and achieves social welfare close to the CEN-RL results at the higher end of parameter space. We also observe that DEC-RL achieves consistent results for different package sizes, and outperforms DEC-GRD by a little margin for most of the

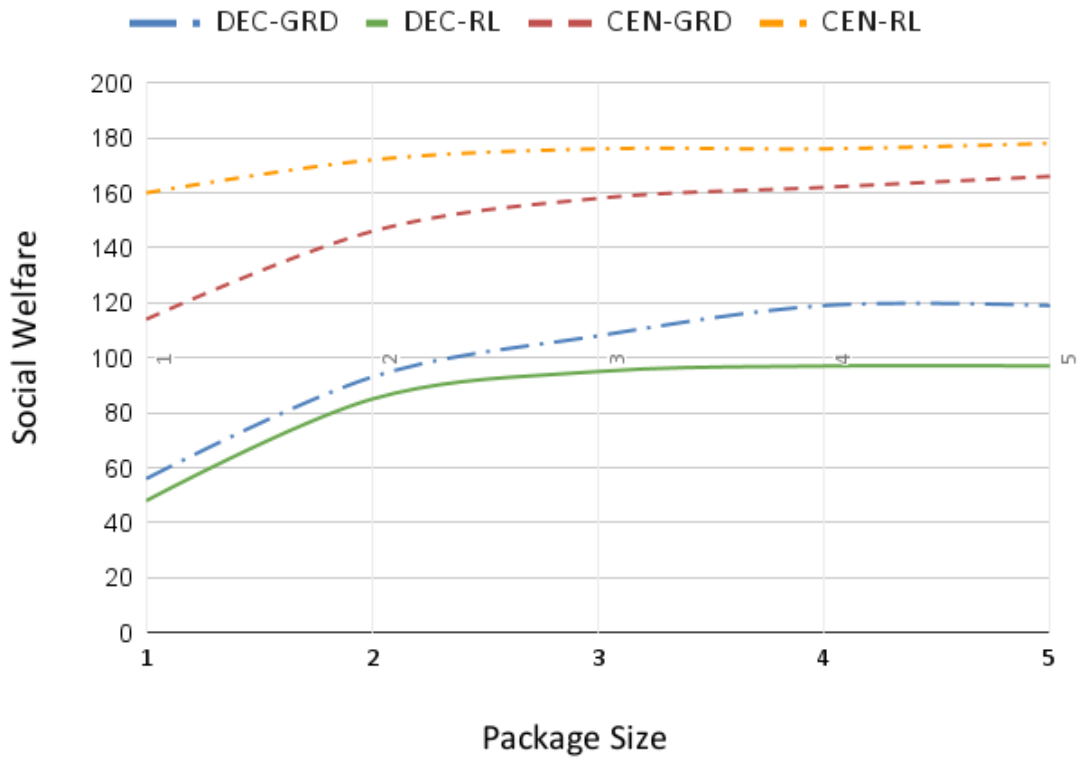

Figure 7.17: Results for varying package size in small-world networks

parameter space.

Figure 7.17 illustrates our simulation results with small-world networks. We observe that the centralized learning-based approach, CEN-RL achieves even higher social welfare compared to the result for scale-free networks. Likewise, DEC-GRD performs better in small-world networks, and the margin between DEC-GRD and DEC-RL is larger over all the parameter space.

Figure 7.18 illustrates our simulation results with random networks. Similar to the results for other networks, the centralized approaches exhibit superior performance. We find that the gap between CEN-RL and CEN-GRD is smaller in random networks. We also observe that DEC-RL outperforms DEC-GRD only for smaller package sizes.

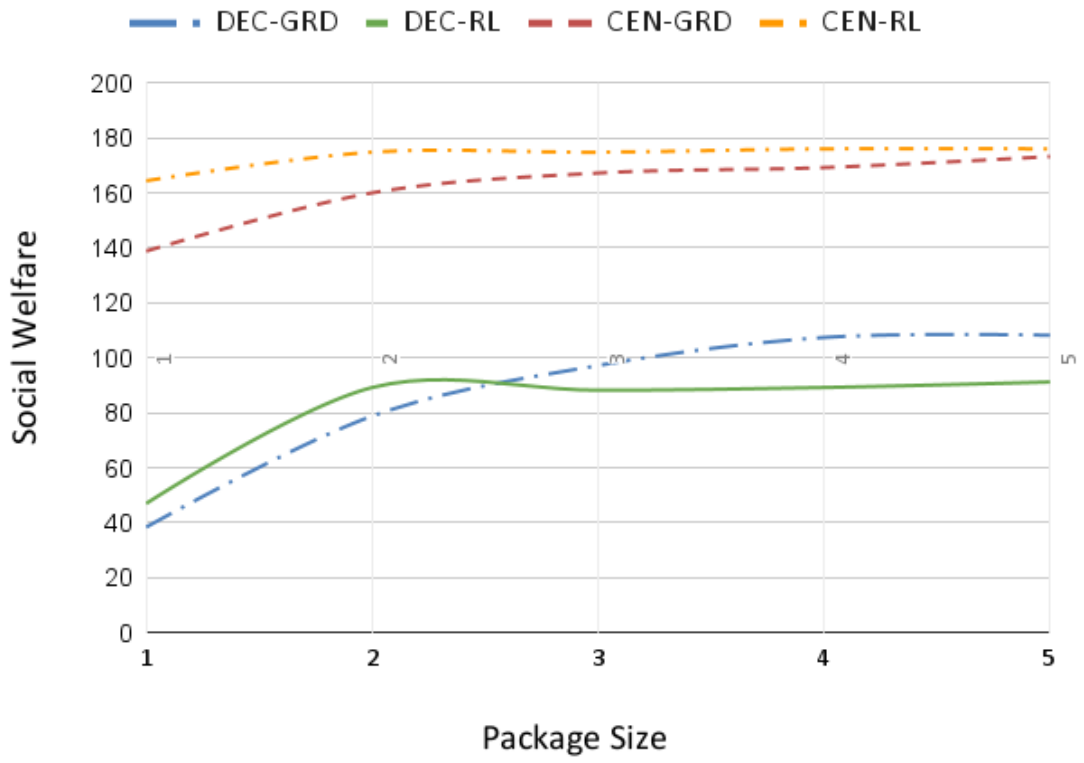

Figure 7.18: Results for varying package size in random networks

7.5.6 Agents

In this experiment, we evaluate the scalability of our centralized and decentralized greedy approaches by increasing the number of agents in the social network. We experiment with different social network types comprising 8, 16, 24, 32 and 40 agents. In cases of small-world and random networks, we use a network average degree of 2, 4, 6, 8, and 10 respectively. The learning-based approaches, CEN-RL and DEC-RL are excluded in this experiment, as the training phase requires more computation time, in particular for the centralized approach CEN-RL. We measure the quality of the greedy approaches, CEN-GRD and DEC-GRD by computing the percentage ratio between the social welfare results achieved by them and by the centralized optimal solution: $100 * socialWelfare_{GRD}/socialWelfare_{OPT}$

In each scenario, the agents are divided into two groups with different available resource quantities: one half faces shortage of resources, receiving only one item each, while the other half has more available resources, with five items per resource type

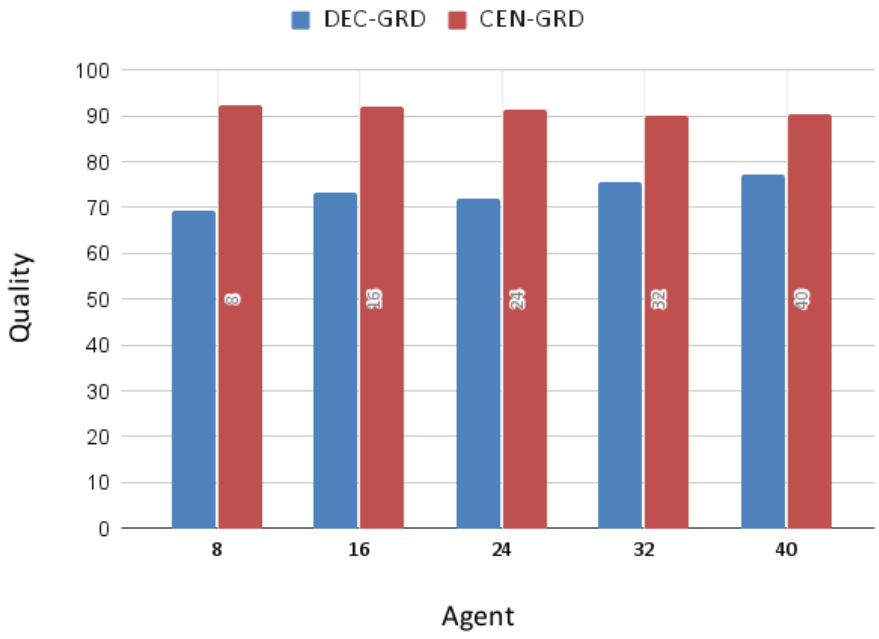

Figure 7.19: Results for varying number of agents in scale-free networks

given to each agent. There are two resource types, and two tasks per agent, with utilities of 9 and 16. Each task requires two items per resource type. Our results are the average over 10 random instances for each social network type.

Figure 7.19 illustrates our simulation results with scale-free networks. We observe that as the number of agents increases, the quality of decentrzalized greedy approach, DEC-GRD increases. On the other hand, the quality of the centralized greedy approach, CEN-GRD slightly decreases for higher numbers of agents. Hence, we find that the gap between the quality of DEC-GRD and CEN-GRD decreases as the number of agents increases. This suggests that while both approaches perform well compared to the optimal solution, DEC-GRD is more scalable than CEN-GRD with increasing number of agents in scale-free networks.

Figure 7.20 illustrates our simulation results with small-world networks. We observe that both CEN-GRD and DEC-GRD exhibit consistent qualities as the number of agents increases. Similar to the results for scale-free networks, we find that the gap between qualities of CEN-GRD and DEC-GRD is slightly less for higher numbers of agents. This suggests that DEC-GRD is more scalable than CEN-GRD with increasing number of agents in small-world networks.

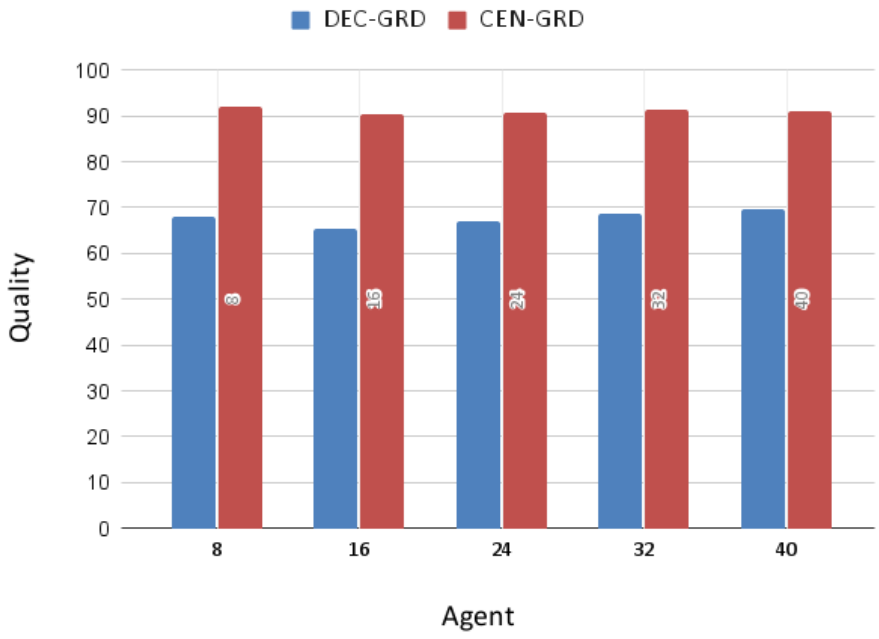

Figure 7.20: Results for varying number of agents in small-world networks

Figure 7.21 illustrates our simulation results with random networks. We observe similar results to the ones for scale-free and small-world networks. Again, we find that the quality of DEC-GRD increases and the quality of CEN-DRG decreases with more number of agents. Therefore, it results in a smaller gap among them when there are more agents in the network. Hence, it is evident that DEC-GRD is more scalable than CEN-GRD with increasing number of agents in random networks.

7.5.7 Network Degree

In this experiment, we evaluate the impact of connectivity among the agents on the quality of the centralized and decentralized greedy approaches by varying the network average degree. This experiment is only done for small-world and random networks, as we do not vary the average degree in scale-free networks. Similar to the previous experiment, the learning-based approaches are excluded in this experiment, and we measure the quality of the greedy approaches by computing the percentage ratio between the social welfare results achieved by them and by the centralized optimal solution: $100 * socialWelfare_{GRD}/socialWelfare_{OPT}$

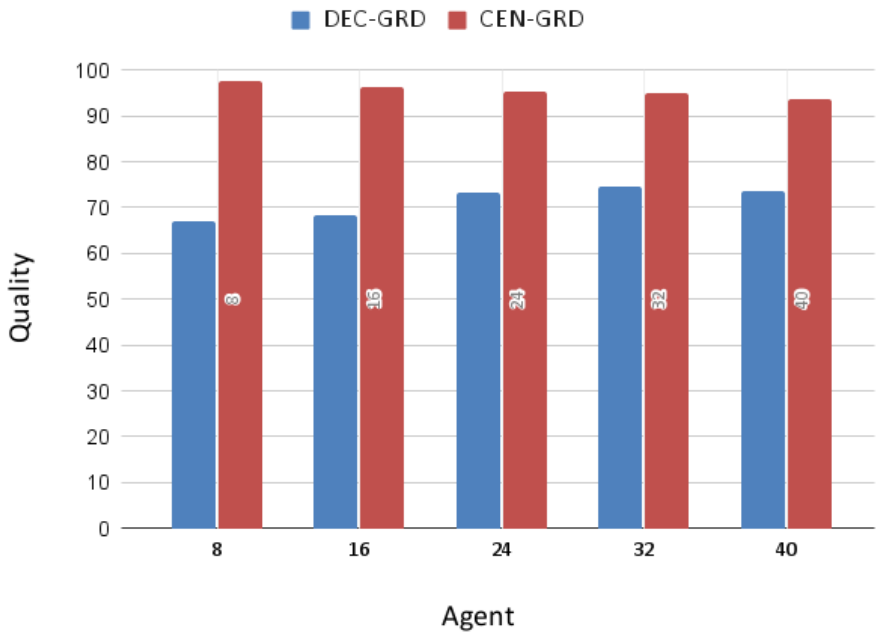

Figure 7.21: Results for varying number of agents in random networks

We use the following parameter settings similar to the ones for previous experiment, except that the number of agents is fixed. There are ten agents in the network. Again, we divide them into two groups. Five agents face shortage of resources and act as requesters by receiving only one item each, and the other five agents have extra available resources and act as potential providers by receiving five items each per resource type. There are two resource types, and two tasks per agent, with utilities of 9 and 16. Each task requires two items per resource type. Then, we vary the network average degree from two to eight in our simulation runs. Our results are the average over 10 random instances for each social network type.

Figure 7.22 illustrates our simulation results with small-world networks. We observe that the quality of centralized greedy approach, CEN-GRD slightly increases with higher values of network degree. Therefore, we find that CEN-GRD performs better in highly connected small-world networks. Conversely, we observe that the quality of decentralized greedy approach, DEC-GRD is slightly better at the network degree of four. This could be because of more unused transferred resource items, while incurring extra transfer cost when there are many connections among the agents. This suggests that DEC-GRD performs better in partially connected small-world networks.

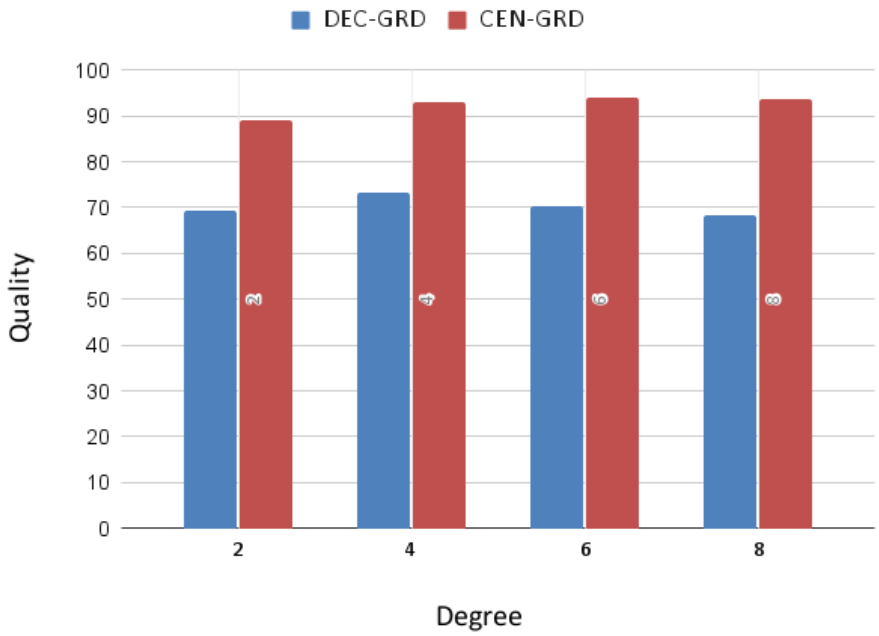

Figure 7.22: Results for varying network average degree in small-world networks

Therefore, the gap between qualities of CEN-GRD and DEC-GRD is slightly less in such network settings.

Figure 7.23 illustrates our simulation results with random networks. We observe that the quality of centralized greedy approach, CEN-GRD is consistent as the network degree increases. On the other hand, the results show that the quality of decentralized greedy approach, DEC-GRD is slightly better at the network degree of four. Hence, similar to the results for small-world network, we find that DEC-GRD performs better in partially connected random networks, resulting in the gap between qualities of CEN-GRD and DEC-GRD to be slightly less in such network settings. Again, this is caused by extra transfer cost of resource items, which might not be used in performing tasks. In other words, when each agent has many connections, the cascading of requests and offers in the decentralized protocol is more complex, and may result in extra transfer cost.

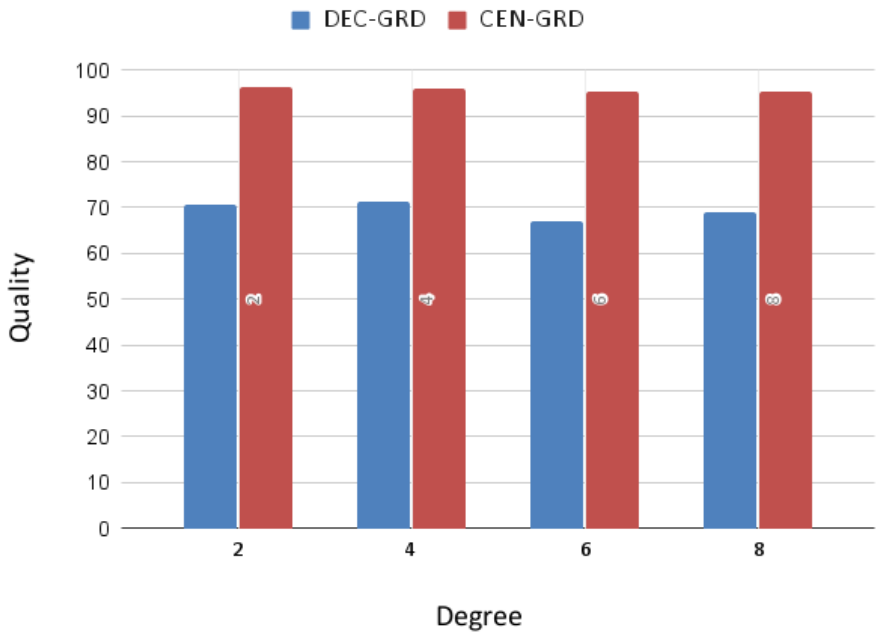

Figure 7.23: Results for varying network average degree in random networks

7.6 Discussion

The series of experiments outlined in the previous sections provide a comprehensive investigation into the performance of centralized and decentralized approaches, incorporating both greedy and learning-based methods, under various parameter settings. The experiments are designed to vary the available resource quantity per agent, the number of resource types, the number of tasks per agent, the environmental dynamics, transfer costs, the number of agents, and the network average degree. Each experiment offers insights into the efficiency and scalability of our different approaches (CEN-GRD, CEN-RL, DEC-GRD, and DEC-RL) in solving the SRAP. In the following sections, we compare their overall performance and discuss their strengths and limitations across different scenarios. Additionally, we discuss the practical implications of our findings and provide recommendations for real-world applications, considering when to employ centralized versus decentralized approaches and when to leverage greedy or learning-based methods.

7.6.1 Centralized vs. Decentralized

Overall, the centralized approaches, CEN-GRD and CEN-RL exhibit superior performance across different social network types and experimental setups. This superiority is attributed to their global view of the social network and complete knowledge of tasks and resources, allowing for near-optimal resource allocation solutions that maximize the social welfare.

From the experiments with resource ratio and resource types, we find that the centralized approaches consistently perform close to the optimal solution, particularly evident in scale-free and small-world networks. In addition, the experiment with a dynamic environment further demonstrates the efficiency of centralized decision-making, particularly under varying supply and demand quantities. However, the performance of centralized approaches substantially diminishes in two scenarios. First, we observe a significant decline with an increase in the number of resource types, indicating an increased complexity in allocating multiple types of resources. Second, as we observe in the experiment with number of tasks, the centralized approaches become less effective with higher number of tasks, indicating another complexity factor in the centralized processes.

On the other hand, the decentralized approaches, DEC-GRD and DEC-RL, while generally underperforming compared to the centralized ones, demonstrate higher efficiency in the settings with higher resource availability or lower number of tasks, as they begin to close the gap with the centralized ones. In addition, by comparison to the optimal solution, they demonstrate higher scalability in the experiments with increasing number of resource types, tasks, and agents. This resilience could be attributed to the localized decision making and self-organizing behavior, which is advantageous as the complexity of SRAP increases. This suggests that the decentralized approaches might be more effective in larger social networks with more complex task requirements, where the centralized approaches become less feasible.

It is also important to note that there is a fundamental difference in our centralized and decentralized approaches, which lies in our algorithm design for allocating resources to tasks in each approach. In the centralized approaches, the master agent iteratively selects feasible tasks and allocates required resource to them. This process is repeated until all available resources are allocated to the feasible tasks. The key point is that only one task is considered in each iteration in the centralized approach. On the other hand, in the decentralized negotiation-based protocol, each agent is

allowed to create a request for its missing resource quantity for a bundle of tasks. Hence, by confirming one or more potential offers, the requester agent may be able to perform one or more tasks given its updated available resources. This bundling behavior in the decentralized approach can potentially improve its scalability with increasing number of tasks, as we observe in the respective experiment.

7.6.2 Greedy vs. Learning-based

The centralized and decentralized greedy approaches, CEN-GRD, DEC-GRD provide a baseline performance in all the experiments. They are particularly effective when the problem is less complex, as observed in the experiments with higher number of available resources, and lower number of resource types. However, as the complexity of problem increases, the performance of both CEN-GRD and DEC-GRD begins to degrade. This decline is evident in experiments involving a higher number of resource types, dynamic supply and demand settings in dynamic environments, and higher transfer costs introduced by the packaging policy.

On the other hand, the centralized and decentralized learning-based approaches, CEN-RL and DEC-RL, demonstrate a higher degree of adaptability and consistently outperform their greedy counterparts in more complex scenarios. These learning-based approaches are able to optimize their decision-making over time, learning from the environment and adjusting their policies to maximize the social welfare. In particular, we observe the superiority of CEN-RL in experiments with higher number of resource types, and higher transfer costs. Similarly, DEC-RL is superior to DEC-GRD in experiments with varying resource ratio, and dynamic environments. This highlights the adaptability of learning-based approaches in optimizing their policies in scenarios with high-dimensional decision spaces, dynamic settings, and non-linear relations.

7.6.3 Social Network Type

Our simulation experiments with centralized and decentralized, greedy and learning-based approaches present consistent social welfare results across different types of social networks in small settings. This consistency suggests the adaptability of our different approaches to solve the SRAP in such social network settings. One might generally expect to observe better performance with the learning-based approaches in small-world and scale-free networks compared to random networks. This is because of

the structured nature of their topology, which might be better captured in the learning process for selecting the best task manager in the centralized approach, and the best offer and confirmation when negotiating on resources in the decentralized protocol. Observing such behavior requires more extensive simulation experiments with larger social networks and varying network degrees. Larger social networks would amplify the impacts of network type and degree on social welfare results, providing clearer insights into how our proposed methods perform in more complex, real-world social network settings.

7.6.4 Practical Implications and Recommendations

In the experiments with lower number of agents, lower number of resource types, and lower transfer cost, we find that the centralized greedy approach might be sufficient to achieve near-optimal outcomes at lower computational costs. However, in the experiments with higher number of agents and higher number of resource types, we observe better scalability of the decentralized protocol as the number of agents and number of resource types increase. The decentralized social welfare results in such experiments suggest a significant potential for real-world large-scale applications, where a centralized approach is impractical due to computational cost, or unjustified due to safety and security concerns regarding a single point of failure. Therefore, it might be advantageous to deploy the decentralized protocol in such large-scale scenarios.

In addition, the adaptability of learning-based approaches in dynamic environments, make them suitable for real-world applications, in which the supply and demand quantities change over time. In particular, in scenarios with real-time constraints, the decentralized learning-based approach is recommended due to its self-organizing behavior and less computational time in the learning process. Therefore, in large social networks with both dynamic and real-time requirements, the decentralized learning-based approach might be the best candidate to ensure both scalability and adaptability at a lower computational overhead, while presenting timely solutions to the SRAP. In the followings, we describe two real-world applications, and discuss the suitability of our solutions.

Hospital Resource Management

An instance of the SRAP can be seen in the hospital resource management. In a network of hospitals, efficient allocation of resources is crucial for enhancing patient

care across multiple locations. In particular, this is evident in situations like the COVID-19 pandemic, where the supply and demand quantities change over time, with sudden spikes in number of patients or sudden shortages of medical supplies, such as ventilators, ICU beds, and medical personnel. In such scenarios, a centralized learning-based approach like CEN-RL could dynamically allocate resources, and achieve near-optimal social welfare at least for small networks. However, in larger networks with more number of hospitals, patients, and resource types, the centralized learning process may become impractical. Hence, a decentralized approach like DEC-RL might be more appropriate. The rationale is that a decentralized protocol has the self-organizing behavior. It supports autonomy and allows the individual hospitals to act more promptly to local changes in supply and demand, without the need for a central control.

Data Center Resource Management

Another instance of the SRAP can be seen in the data center resource management. A network of data centers can leverage an optimization process when allocating cloud resources to their computing jobs. Given limited number of resource types, including computing power, memory, storage, and network bandwidth, a centralized learning-based approach like CEN-RL might be the best candidate for solving the cloud-based resource allocation problem in small networks. On the other hand, in case of larger networks with many data centers, a centralized greedy approach like CEN-GRD might be more appropriate, since it is computationally faster than the learning-based approach. However, with increasing number of tasks or computing jobs, a centralized approach may not be scalable. In particular, in scenarios that the jobs have deadlines, a decentralized approach like DEC-RL might be a better candidate by enhancing responsiveness and reducing latency, while achieving social welfare results close to the centralized ones.

7.6.5 Limitations

Our evaluation using simulation experiments in this chapter includes the following limitations. We assume that the agents are always responsive and execute the processes in Algorithm 5 consecutively. We also assume that the communication channel is reliable, allowing the messages to be received after a known transmission delay. Moreover, we simulate their interaction phases in a synchronous approach. This is

done by including a rejection response for requests; i.e. any received request must be replied with an offer with $q_{off} \geq 0$, where $q_{off} = 0$ implies a rejection. Therefore, we can run our experiments synchronously without incorporating real timeouts, allowing us to run for many episodes, in particular in the learning-based approach. However, this might not be practical in a real-world scenario, where agents may be down, and therefore not responsive, or the messages may be lost, causing their neighbors to be waiting for replies to their requests. That is why our protocol design in chapter 6 involves asynchronous interactions by applying timeouts, allowing the interactive behaviour to be reliable. Further limitations in our evaluation include limited experiments, number of agents, number of tasks, task requirements, and transfer costs. In addition, our learning-based results are based on a limited number of training episodes.

Chapter 8

Conclusions and Future Work

In this thesis, we investigate the social resource allocation problem (SRAP) in multi-agent systems, focusing on maximizing social welfare by efficiently distributing limited resources in social networks with limited connectivity. The SRAP is relevant in various real-world scenarios, such as healthcare, disaster response, cloud computing, where different types of resources need to be allocated to participants distributed along the social network. We provide a comprehensive analysis of solving SRAP in three types of social networks, including small-world, scale-free, and random networks. We explore both centralized and decentralized approaches, utilizing different heuristics, greedy, Markov Decision Process (MDP), deep Q-learning, and optimization techniques summarized as follows:

- *DEC-GRD*: a decentralized negotiation-based protocol with greedy offering and confirming using request utilities and offer costs
- *DEC-RL*: a decentralized negotiation-based protocol with learning-based offering and confirming by modeling the offering and confirming processes as MDPs
- *CEN-GRD*: a centralized greedy approach using task utilities and resource transfer costs
- *CEN-RL*: a centralized reinforcement learning approach by modeling the task selection process as a MDP
- *OPT*: a centralized optimal solution by modeling the SRAP as an integer linear programming (ILP)

The simulation results of this research offers valuable insights into the strengths and limitations of different approaches (DEC-GRD, DEC-RL, CEN-GRD, and CEN-RL) to solve the SRAP in different social networks. Our findings can be summarized as follows.

- *Centralized vs. Decentralized*: the centralized greedy and learning-based approaches leverage global knowledge in their decision making process. These methods generally achieve higher social welfare results compared to the decentralized greedy and learning-based approaches. However, the centralized approaches struggle with scalability when the number of agents, tasks, or resource types increases. On the other hand, the decentralized approaches are more suitable for large-scale systems. In addition, they can avoid a single point of failure and exhibit self-organization, since they only rely on local knowledge and local interactions among the agents.

- *Greedy vs. Learning-Based*: the greedy mechanisms offer a straightforward mechanism to solve the SRAP, by prioritizing tasks based on their efficiency. These methods are effective in static environments and for less complex settings. On the other hand, the learning-based mechanisms are more suitable in dynamic environments, where the resource supply and demand quantities change over time. They exhibit adaptability by optimizing their policies over time, in particular, when the problem has high-dimensional decision spaces, dynamic settings, and non-linear relations.

- *Practical Implications*: The scalability of the decentralized protocol (DEC-GRD or DEC-RL) makes it highly suitable for large social networks, where a centralized approach (CEN-GRD or CEN-RL) might be impractical because of high computational cost. In addition, the ability to adapt to changes in dynamic environments is crucial for effective resource allocation in real-world scenarios. This research highlights the advantage of incorporating deep reinforcement learning in developing adaptive approaches that respond to evolving supply and demand quantities. The learning-based mechanisms presented in this thesis, particularly DEC-RL, exhibit promising performance in such dynamic settings. Furthermore, the self-organization property of the decentralized protocol might be advantageous in scenarios with safety and security concerns regarding a single point of failure. Therefore, the decentralized learning-based approach (DEC-

RL) might be the best candidate to ensure efficiency, scalability, adaptability, and self-organization in real-world large-scale applications.

It is important to note that there are some limitations in our formulation of the social resource allocation problem (SRAP), as well as our presented mechanisms and simulation models. In the followings, we outline a number of future research directions that could address those limitations.

- One of our main assumptions is that the social network structure does not vary over time, i.e. during the episodes in our simulation experiments. This allows the learning-based methods to perform well given static nature of the network environment. However, in many real-world scenarios, the networks might be dynamically changing over time, by adding or removing nodes and connections. In such scenarios, pre-existing trained models might not be as effective as in the static case, depending on the degree of variation in the network. Therefore, future work could incorporate online learning to continuously update the learning models in order to adapt to the changes in network environment.

- Another assumption in this thesis is that the agents are considered to be honest, and to have a common goal to increase the social welfare. This allows the agents to trust each other in their negotiations and processing of request utility and offer cost functions. However, in a more general scenario where agents might be deceptive and also pursuing their individual goals in addition to the common goal, future work could investigate a game-theoretic approach to design their interactions, and collaborative behaviour.

- In our simulation experiments, we assume that the agents are always responsive; and also the communication channel is reliable; i.e. the messages are received after a known transmission delay. An extension of this work could focus on possibility of unresponsive agents and lossy communication. Another possibility is to consider malicious interference by a third party, where the communication channel could be intercepted or the messages could be changed. In such environments, the agents should still be able to make theirs decisions and continue their operations.

- In our decentralized learning-based approach (DEC-RL), we model the offering and confirming action spaces based on all the possible offering and confirming

quantities up to the requested quantity in the negotiation process. In scenarios with large resource quantities or continuous numbers, action space discretization might be necessary in order to reduce the complexity of the action space, leading to a more efficient learning process. Alternatively, other reinforcement learning techniques such as policy gradient or actor-critic could be explored in order to handle a large space of actions.

- Future research could also explore hybrid approaches that combine the centralized and decentralized mechanisms. Combining the strengths of global optimization and local adaptability may offer more effective solutions in complex social network settings and dynamic environments.

- Future work should also focus on validation with real data by targeting one or more specific real-world instances of the SRAP, such as healthcare or cloud resource management. Experimenting with real-world social networks can provide valuable insights on the quality of our centralized and decentralized approaches, and also the sensitivity of our results to various parameters used in our algorithms and learning models.

Bibliography

[1] Deeplearning4j: Open-source, distributed deep learning for the jvm. `https://deeplearning4j.konduit.ai`, 2023. Last accessed: September 2023.

[2] Fipa: The foundation for intelligent physical agents. `http://www.fipa.org`, 2023. Last accessed: March 2023.

[3] Samir Aknine, Suzanne Pinson, and Melvin F Shakun. An extended multi-agent negotiation protocol. *Autonomous Agents and Multi-Agent Systems*, 8(1):5–45, 2004.

[4] Bo An, Victor Lesser, and Kwang Mong Sim. Strategic agents for multi-resource negotiation. *Autonomous Agents and Multi-Agent Systems*, 23(1):114–153, 2011.

[5] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[6] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[7] Emanuele Blasioli, Bahareh Mansouri, Srinivas Subramanya Tamvada, and Elkafi Hassini. Vaccine allocation and distribution: a review with a focus on quantitative methodologies and application to equity, hesitancy, and covid-19 pandemic. In *Operations research forum*, volume 4, page 27. Springer, 2023.

[8] Engin Bozdag. A survey of extensions to the contract net protocol. *Technical report, CiteSeerX-Scientific Literature Digital Library and Search Engine*, 2008. `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ececc072fedc78af4589df6be540c2a391642814`.

[9] Davide Calvaresi, Kevin Appoggetti, Luca Lustrissimini, Mauro Marinoni, Paolo Sernani, Aldo Franco Dragoni, and Michael Schumacher. Multi-agent systems'

negotiation protocols for cyber-physical systems: Results from a systematic literature review. In *Proc. of the 10th Int. Conf. on Agents and Artificial Intelligence (ICAART 2018)*, volume 1, pages 224–235. SCITEPRESS, 2018.

[10] Davide Calvaresi, Mauro Marinoni, Arnon Sturm, Michael Schumacher, and Giorgio Buttazzo. The challenge of real-time multi-agent systems for enabling iot and cps. In *Proc. of the Int. Conf. on Web Intelligence (WI '17)*, pages 356–364. ACM, 2017.

[11] Yann Chevaleyre, Ulle Endriss, and Nicolas Maudet. Distributed fair allocation of indivisible goods. *Artificial Intelligence*, 242:1–22, 2017.

[12] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[13] Mathijs De Weerdt, Yingqian Zhang, and Tomas Klos. Distributed task allocation in social networks. In *Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 1–8, 2007.

[14] Mathijs M de Weerdt, Yingqian Zhang, and Tomas Klos. Multiagent task allocation in social networks. *Autonomous Agents and Multi-Agent Systems*, 25(1):46–86, 2012.

[15] Maria del Carmen Delgado-Roman and Carles Sierra. A multi-agent approach to energy-aware wireless sensor networks organization. In *Agreement Technologies: Proc. of 2nd Int. Conf.*, pages 32–47. Springer, 2013.

[16] John A Doucette, Graham Pinhey, and Robin Cohen. Multiagent resource allocation for dynamic task arrivals with preemption. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–27, 2016.

[17] Wilfried Elmenreich, Raissa D'Souza, Christian Bettstetter, and Hermann de Meer. A survey of models and design methods for self-organizing networked systems. In *Int. Workshop on Self-Organizing Systems*, pages 37–49. Springer, 2009.

[18] S. Shaheen Fatima and Michael Wooldridge. Adaptive task and resource allocation in multi-agent systems. In *Int. Conf. on Autonomous Agents*, pages 537–544, 2001.

[19] FIPA. *FIPA Contract Net Interaction Protocol Specification*. FIPA, 2001.

[20] Xiangqiang Gao, Rongke Liu, and Aryan Kaushik. Hierarchical multi-agent optimization for resource allocation in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):692–707, 2020.

[21] Athina Georgara, Juan A. Rodríguez-Aguilar, and Carles Sierra. Allocating teams to tasks: an anytime heuristic competence-based approach. In *Multi-Agent Systems: Proc. of 19th European Conf.*, pages 152–170. Springer, 2022.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] Wenxia Guo, Wenhong Tian, Yufei Ye, Lingxiao Xu, and Kui Wu. Cloud resource scheduling with deep reinforcement learning and imitation learning. *IEEE Internet of Things Journal*, 8(5):3576–3586, 2020.

[24] Gurobi Optimization. Gurobi optimizer reference manual. `https://www.gurobi.com/documentation/10.0`, 2023.

[25] Hado Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[26] Chung-Yuan Huang, Chuen-Tsai Sun, and Hsun-Cheng Lin. Influence of local information on social simulations in small-world network models. *Journal of Artificial Societies and Social Simulation*, 8(4), 2005.

[27] Yichuan Jiang, Yifeng Zhou, and Wanyuan Wang. Task allocation for undependable multiagent systems in social networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(8):1671–1681, 2012.

[28] Ian Kash, Ariel D Procaccia, and Nisarg Shah. No agent left behind: Dynamic fair division of multiple resources. *Journal of Artificial Intelligence Research*, 51:579–603, 2014.

[29] Tore Knabe, Michael Schillo, and Klaus Fischer. Improvements to the fipa contract net protocol for performance increase and cascading applications. In *In Int. Workshop for Multi-Agent Interoperability at the German Conference on AI (KI-2002)*. Citeseer, 2002.

[30] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[31] Kai Li, Sentang Wu, Yongming Wen, and Ying Wang. Task allocation of multiagent groups in social networked systems. *IEEE Internet of Things Journal*, 9(14):12194–12208, 2021.

[32] Kathryn Macarthur, Ruben Stranders, Sarvapali Ramchurn, and Nicholas Jennings. A distributed anytime algorithm for dynamic task allocation in multiagent systems. *AAAI Conf. on Artificial Intelligence*, 25(1):701–706, 2011.

[33] Mojtaba Malek Akhlagh and Jernej Polajnar. Distributed deliberation on direct help in agent teamwork. In *Proceedings of the 12th European Conference on Multi-Agent Systems (EUMAS 2014)*, Prague, Czech Republic, December 2014.

[34] Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. Jgrapht—a java library for graph data structures and algorithms. *ACM Trans. Math. Softw.*, 46(2), May 2020.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[37] Narek Nalbandyan, Jernej Polajnar, Denish Mumbaiwala, Desanka Polajnar, and Omid Alemi. Requester vs. helper-initiated protocols for mutual assistance in agent teamwork. In *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC'13)*, pages 2741–2746, Manchester, UK, October 2013.

[38] Antoine Nongaillard and Philippe Mathieu. A multi-agent resource negotiation for social welfare. In *2009 IEEE/WIC/ACM International Joint Conference on*

Web Intelligence and Intelligent Agent Technology, volume 2, pages 58–61. IEEE, 2009.

[39] Antoine Nongaillard, Philippe Mathieu, and Brigitte Jaumard. A realistic approach to solve the nash welfare. In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, pages 374–382. Springer, 2009.

[40] Jernej Polajnar, Mojtaba Malek Akhlagh, Narek Nalbandyan, Denish Mumbaiwala, and Desanka Polajnar. Decentralized reactive adjustment of agent teamwork organization to changing environment. In *Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC'14)*, pages 1457–1462, San Diego, California, October 2014.

[41] Jernej Polajnar, Narek Nalbandyan, Omid Alemi, and Desanka Polajnar. An interaction protocol for mutual assistance in agent teamwork. In *Proceedings of the 6th International Conference on Complex, Interactive, and Software-Intensive Systems (CISIS 2012)*, pages 6–11, Palermo, Italy, July 2012.

[42] Faezeh Rahimzadeh, Leyli Mohammad Khanli, and Farnaz Mahan. High reliable and efficient task allocation in networked multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 29:1023–1040, 2015.

[43] Donya Rahmani. Designing a robust and dynamic network for the emergency blood supply chain with the risk of disruptions. *Annals of Operations Research*, 283:613–641, 2019.

[44] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *AAAI*, volume 93, pages 256–262, 1993.

[45] Tuomas W Sandholm and Victor R Lesser. Leveled commitment contracts and strategic breach. *Games and Economic Behavior*, 35(1-2):212–270, 2001.

[46] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organization in multi-agent systems. *The Knowledge engineering review*, 20(2):165–189, 2005.

[47] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1-2):165–200, 1998.

[48] Liat Sless, Noam Hazon, Sarit Kraus, and Michael Wooldridge. Forming k coalitions and facilitating relationships in social networks. *Artificial Intelligence*, 259:217–245, 2018.

[49] Liat Sless, Noam Hazon, Sarit Kraus, and Michael J Wooldridge. Forming coalitions and facilitating relationships for completing tasks in social networks. In *AAMAS*, pages 261–268. Citeseer, 2014.

[50] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. on Computers*, 29:1104–1113, 1980.

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

[52] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[53] Wanyuan Wang and Yichuan Jiang. Community-aware task allocation for social networked multiagent systems. *IEEE transactions on cybernetics*, 44(9):1529–1543, 2013.

[54] Wanyuan Wang and Yichuan Jiang. Multiagent-based allocation of complex tasks in social networks. *IEEE Transactions on Emerging Topics in Computing*, 3(4):571–584, 2015.

[55] Duncan Watts and Steven Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[56] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[57] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[58] Guoli Yang and Vincent Danos. Learning in open adaptive networks. In *2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 50–59. IEEE, 2016.

[59] Dayong Ye, Minjie Zhang, and Athanasios V Vasilakos. A survey of self-organization mechanisms in multiagent systems. *IEEE Trans. on Systems, Man, and Cybernetics: Systems*, 47(3):441–461, 2016.

[60] Chongjie Zhang, Victor Lesser, and Prashant Shenoy. A multi-agent learning approach to online distributed resource allocation. In *Twenty-first Int. Joint Conf. on Artificial Intelligence*, 2009.

Appendix

Appendix A

Notational Conventions

Table A.1: List of symbols used in algorithms

Symbol	Description
a	agent
a_i	agent i
a_p	provider agent
a_t	agent with task t
$\mathtt{a_t}$	Markov action in step $\mathtt{t}$
A	set of agents
A_{req}	set of request recipients
cst_{off}	offer cost function
$cost_{trf}$	cost of transfer
D	replay memory
dst	distance function
e	episode
itm_{cnf}	confirmed resource items
itm_{off}	offered resource items
loc	task location function
N	capacity of D
q_{alc}	allocated quantity
q_{avl}	available quantity
q_{cnf}	confirmed quantity
q_{off}	offered quantity

Continued on next page

Table A.1 continued from previous page

Symbol	Description
q_{req}	requested quantity
q_{rqr}	required quantity
q_{trf}	transferred quantity
$\mathtt{Q}$	policy action-value function
$\mathtt{Q}'$	target action-value function
r	resource type
rqr	task requirement function
R	set of resource types
rsc	individual agent resource function
$\mathtt{r_t}$	Markov reward in step $\mathtt{t}$
$\mathtt{s_t}$	Markov state in step $\mathtt{t}$
SW	social welfare
$\mathtt{t}$	step
t	task
T	set of tasks
T_e	set of tasks in current episode
tcs	transfer cost function
$\mathcal{N}_{a_i}$	neighbors of a_i
τ_{off}	offer timeout
τ_{req}	request timeout
τ^o_{req}	original request timeout
θ	weights of policy network
θ'	weights of target network
trc	total (all agents) resource function
tsk	task function
utl_{req}	request utility function
$util_t$	utility of t