PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

TÚLIO LIMA BASÉGIO

# DECENTRALISED ALLOCATION OF STRUCTURED TASKS IN HETEROGENEOUS AGENT TEAMS

Porto Alegre

2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*


Pontifícia Universidade Católica
do Rio Grande do Sul

**PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL**
**SCHOOL OF TECHNOLOGY**
**GRADUATE PROGRAM ON COMPUTER SCIENCE**

# DECENTRALISED ALLOCATION OF STRUCTURED TASKS IN HETEROGENEOUS AGENT TEAMS

**TÚLIO LIMA BASÉGIO**

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Advisor: Prof. Dr. Rafael Heitor Bordini

**Porto Alegre**
**2018**

## Ficha Catalográfica

B327d Baségio, Túlio Lima

Decentralised allocation of structured tasks in heterogeneous agent teams / Túlio Lima Baségio . – 2018.
143 p.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rafael Heitor Bordini.

1. multi-agent system. 2. multi-robot system. 3. coordination. 4. task allocation. I. Bordini, Rafael Heitor. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Túlio Lima Baségio

# Decentralised allocation of structured tasks in heterogeneous agent teams

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on August 24, 2018

**Committee Members:**

Prof. Dr. Luís Alvaro de Lima Silva (UFSM)

Prof. Dra. Diana Francisca Adamatti (FURG)

Prof. Dra. Renata Vieira (PPGCC/PUCRS)

Prof. Dr. Rafael Heitor Bordini (PPGCC/PUCRS – Advisor)

“Attitude is a little thing that makes a big difference.”
(Winston Churchill)

## ACKNOWLEDGMENTS

First I would like to express my deepest gratitude to my advisor, Dr. Rafael Heitor Bordini, for his support, guidance and belief in me. I would like to thank you people from the SMART research group who share the lab with me while I was doing the PhD. I would like to thank the Instituto Federal do Rio Grande do Sul (IFRS) for supporting and partially funding my academic research. Lastly, I would like to thank my friends, family, and especially, my amazing wife, for believing in me and supporting me even in the most challenging times.

# DECENTRALISED ALLOCATION OF STRUCTURED TASKS IN HETEROGENEOUS AGENT TEAMS


## RESUMO

Sistemas multiagentes permitem o desenvolvimento de soluções flexíveis e robustas e têm sido utilizados há vários anos na academia e na indústria para projetar e implementar sistemas distribuídos complexos em vários domínios. No entanto, ainda há desafios no desenvolvimento de estratégias apropriadas para que times de agentes operem de maneira eficiente. Um aspecto crítico é a coordenação entre os agentes, que, apesar dos esforços dos pesquisadores, ainda hoje é um desafio. Agentes precisam se coordenar para alcançar objetivos que não conseguem realizar sozinhos, devido à falta de conhecimento sobre o mundo ou por qualquer outro motivo, como recursos limitados ou distância espacial. Na robótica, sistemas com múltiplos robôs também carecem de complexos métodos de coordenação, sem os quais se torna impossível construir verdadeiros times de robôs. Existem diferentes abordagens propostas para a coordenação em sistemas multiagentes e em sistemas multi-robôs, dentre as quais muitas lidam diretamente com o problema de alocação de tarefas. De fato, a alocação de tarefas é uma importante área de pesquisa quando se lida com o problema de coordenar grupos de agentes ou robôs. Além disso, cenários do mundo real geralmente requerem o uso de entidades heterogêneas e a execução de tarefas com estruturas e complexidades diferentes. Assim, é necessário desenvolver métodos que permitam projetar e implantar aspectos relacionados a alocação de tarefas tornando os sistemas cada vez mais eficientes. Considerando essa necessidade, apresentamos um mecanismo decentralizado para a alocação de diferentes tipos de tarefas entre múltiplos agentes heterogêneos que desempenham papéis e executam tarefas de acordo com suas capacidades. A avaliação do nosso mecanismo de alocação de tarefas foi realizada através de várias simulações. Também avaliamos nosso mecanismo em uma simulação com tarefas relacionadas ao cenário de busca e resgate em desastres naturais por inundação, onde vários robôs autônomos podem ser empregados para apoiar a equipe de resgate. Os resultados mostram que o mecanismo proposto fornece alocações próximas ao resultado ótimo.



**Palavras-Chave:** sistemas multiagentes, sistemas multi-robôs, coordenação, alocação de tarefas.

# DECENTRALISED ALLOCATION OF STRUCTURED TASKS IN HETEROGENEOUS AGENT TEAMS


## ABSTRACT

Multi-agent systems allow the development of flexible and robust solutions and have been used for several years in academia and industry to design and implement complex distributed systems in various domains. However, there are many challenges in developing appropriate strategies for multi-agent teams so that they operate efficiently. One critical aspect is the coordination between agents, which despite much research effort is still a challenge. Agents need to coordinate to achieve goals that, for whatever reason, cannot be accomplished alone, due to the lack of knowledge about the world or for any other reason, such as limited resources and spatial distance. In robotics, systems with multiple robots also require complex coordination methods, without which it is impossible to build real robotic teams. There are many approaches proposed in the literature for MAS and multi-robot system coordination, many of them directly related to task allocation problems. In fact, task allocation is an important research area in dealing with the problem of coordinating a group of agents or robots. Besides that, real-world scenarios usually require the use of heterogeneous entities and the execution of tasks with different structures and complexities. Thus, it is necessary to develop further methods to support the design and implementation of aspects related to task allocation. Taking that into account, we present a decentralised task allocation mechanism considering different types of tasks for heterogeneous agent teams where agents play different roles and carry out tasks according to their own capabilities. We have run several experiments in order to evaluate the proposed mechanism. We also evaluate our task allocation mechanism in a simulation with tasks related to the search and rescue scenario in natural disaster by flooding where multiple autonomous robots can be employed to support human rescuers. The results show that the proposed mechanism provides near-optimal allocations.



**Keywords:** multi-agent system, multi-robot system, coordination, task allocation.

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

A&A – Agents and Artifacts

AI – Artificial Intelligence

BDI – Belief-Desire-Intention

CDM – Centre for Disaster Management

CM – Compound for Many

CN – Compound for Exactly N

DCOP – Distributed Constraint Optimization Problems

DEC-POMDPS – Decentralised POMDPs

DS – Decomposable Simple Task

EA – Entirely Allocated

EIS – Environment Interface Standard

GLPK – GNU Linear Programming Kit

ICBAA – Iterative Consensus-Based Auction Algorithm

MAPC – Multi-Agent Programming Contest

MAS – Multi-Agent Systems

MASSIM – Multi-Agent Programming Simulation platform

MATA – Multi-Agent Task Allocation

MCTS – Monte Carlo Tree Search

MDP – Markov Decision Process

NA – Completely Unallocated

PA – Partially Allocated

POMDPS – Partially Observable MDPs

PRS – Procedural Reasoning System

RBTA – Role-based Task Allocation

SSIA – Sequential Single-Item Auction Algorithm

TAA – Task Allocation Algorithm

UAV – Unmanned Aerial Vehicle

UGV – Unmanned Ground Vehicle

USV – Unmanned Surface Vehicle

# CONTENTS

# 1. INTRODUCTION

Multi-agent systems (MASs) have been used for many years in academia and industry to design and implement complex systems in different areas such as manufacturing, process control, telecommunication systems, air traffic control, among others [LMP03]. A MAS consists mainly of entities called software agents, which act in an environment and interact with each other to solve a problem. Russel and Norvig in [RN09] define an agent as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators". Based on these definitions, it is possible to highlight some essential characteristics of a software agent, such as autonomy, intelligence and acting in an environment. Considering these individual characteristics, MASs allow the development of flexible and robust solutions. However, it is necessary to coordinate the agents so that the system executes efficiently [Par14].

In fact, coordination is an essential aspect of MASs [VSL+10]. In [HS99], Huhns and coauthors define coordination as "a property of a system of agents performing some activity in a shared environment". According to Wooldridge in [Woo09], coordinating means "managing interdependencies between agents' activities" to achieve a goal. In practical terms, to coordinate means managing the execution of the activities to avoid that, for example, agents interfere with/impact negatively on the achievement of the objectives of other agents. Coordination is related to the social skills of agents, where agents communicate not only to share data, but communicate at the knowledge level (which would be the ability to communicate their beliefs, goals, and plans to another agent) [BHW07]. With coordination, agents can achieve joint objectives and plans that otherwise might not be possible, thus increasing the problem-solving capacity of a MAS; it is also possible to ensure that agents coherently and efficiently perform their tasks, synchronising their actions and interactions with other agents [BBO+02, HS99].

Multi-agent environments can be cooperative or competitive [RN09]. In cooperative environments, agents usually have joint goals and plans, where there are actions defined for each agent [RN09]. In this case, the agents form a kind of team to achieve the goals. In competitive environments, there is no commitment to a joint plan, since agents maintain competition with each other [RN09]. Both cooperation and competition are related to the agents' ability to communicate [HS99, BBO+02, YJC13].

Coordination is an important aspect also in the Robotics research area. Nowadays, much of the research in the robotics area has discussed solutions related to the use of multi-robot systems [PGCM+13, YJC13]. Although a system based on a single robot can perform a series of tasks, other tasks can only be executed by multiple robots, for example because of inherent physical distribution.

The Agents paradigm has been applied for several years in the area of robotics [BA95, FRD98, DC98, ILS07, WdS08]. In fact, it is possible to make associations between

the characteristics of an individual agent and a robot as well as between the characteristics of an MAS and those of a multi-robot system, which allows the use of concepts and techniques from the area of agents to manipulate and control robots in simple systems and systems composed of multiple robots. Associations are also possible between the challenges of the two areas: according to [YJC13], one of the challenges in developing multi-robot systems is precisely the design of coordination strategies in such a way that robots perform their operations efficiently.

Yan and co-authors suggest that creating coordination strategies that enable agents to perform tasks efficiently is not a trivial task [YJC13]. Without such strategies, the use of multi-robot systems in complex scenarios such as rescue operations after a natural disaster becomes limited or even unfeasible. More generally, the same applies to multi-agent systems.

Also, there are new lines of research in MASs, such as open systems, which add even more complexity to coordination aspects, since in open systems it may be necessary to coordinate heterogeneous agents [BBO+02] into a group of dynamically changing agents. The need to coordinate heterogeneous entities is also a challenge in multi-robot systems since they can be composed of robots with different types of mobility, sensors, physical and computational capabilities.

In the recent literature, there are various approaches proposed for the coordination of agents in MASs [DS08, XLL12, YWN10] as well as multi-robot systems [SUIM10, CSVJRC14, Hui10]. Regardless of the characteristics of the proposed coordination approach, an important aspect considered in coordination problems is task allocation [LCS13, Cor14, FUMP13, LTL+15, LCS15a, SGSTS15, SWWA14, YJC13].

There are several features that should be considered by a mechanism for allocating tasks to multiple robots or agents so that it can be used in real-world scenarios. In fact, the literature mentions different characteristics of allocation mechanisms, such as decentralised decision making, the possibility of using heterogeneous entities (agents or robots with different physical and computational capacities, e.g., different sensors and types of mobility), the impact of individual variability to assign specific roles to individual entities, the definition and allocation of different types of tasks, among others.

However, available solutions consider only a small set of desirable features in the allocation mechanisms. Although many real-world scenarios, such as disaster rescue, for example, typically require the use of heterogeneous robots and the execution of tasks with different complexities and structures, most of the solutions in the literature only deal with the allocation of one type of task, mainly atomic tasks. The approaches that deal with more complex tasks usually focus on one type of constraint only, and the ones that deal with more features commonly have high computational cost. Using the solutions available in an integrated way is not always possible due to the different assumptions and architectural requirements used by different authors.

The main contribution of our work is a decentralised mechanism for the allocation of different types of tasks to heterogeneous agent teams, considering that they can play different roles during a mission and carry out tasks according to their own capabilities, which is particularly important for applications in multi-robot systems. The proposed mechanism was initially inspired by [LCS15a], but it is significantly different since we are working with different types of tasks, considering the use of roles, and verification of constraints related to the heterogeneity of robots.

Some of the ideas presented in this thesis first appeared in [BB17], but here we provide details about the method and algorithms, we provide new results from the comparison with optimal solutions, and we introduce a comparison with other decentralised approaches. We also detail the way in which we determine the end of the allocation process.

In this thesis, we use a flood disaster scenario throughout the text to exemplify our approach. Flooding disasters are typically very dynamic and have complex tasks to be executed [SKV+12]. In fact, it was the typical tasks in flooding rescue that inspired us to work on a task allocation mechanism that could address different types of tasks. During a rescue phase in a flooding disaster, teams are called into action to work on tasks such as locating and rescuing victims [MTN+08]. Such teams are typically organised by a hierarchy model [RHI+15], with individuals playing different roles during a mission. The execution of tasks during the rescue stage poses many risks to the teams. Using robots in a coordinated way to help the team may minimise such risks. In our running example, the term *agent* refers to the main control software of an individual *robot*, so we use both terms interchangeably.

## 1.1 Motivation

There is extensive research that deals with coordination aspects related to the allocation of tasks in MASs or multi-robot systems. In fact, researchers have been making significant progress proposing techniques, approaches, applications, platforms, and frameworks for the allocation of tasks to multiple entities, whether software agents or robots.

However, there are still many issues that deserve the attention of the scientific community to develop entities capable of allocating tasks in a distributed and dynamic way while achieving efficient solutions in complex real world environments. This view is corroborated by researchers such as Yan and coauthors [YJC13], who indicate that several problems related to multi-robot coordination have not yet been fully solved, among them problems related to task allocation.

Besides, several scenarios and applications would benefit from a more efficient task allocation solution. Also, there are scenarios where centralised solutions may not be appropriate because they represent a single point of failure for the entire system, and suffer from the communication overhead required during the allocation process. In this way, decentralised solutions for the allocation of tasks are desirable.

Also, scenarios closer to the real world require the execution of more complex tasks where, for example, a task can be composed of subtasks. Most of the solutions found in the literature deal only with the allocation of atomic tasks (simple), and those that deal with more complex tasks, for the most part, focus only on one type of task and one type of constraint.

## 1.2 Objectives

The main objective of this thesis is to develop a decentralised mechanism for the allocation of different types of tasks in systems with multiple and heterogeneous agents that can play different roles. To accomplish this objective, we defined the following specific objectives:

- Design a decentralised mechanism for allocating different types of tasks in systems with multiple and heterogeneous agents that can play different roles;
- Implement the mechanism by integrating it with a MAS development platform (JaCaMo);
- Evaluate the proposed mechanism against the optimal solution and other solutions appearing in the literature;
- Design and implement a simulation environment to evaluate the impact of the mechanism on the execution of tasks.
- Evaluate the impact of the proposed mechanism and compare it with other solutions on the execution of tasks in the simulation environment.

## 1.3 Thesis Organisation

The thesis is organised as follows: Chapter 2 provides a theoretical background about the main concepts addressed in this work. More specifically, we present concepts about intelligent agents and multi-agent systems, cooperation and coordination in multi-agent systems, task allocation, and programming multi-agent systems. In Chapter 3, we describe the type of tasks considered in this thesis, provide a scenario example, as

well as we formalise our task allocation problem. Chapter 4 introduces the proposed mechanism for allocating tasks among multiple and heterogeneous agents. Chapter 5 presents the experiments performed to evaluate our mechanism by comparing it with the optimal solution and other decentralised approaches. In Chapter 6 we present the work most closely related to this thesis. Finally, this thesis concludes in Chapter 7, with a summary of our contributions and discussion of future work.

# 2. BACKGROUND

This chapter presents a brief review of important concepts for a better understanding of this work. More specifically, we introduce concepts about intelligent agents and MASs, cooperation and coordination in multi-agent systems, task allocation and multi-agent programming. As mentioned in Chapter 1, it is possible to make associations between the characteristics of a MAS and those of a multi-robot system, which allows the use of concepts and techniques in the area of agents to manipulate and control robots in systems composed of multiple robots. Thus, throughout this thesis, we present concepts focusing on both agents and robots because we understand that both contribute to the research.

## 2.1 Intelligent Agents and Multiagent Systems

Russel and Norvig define an agent (whether it is a software or not) as "anything that can be considered capable of perceiving its environment by means of sensors and of acting on that environment by means of actuators" [RN09]. In [Woo09] the following definition for an agent is provided: "An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its delegated objectives" [Woo09]. Figure 2.1 presents an abstract representation of these definitions, in which an agent perceives the environment and acts on it.

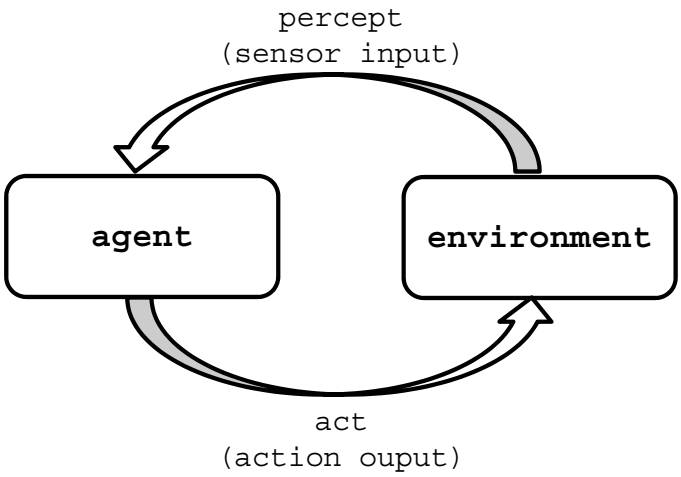


Figure 2.1 – Abstract view of the relationship of an agent with the environment.

Based on these definitions, it is possible to highlight some essential characteristics of a software agent, such as autonomy, intelligence and acting in an environment. Concerning intelligence, in [WJ95] the authors define a list of characteristics that an

agent must present to be considered intelligent. Such characteristics are used by agents to achieve their goals. The following are considered characteristics of an intelligent agent, according to [WJ95]:

- **Reactivity:** refers to the agent's ability to perceive and react to changes in the environment to satisfy its objectives. It means that when changes in the environment prevent the agent from continuing to execute a plan, the agent must react to that change and choose an alternative course of action.

- **Proactivity:** agents exhibit goal-oriented behaviour and take the initiative of actions. Thus, it is expected that when an agent has a goal, it proactively try to achieve that goal.

- **Social skills:** ability to communicate and interact with other agents. This is a desirable characteristic, for example, for the agents to cooperate and to coordinate their actions.

These characteristics are valuable when we think in a decentralised task allocation mechanism. For example, when an agent perceives new tasks in the environment, it can proactively communicate with other agents which may react trying to allocate the tasks they would like to perform. Social skills are also crucial since the agents need to communicate to negotiate during the allocation.

Agents can be defined based on different architectural patterns. An architecture widely used for modelling agents is called Belief-Desire-Intention (BDI) [BIP88, RG95]. According to [Woo13], the BDI architecture aims to represent practical reasoning, more precisely the process of deciding which are the relevant goals and how the agent will achieve them. The basis of the BDI architecture are beliefs, which represent the agent's knowledge about the environment in which it acts, its desires, which are goals that the agent would like to achieve, and the intentions that are the goals that the agent decided in fact achieve. Figure 2.2 shows a generic BDI architecture.

The *belief revision* function receives input from the sensors and based on the current beliefs in the *beliefs base* it defines if the beliefs must be updated. The *generate options* function uses the current beliefs and intentions of the agent to determine new options (*desires*) to the agent. These options represent possible courses of actions for the agent. The process of determining the *intentions* of the agent based on the current beliefs, desires and intentions is performed by the *filter* function. Finally, an *action selection* function determines which intention should be performed as an action. This process represents how the practical reasoning works in a BDI agent.

When several intelligent agents act in a shared environment and interact with each other, it is said that there is a Multiagent System (MAS). According to Wooldridge

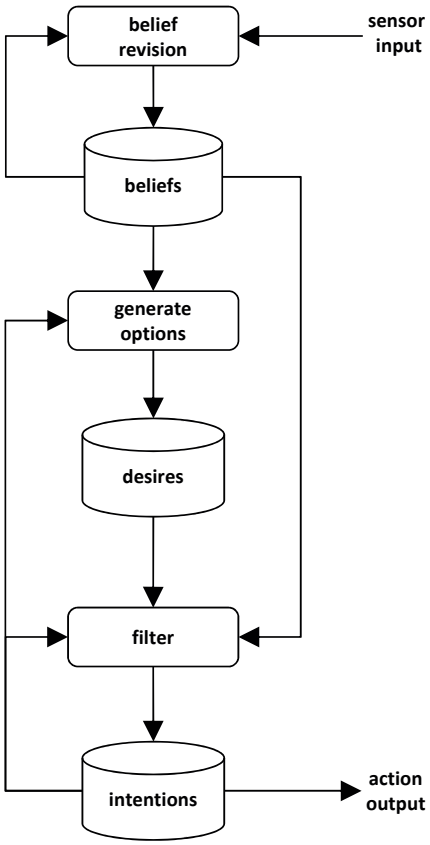


Figure 2.2 – Generic BDI architecture - adapted from [Woo13]

in [Woo09], "Multiagent systems are systems composed of multiple interacting computing elements, known as agents". According to Bordini and coauthors in [BHW07], the possibility of sharing the environment among several agents in a MAS makes the system as a whole more complex, since the actions of an agent can influence the perceptions and actions of other agents.

Russell and Norvig in [RN09] present a set of characteristics that help define the complexity of an environment in a MAS:

- **Fully observable or partially observable:** fully observable is the environment where the agent can at any time obtain complete, accurate and up-to-date information about the environment. Most real environments are partially observable.

- **Deterministic or stochastic:** if for any action in the environment there is no uncertainty about its result and there is a single guaranteed effect, then the environment is deterministic. Otherwise, the environment is said stochastic.

- **Static or dynamic:** if the environment is changed only by the agent's actions, then it is static. If other agents and processes can change it, it is said to be dynamic. The real world is typically dynamic.

- **Discrete or continuous:** if there is a fixed and finite number of actions and perceptions in the environment, it is discrete. Otherwise, the environment is said to be continuous.

According to the authors, more complex environments are those partially observable, nondeterministic, dynamic and continuous [RN09]. In a MAS, agents can be part of one or more organisations or eventually work individually (which is not typical). An agent in an organisation may be related to its peers or other agents in an authority/hierarchy structure [BHW07]. Agents in an organisation tend to cooperate to achieve organisational goals that would not be possible individually, either because of lack of capacity or knowledge [Woo13, DP13].

Multi-agent systems (MASs) can be applied to design and implement complex systems in different areas such as manufacturing, process control, telecommunication systems, air traffic control, among others [LMP03]. Regardless domain, an essential property of agents operating in a shared environment is coordination, especially when agents act cooperatively [HS99]. Next, we provide an overview of the process of coordinating a group of agents.

## 2.2 Coordination in Multiagent Systems

In cooperative environments, agents typically have joint goals and plans [RN09], and in such cases, agents form a kind of team to achieve a common goal. In situations where agents need to cooperate, interaction is usually required to achieve the objectives and increase the overall utility of the system [YJC13]. Cooperation is an essential aspect because sometimes a single agent does not have sufficient capacity, resources or information to solve a problem individually [Woo09]. As agents must act autonomously, they must also be able to coordinate their activities by cooperating with each other dynamically.

According to Wooldridge in [Woo09], coordinating means "managing interdependencies between agents' activities" to achieve a goal. These interdependencies are related to the interaction between activities. Two activities can interact whether, for example, they try to use the same resource at the same time, or whether one activity depends on the completion of another [Woo09]. In practical terms, to coordinate means managing the execution of the activities to avoid that, for example, agents interfere with-/impact negatively on the achievement of the objectives of other agents. Coordination techniques can be used to decrease or rid a system of conflicts.

In the recent literature, there are a variety of solutions proposed for the coordination of agents in a MAS. The available approaches can usually be classified according to the type of solution offered (whether centralised or decentralised), by the kind of

agents (or robots) considered (whether homogeneous or heterogeneous), by the type of coordination strategy used, among others.

Examples of centralised coordination approaches are presented in [DS08, BBT02, VDBSLM10, SSFS12]. Centralised approaches have the disadvantage of having a point of failure that compromises the whole system, which does not occur in decentralised approaches. Examples of decentralised approaches are presented in [dCFS13, Hui10]. In [dCFS13], for example, an approach is proposed for autonomous and cooperative vehicles to negotiate their speeds to pass through the crossing safely, avoiding collisions. Similarly, [Hui10] considers the need for negotiation to avoid collisions in multi-agent moving environments, also in a decentralised manner.

According to [YWN10], considering agent heterogeneity is a challenging issue in coordinating SMAs. In [WGL14] a solution is presented that aims to improve the coordination efficiency in rescue scenarios with heterogeneous agents. The approach proposed by the authors focuses on the partitioning of the search region for the allocation of agents in different positions. A partitioning algorithm divides the map into some partitions based on the number of agents and buildings, and then each agent assigns a partition to itself.

Different coordination strategies are also available in the literature. One of these strategies is based on consensus algorithms. [CFNM15], for example, deals with the problem of coordinating the movements of a set of aerial robots employing a decentralised algorithm based on the consensus theory, where agents exchange information between them and each agent generates its own trajectory, obtaining the desired formation in a coordinated way.

[YWN10] presents a biological systems approach that allows distributed agents to arrive at a group decision (or global consensus) using what they call "implicit leadership." The authors extend a distributed consensus algorithm to deal with cases where agents may have varying degrees of information. The proposed approach is based on local rules that allow agents to reach consensus. In [CGP15], the focus is on vehicle formation control in the presence of communication delays and a consensus algorithm is also used.

Another coordination strategy is called leader-follower. In [CSVJRC14] is presented a decentralised solution for coordination of aerial vehicles using the leader-follower strategy. The leader is the only one who knows the desired trajectory. The leader communicates its position to the first follower, which communicate to the next follower and so on. Another approach to formation control using the leader-follower strategy is found in [AAT13], where the authors use an air vehicle as a leader and land vehicles as followers. The aerial vehicle follows a predefined path, and the ground vehicles must make a formation (square) around the position of the aerial vehicle. In [ZCMZ13] the focus is on the pursuit of fugitives by a group of robots, where the formation change

is performed using a leader-follower strategy. In this work, the evolution of formation of the followers occurs with the movement variation of the fugitives using "Markovian Partially Observable Decentralized Decision Processes".

A coordination strategy different from those presented above is proposed in [SUIM10]. In this paper, the authors combine implicit coordination with belief sharing. Coordination is said to be implicit since the utilities of the robots for a task are not shared, but calculated locally. Robots initially share their beliefs with others and then use predictive models to calculate locally the utilities of each robot based on the beliefs received. The approach is applied to a team of robots in a football scenario. The focus of the work presented in [ZL14] are situations where humans work alongside robots to perform tasks, where long hours of work can cause fatigue in humans. The authors integrate a model that considers human fatigue in a mixed coordination framework (robot-human) to predict team performance.

Regardless of the coordination approach, one important aspect considered in coordination problems is the allocation of tasks [YJC13]. Task allocation is an important research area when dealing with the problem of coordinating groups of autonomous entities [SGSTS15, SWWA14, LTL+15]. In fact, several authors consider task allocation as an essential part of the coordination problems of multiple robots or agents [LCS13, FUMP13, Cor14, LCS15a]. For example, the work presented in [NS14] focuses on the use of ontologies for the allocation of tasks, aiming to coordinate the work between a team of autonomous agents. [ZMY+14] presents a distributed solution to the task allocation problem among multiple robots considering the workload balancing between them. More details about task allocation are provided in the next section.

## 2.3 Multi-Agent Task Allocation

Task allocation among multiple robots and more generally among multiple agents consists of identifying which agents should perform which tasks in order to achieve cooperatively as many global goals as possible and in the best possible way. Previous to the definition of our Multi-Agent Task Allocation (MATA) problem, it is important to review some relevant concepts for the allocation process (such as task, constraint, and utility) and in particular how they are treated in this work.

### 2.3.1 Constraint

A constraint restricts the possibilities of allocation, reducing, consequently, the number of possible solutions but also making allocation more complex given one must

ensure that any candidate solution satisfies all constraints. Constraints may be related to tasks, agent's capacity, capabilities of the agents, among others.

Task constraints, for example, may indicate that some subset of the tasks must be carried out concurrently [KSD13], or by a single robot. For example, consider a company which uses autonomous robots to perform tasks such as buy and delivery items. The company needs to buy an item in a location A and also needs to deliver another item in a location B which is very close to location A. In this case, it is quite possible that the company will impose a constraint that both tasks should be performed by the same robot. Capacity constraints may limit how many tasks a robot can allocate at a time. This constraint may be related, for example, to the amount of energy (fuel) available to a robot. So, for the example above, a robot may not be able to take those tasks because it does not have enough energy (capacity) to perform both tasks. Capability constraints may restrict some robots from performing specific tasks. For example, consider a task for which the robot needs to go to a location and take a picture. If a robot does not have a camera, it is not capable of performing that task.

### 2.3.2 Utility

According to Korsah and coauthors in [KSD13], task allocation solutions aim to find the allocation of tasks that optimise some objective and, for that, it is necessary to determine what should be optimised. Many mechanisms have the objective of maximising or minimise the utility value.

A utility is a value that expresses how much a task contributes to the robot's objectives when executed by it. The utility for each task is quantified through a function which can combine several factors (e.g., the quality with which a particular robot is likely to accomplish that task, how quickly that is likely to be done, and so forth) [GM04]. The global team utility can be quantified as a combination of the individual utilities.

The utility values must be scalable and possible to be compared with so that one can establish the best tasks for each robot and the best robot for each task. For example, if a robot $r_1$ estimates utility values 10 and 20 for the tasks $t_1$ and $t_2$ respectively, the utilities can be compared and the robot $r_1$ may have the preference for task $t_2$ instead of $t_1$ since it has a higher utility (if the objective is to maximise its utility). In the other hand, if a robot $r_2$ estimates 30 as the utility value for $t_2$, it will probably have preference over $r_1$ on allocating the task $t_2$.

To calculate the utility value of a task, relevant aspects of the robot, of the environment, and the actual task requirements should be taken into consideration [GM04]. In many cases, the utility calculation can be carried out independently for each task. In other cases, it may be necessary to consider other tasks previously allocated to the agent for the utility calculation [KSD13]. In this work, we assume that only the robot itself is

able to determine its utility for a given task accurately; there is no way to compute in a centralised way the utility functions for all the robots, hence the importance of a decentralised approach to task allocation. Note that, as will be specified in the Section 3.3, in this thesis the robots are allocated to subtasks and not directly to tasks, so the utility is calculated to the subtasks.

### 2.3.3 Tasks

Agents in a MAS can help in the accomplishment of tasks, and for this, it is necessary to specify what tasks are required to reach the global objectives of the system [Woo09]. According to Gerkey and Matarić in [GM04], each task is defined to solve part of the problem/goal of the system and may have different characteristics and vary regarding complexity.

Although many real-world scenarios typically require the execution of tasks with different complexities and structures, most of the task allocation solutions in the literature assume that tasks are independent atomic units. A task is atomic if it cannot be decomposed into subtasks, that is, it can only be carried out by a single agent [Zlo06]. Examples of solutions considering only atomic tasks can be found in [GA13, LMB+13, SP13, GSW+14, SGSTS14, CsTPcP14, LZK15, TMS15, SGSTS15]. In general, a task is considered atomic or not, depending on the level of the domain model being used. A task can also be defined as non-atomic. Non-atomic tasks are not tasks composed of subtasks. Non-atomic are tasks that can be partially executed, at different times and by different robots, until they are completed [FGC14, SZB16, WLL+16, IF16, MVDB17].

A task can also be defined as a compound task, that is, a task that can be decomposed into a set of atomic subtasks. Examples of solutions considering tasks composed by subtasks can be found in [DMC14, AHG17, MSG+17, LZ03]. For [Zlo06] a compound task can be decomposed into a set of atomic or compound subtasks where each subtask can be allocated to a different agent.

According to [Zlo06] a task can also be defined as a decomposable simple task, that is, a task that can be decomposed into a set of atomic subtasks or other decomposable simple tasks as long as the different decomposed parts cannot be carried out by more than one agent. In Section 3.1 we describe the type of tasks considered in this thesis.

### 2.3.4 Multi-Robot Task Allocation

The most basic task allocation problem addressed in the robotics and also in the agents literature is the one-to-one assignment [LCS15a]. This task allocation problem

can be defined as a set of $n_r$ robots $R = \{r_1, \ldots, r_{n_r}\}$ and a set of $n_t$ tasks $T = \{t_1, \ldots, t_{n_t}\}$ with $n_r = n_t$, where each task $t_j$ may be allocated to at most one robot (equation 2.1), and each robot $r_i$ can perform at most one task (equation 2.2). Considering a binary variable $f_{ij}$ indicating whether $t_j$ is assigned to $r_i$, and $u_{ij}$ as the utility value associated for the allocation of $t_j$ to $r_i$, the objective is to find an allocation that maximises the sum of utilities of the agents, i.e., the total utility (equation 2.3). The optimisation problem above can be stated as follows:

$$\sum_{i=1}^{n_r} f_{ij} = 1 \;\; \forall j = 1, \ldots, n_t. \tag{2.1}$$

$$\sum_{j=1}^{n_t} f_{ij} = 1 \;\; \forall i = 1, \ldots, n_r. \tag{2.2}$$

*Objective:*

$$\max \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} u_{ij}.f_{ij} \tag{2.3}$$

In this thesis, we extend the basic problem in some aspects. First of all, we consider that each robot has a maximum number of tasks that can allocate to itself rather than only one. This constraint may be related, for example, to the amount of energy (fuel) available to a robot thus limiting the number of tasks it can be assigned. It means that we account for not only the number of allocated tasks but also that tasks can occupy different proportions within the limit of tasks that an agent can allocate. Also, the basic problem assumes that tasks are independent atomic units. We here consider that tasks can be composed of subtasks, with different restrictions on how the subtasks are assigned to different robots. Our approach also considers robots with different capabilities which may restrict some robots from performing specific tasks. Since our approach can be used more generally than only in multi-robotics, we often use the term agent instead of robot.

### 2.3.5 Examples of approaches used for Task Allocation

The approaches for task allocation range from solutions using market-based approaches, DCOP approaches, centralised and decentralised solutions, among others. Some of these approaches will be briefly described below. Surveys about task allocation can be found in [JM13, LHK13, Jia16].

**Centralised and decentralised task allocation:** an important aspect when working with task allocation is to decide whether the approach used will be centralised or decentralised. In a centralised approach, a single point is responsible for allocating tasks between agents, while in a decentralised approach the agents can communicate and decide between them the best allocation of tasks. Some examples of centralised approaches can be found in [DMC14, FGC14]. Examples of decentralised approaches can be found in [SGSTS15, LCS15a, SP13]. Generally, centralised approaches perform faster than decentralised approaches [GM04, WMC15] and may obtain better results because they have all the information about all agents [TMS15]. A disadvantage of centralised approaches is to have a single point of failure for the system [SGSTS15, SWWA14, WMC15]. Another drawback of centralised approaches is a possible communication overload [GM04, SGSTS15, WMC15, TMS15, SWWA14], mainly in the central entity, since agents need to communicate all information about themselves and the environment to this central entity [SGSTS15, TMS15]. Such overloading occurs, especially in large-scale systems [TMS15]. Distributed approaches can be used to overcome these limitations [TMS15]. However, in real networks may occur inconsistencies that lead to allocation conflicts [WMC15], partly reducing the advantages over centralised approaches to the communication problem.

**Allocation of tasks based on auctions:** allocation of tasks through auctions are based on the idea of an agent acting as auctioneer (offering the other agents one or more tasks that need to be performed), while the other agents make offers for the tasks of their interest. The agent that makes the highest bid for each task is chosen to execute it [SD15]. According to [WMC15], the fact of having a central controller to define the winner of a task has the disadvantage that all agents need to communicate directly to a single point (auctioneer). Another possibility are the combinatorial auctions. These are auctions where agents make offers for sets of tasks rather than for a simple task. [SGSTS14] presents a framework for task allocation based on this type of auction. According to [LMB+13], combinatorial auctions offer a good resulting allocation, but they become computationally costly as the number of tasks increases. [CBH09] presents a distributed algorithm called Consensus-Based Bundle Algorithm (CBBA), which combines auctions-based approaches and consensus-based algorithms. In the algorithm proposed by the authors, similar tasks are grouped to form bundles of tasks.

**Task allocation based on DCOPs:** task allocation problems can also be represented as Distributed Constraint Optimization Problems - DCOP. A DCOP is represented by a set of agents, a set of variables, a set of domains for the variables, and a set of constraints [YZF14]. Each variable is controlled by an agent, and each agent can control multiple variables [YZF14]. According to [SD15], the constraints can be temporal, spatial,

among others. Different DCOPs approaches are found in the literature. In [SFOT05], for example, a task allocation algorithm called LADCOP (Low-communication Approximate DCOP) is proposed, which requires little communication and is based on token passing. [RPF+10] presents a solution for coalition formation considering spatial and temporal constraints.

## 2.4 Multi-agent programming

Traditional programming was not designed to deal with autonomy issues, which is required when working with agents [BD13]. Shoham initially identified this particularity in [Sho93], introducing an agent-oriented programming language. Since then, different agent-oriented programming languages have been proposed, such as Jason [BHW07], JACK [BRHL99], 2APL [Das08], GOAL [HBHM01], ALOO [Ric14] and SARL [RGG14].

Some studies present comparisons between the Jason language and other agent-oriented languages or compare it with languages of different paradigms. A comparison of performance between the languages Jason, 2APL and GOAL are shown in [BHH+10], indicating better performance of the Jason language in comparison to the others. [CHB13] presents a comparison between Jason and two actor-oriented languages, Erlang and Scala. The comparison was made regarding the execution time, memory consumption and use of the cores of the processors. According to the authors, although Jason represents a so-called "heavy" paradigm, its scalability and performance were close to those presented by actor-oriented languages. In [CZHB13] three actor-oriented languages (Erlang, Akka and ActorFoundry) and three agent-oriented languages (Jason, 2APL and GOAL) were compared. Concerning agent-oriented languages, in general, the Jason language obtained the best results.

Agent-oriented programming languages initially focused on the programming of individual agents, leaving aside aspects related to the environment as well as social aspects [BD13]. The abstraction of these aspects together with the abstraction related to agent-oriented programming itself makes agent-oriented programming more powerful and useful for solving complex problems.

In this direction, [BD13] presents characteristics that platforms oriented to the programming of systems with multiple agents should consider:

- **Long-term goals and reaction to events:** agents must be alert to changes in the environment, responding to them appropriately without neglecting long-term goals.

- **Courses of action depend on circumstances:** an event can trigger different courses of action depending on the circumstances, i.e. depending on the information the agent has about himself, other agents and the environment, a course of action or other can be performed. Thus, it should be possible to specify different courses of action for the same event.
- **Choosing the course of action only when it is close to acting:** courses of action selected at an early stage may not be more interesting or even valid when they are performed due to possible changes in the environment, hence the importance of this feature.
- **Plans failures:** Plans may fail during execution. There must be a mechanism to deal with these failures.
- **Rational behaviour:** agents must behave rationally. When an agent has a goal, it should be able to reason about how to achieve it and only give it up if there is evidence that he can no longer be achieved or there is no more motivation for it.
- **Social skills - communication and organisation:** agents need to interact to cooperate and coordinate their actions. Besides, agents can be part of an organisation that regulates their joint activities.
- **Code change at runtime:** platforms should allow code change during execution. That includes changes in the plan library and eventually changes in the social structure and norms in an agent organisation.

Some MAS development platforms that present levels of abstraction beyond agent orientation are JaCaMo [BBH+13], Jadex [BPL05], Magentix2 [SGFEB13] and TAEMS [Dec96]. JaCaMo was chosen for the development of the mechanism proposed in this work since it presents the necessary characteristics for its development, besides to use the language Jason that, as previously seen, performs well against other languages oriented to agents. The mentioned characteristics are presented in the next section.

### 2.4.1 JaCaMo

JaCaMo [BBH+13] is a platform for programming MASs that integrates three solutions: the agent-oriented programming language Jason [BHW07], providing support for agent development; CArtAgO [RVO07] which provides abstraction required for environment-level programming; and Moise [HSB07], which allows working at the organisation level. Figure 2.3 gives an overview of the dimensions of the JaCaMo platform.

The Jason language, which is used for coding the agents' dimension, is based on the BDI architecture, which allows developing agents based on the concepts of beliefs,

desires and intentions. Jason-encoded agents are capable, among other things, of communicating, interacting and reacting to events according to the plans available to the agent.

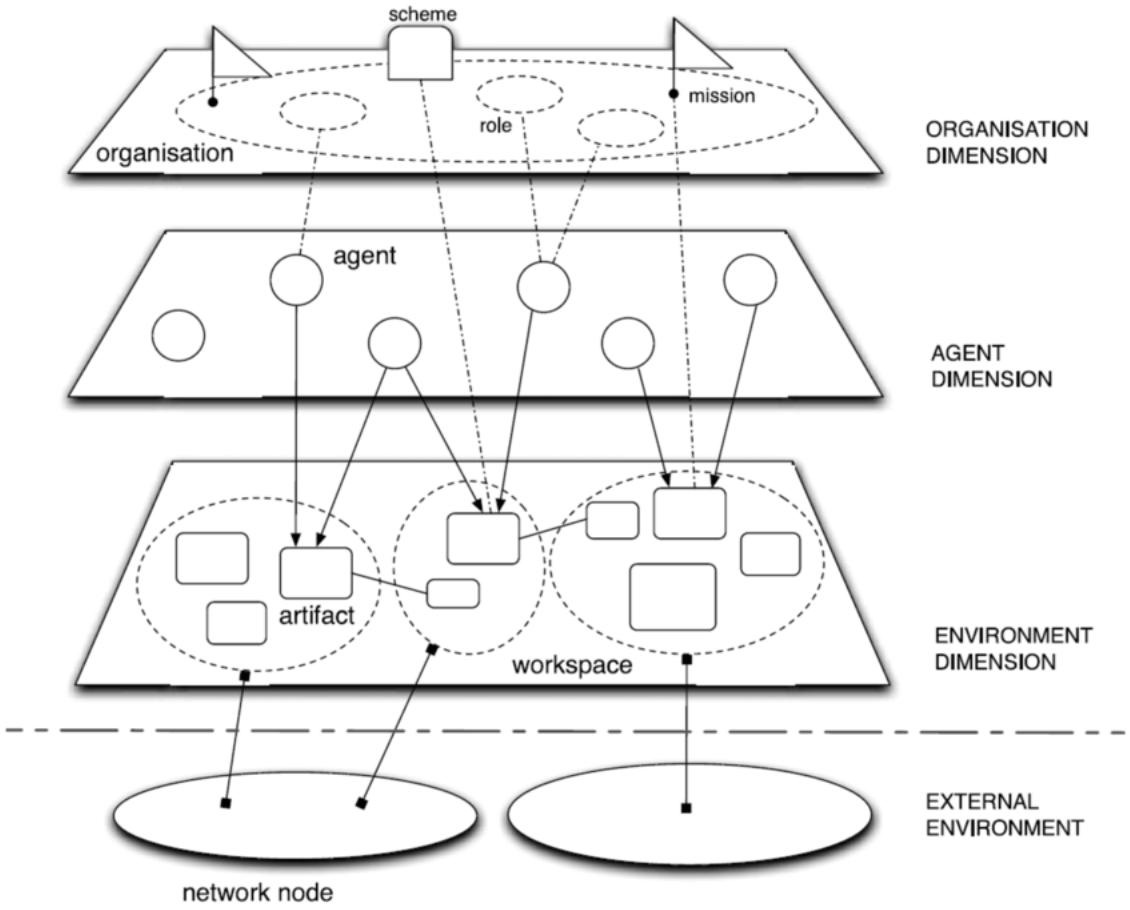


Figure 2.3 – JaCaMo architecture overview [BBH+13].

Through Moise, an organisation can be defined in terms of groups, roles, social plans and norms. It is also possible to determine the sequence in which the goals must be executed and also define the goals that need to be performed in parallel, besides allowing to specify the obligations and permissions for the roles assumed by the agents.

CArtAgO is used for environment level programming. In CArtAgO the environment or parts of the environment can be represented by entities called artifacts. Artifacts store information that is accessed by agents through observable properties. Agents can generate updates to the values of observable properties using the operations provided by the artifacts themselves. Artifacts may be distributed on different nodes of a network.

The features presented by the solutions that make up the JaCaMo platform make it appropriate for the development of the mechanism proposed in this work. In fact, the Jason language is used for the coding of the agents, including the algorithms for the allocation of tasks. CArtAgO artifacts are used to communicate roles and the tasks that need to be performed.

In order to improve understanding, some of the mentioned aspects will be presented in the next sections through an example called *Building-a-House* described in [BBH+13]. In the example, an agent named Giacomo wants to build a house and for that Giacomo needs to hire contractors to perform different tasks during the building process. In the example, the contracting of the services is carried out by means of a simple auction-based mechanism. Service providers only bid on the tasks they can perform, and the best offer for each task (lowest cost) is the winner. After the auction, the service providers begin the execution of the tasks, where a coordination mechanism is used to guarantee the execution of the tasks in a coherent way. In the example, six agents are defined, five of them representing the service companies, plus Giacomo agent.

### 2.4.2 Jason

The Jason language is based on the BDI architecture, which allows developing agents based on the concepts of beliefs, desires and intentions. Jason is an extension of the AgentSpeak language introduced by Rao [RG95]. According to Rao [RG95], AgentSpeak language can be viewed as a textual language of Procedural Reasoning System (PRS). PRS is an architecture that explicitly incorporates the belief-desire-intention (BDI) architecture. The PRS, represented in Figure 2.4, was originally developed by Georgeff and Lansky [PGL86].

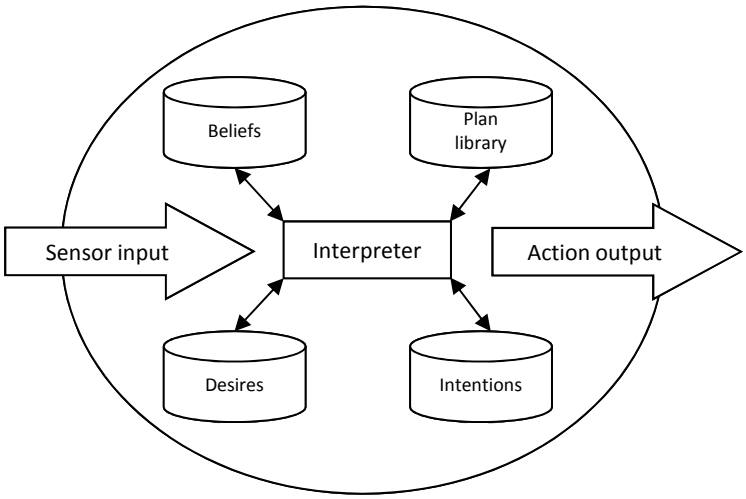


Figure 2.4 – Procedural Reasoning System (PRS)

In Jason, such as in PRS, an agent has a set of beliefs, a set of desires, a set of intentions and a set of plans (plan library). As mentioned earlier, beliefs represent the agent's knowledge about the environment, the desires are goals that the agent would like to achieve, and intentions are the goals that the agent decided in fact achieve. Each plan has: a goal, which is the postcondition of the plan, i.e. the objectives it achieves;

a context, which represents the precondition of the plan, i.e. the things that when became true in the environment will trigger the execution of that plan and; a body, which represents the course of actions to achieve the goal. The interpreter (Figure 2.4) determines the agent's reasoning cycle of an agent, i.e. considering perceptions from the environment, beliefs, desires and intentions of an agent it reasons about how to achieve the agent's goals. Figure 2.5 represents the Jason reasoning cycle.

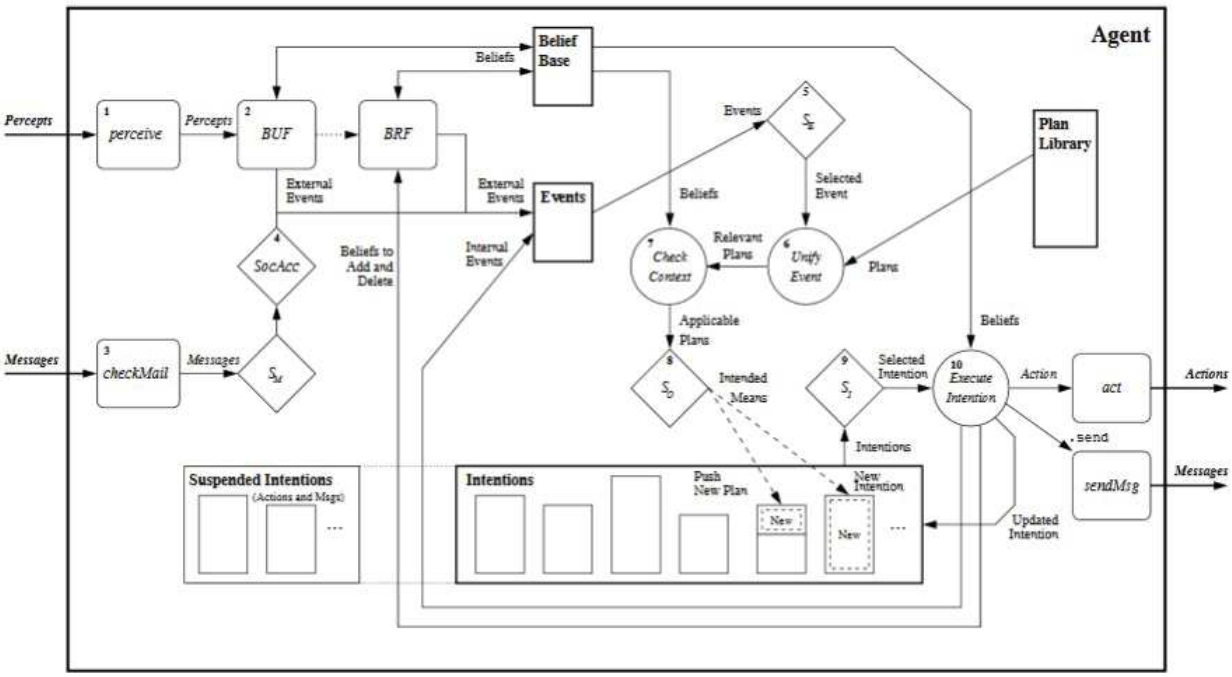


Figure 2.5 – Jason reasoning cycle [BHW07].

.

In the *Building-a-House* example, Jason is used to define the initial goals, beliefs, plans, and rules for each of the agents. Listing 2.1 shows part of the Giacomo agent code. Agent Giacomo has an initial goal of building a house (*!have_a_house*). The plan *+!have_a_house* has two subgoals *!contract* and *!execute*. The first one is for contracting the companies service and the second one for the managing the building itself. The *+!contract* plan also has two subgoals: *!create_auction_artifacts* which creates the auction artifacts for the services (tasks) that agent Giacomo needs to hire; and *!wait_for_bids* which makes the agent wait some time for the companies to bid on the tasks before processing the winners. In other words, the agent starts running by creating the auctions for tasks that need to be performed and then wait for bids from the companies.

```
1  !have_a_house.
2
3  +!have_a_house
4  <- !contract;
5  !execute.
6
7  +!contract
8  <- !create_auction_artifacts;
9  !wait_for_bids.
10
11 +!wait_for_bids
12 <- .wait(5000);
13 !show_winners.
14 ...
```

Listing 2.1 – Operations available in the roles artifact.

### 2.4.3 Cartago

CArtAgO [RVO07] is a framework that provides environment level programming. CArtAgO is based on the A&A meta-model [RPV11] – see Figure 2.6. In CArtAgO the environment can be represented through entities called artifacts (programmed in Java language) which can be created by the agents. Artifacts store information that can be accessed by agents through observable properties. Observable properties can be generated when the artifact is instantiated or through operations. Agents also use the operations to update the values of the observable properties. When an agent focus on an artifact, the observable properties of the artifact are added as beliefs and the agent is also able to use the operations available in the artifact. Artifacts may be distributed on different nodes of a network.

In the *Building-a-House* example, CArtAgO framework is used to create an artifact to aid in the auction mechanism, as shown in Listing 2.2. In the artifact four observable properties are defined: *task*, representing the description of the task; *maxValue* being the maximum value to be accepted for the task; *currentBid*, representing the lowest bid amount received; and *currentWinner* representing the current winner of the auction for the task. These observable properties are generated when the Giacomo agent creates instances of the artifact for each service that needs to be auctioned (Listing 2.1, line 8, *!create_auction_artifacts*).

The artifact in the example provides the *bid* operation, through which agents can bid on the task. When an agent performs the *bid* operation, the artifact verifies if the received bid value is lesser than the *currentBid* value, and then updates the value of the *currentBid* to the received value and also the value of the *currentWinner* for the agent who provided the bid.

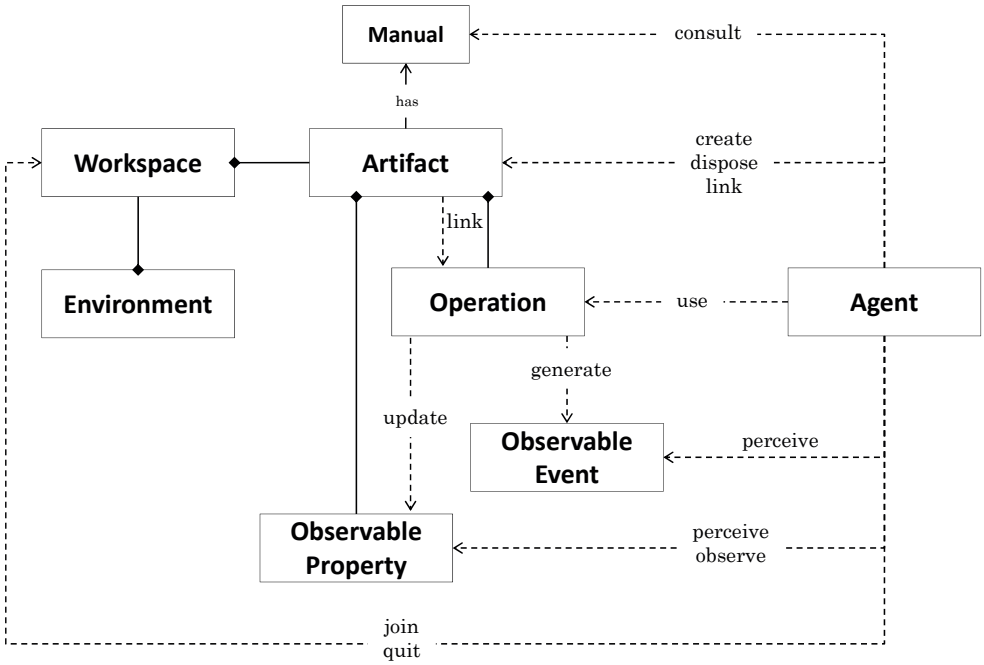


Figure 2.6 – CArtAgO A&A meta-model [RPV11]

.

```
1  public class AuctionArt extends Artifact {
2
3     @OPERATION public void init(String taskDs, int maxValue)  {
4        // observable properties
5        defineObsProperty("task",          taskDs);
6        defineObsProperty("maxValue",      maxValue);
7        defineObsProperty("currentBid",    maxValue);
8        defineObsProperty("currentWinner", "no_winner");
9     }
10
11    @OPERATION public void bid(double bidValue) {
12       ObsProperty opCurrentValue  = getObsProperty("currentBid");
13       ObsProperty opCurrentWinner = getObsProperty("currentWinner");
14       if (bidValue < opCurrentValue.doubleValue()) {
15          // the bid is better than the previous
16          opCurrentValue.updateValue(bidValue);
17          opCurrentWinner.updateValue(getCurrentOpAgentId().getAgentName());
18       }
19    }
20 }
```

Listing 2.2 – Example of artifact defined in CArtAgO.

Listing 2.3 shows part of the agent code for a company in the *Building-a-House* example. The company has an initial belief *my_price(300)* representing the price the company will use as bid. The initial goal of the company agent is to look for and focus on an artifact related to the auction for a Plumbing service (*!discover_art("auction_for_Plumbing")*).

When the agent focus in the *"auction_for_Plumbing"* artifact (see Figure 2.7 (A)), the observable properties of the artifact are added as beliefs, and the agent is also able to use the operations available in the artifact. In the example, it means that the four observable properties *task*, *maxValue*, *currentBid* and *currentWinner* are added as beliefs in the belief base of the agent. When the observable property *currentBid* is updated by some other agent (see Figure 2.7 (B)), those changes will update the beliefs in the agents who focused on the artifact. In the example, the changes may trigger the agent plan *+currentBid(V)* (Listing 2.3, line 5). The plan will be triggered whenever its context is true, in this case only if the agent is not the current winner of the task and if the price it can offer is lesser than the current bid value. In case the plan is triggered, the agent will use the bid operation available on the artifact.

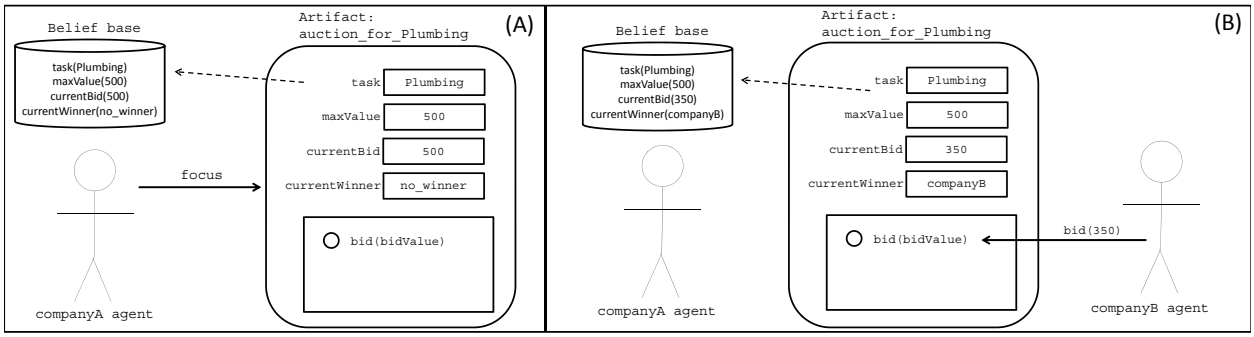


Figure 2.7 – Example of artifact defined in CArtAgO.

```
1  my_price(300). // initial belief

3  !discover_art("auction_for_Plumbing").

5  +currentBid(V)[artifact_id(Art)]
6  : not i_am_winning(Art)  & my_price(P) & P < V
7  <-
8  bid( P ).

10 +!plumbing_installed   // the organisational goal (created from an obligation)
11 <- installPlumbing.

13 ...
```

Listing 2.3 – Operations avaialable in the roles artifact.

### 2.4.4 Moise

Moise [HSB07] allows to work at the organisation level, by defining groups, roles, social plans, norms, obligations and permissions for the roles assumed by the agents, as well as to determine if the goals must be executed in parallel or sequence.

In the example, Moise is used to specify the organisation defining, for example, roles such as house_owner that is assumed by the agent Giacomo and the electrician role that can be assumed by an agent that has the capability to provide electrical services. In the example, the main goal of the organisation is the construction of the house ("house_built" goal), which is achieved through smaller goals, such as "walls_built", "plumbing _installed". These goals are defined in the functional specification of the organisation. To achieve the goals, missions such as build_walls are defined and also the roles responsible for each mission. In the example, the electrician role is responsible for the install_electrical_system mission. Although Moise could be a great option to work with the organisational level, in this thesis we kept the organisation defined in a more simplified way.

# 3. MULTI-AGENT TASK ALLOCATION PROBLEM

In this chapter, we first present the type of tasks considered in this thesis. Then we describe the flooding scenario to exemplify how these type of tasks fits into a complex scenario. Finally, we present formalisation aspects of our Multi-Agent Task Allocation problem.

## 3.1 Type of Tasks

Different types of tasks can be used to address different requirements involved in real-world scenarios, which cannot be adequately represented by only one type of task. This is because in real-world scenarios tasks may have complex structures and other domain-specific dependencies. In this thesis we consider the types of tasks below which are based in [Zlo06]. Figure 3.1 represents the type of tasks considered in this work.

**Atomic task:** a task is atomic if it cannot be decomposed into subtasks. It can only be carried out by a single robot. A task is considered atomic or not, depending on the level of the domain model being used. In this work, atomic tasks will be considered as subtasks of both compound tasks and decomposable simple tasks.

**Decomposable simple task:** a task that can be decomposed into a set of atomic subtasks or other decomposable simple tasks as long as the different decomposed parts cannot be carried out by more than one robot, that is, the decomposed parts must all be carried out by the same robot. We refer to this type as a **DS task** throughout this work.

**Compound task:** a task that can be decomposed into a set of atomic or compound subtasks, presenting only one possible decomposition at any level. We consider that the subtasks of a compound task may be impacted by constraints, and then we separate the compound task type into two types.

**Compound for many:** we consider that the subtasks here are not impacted by constraints, that is, the subtasks can be allocated from one up to $M$ (many) robots, where $M$ is the number of subtasks; in this case, we call it a **CM task**.

**Compound for exactly N:** when each of the subtasks needs to be allocated to a different robot, we call it a **CN task** (since there are $N$ subtasks that need exactly $N$ robots).

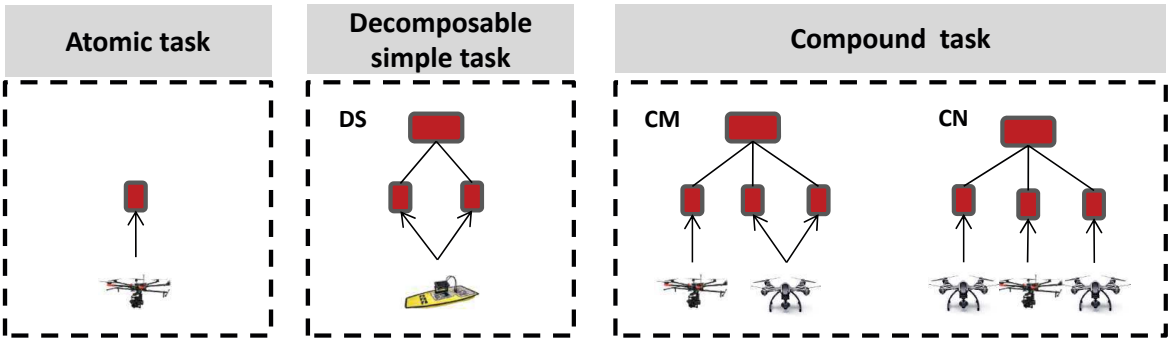


Figure 3.1 – Types of tasks considered in this thesis.

Next section we describe the flooding scenario with tasks that could be performed or supported by robots and exemplify how the type of tasks described here are suitable for a complex scenario like that.

## 3.2 Scenario Example - Rescue in Flood Disasters

Disasters, whether natural or man-made, are damaging and are life-threatening for people in impacted areas. According to the World Disaster Report released by the International Federation of Red Cross and Red Crescent Societies [IFR15], 518 disasters were reported worldwide in 2014, affecting approximately 107 million people. The damage caused by these disasters is estimated at 99 billion USD in 2014 alone.

From 518 disasters reported in the world in 2014, 132 are related to floods [IFR15]. There were more than 1,700 flood-related disasters in the last ten years (it should be noted that these numbers consider different types of floods, including floods by storms and sea waves). The number of people affected by floods in 2014 was more than 36 million, with losses estimated at 37 billion USD.

In addition to damage to properties, flooding has the potential to put people's lives at risk [Cou07]. Depending on the area where the flood occurs, the risks can be considered major or minor. In sparsely populated or unpopulated areas, for example, flooding offers low risks to people's lives. In large populated areas, such as urban centres, the risks are greatest [Cou07]. Human activities such as devastation, mining and industrialisation, in general, contribute to the occurrence of floods [SKV+12].

Disasters, regardless of type, are treated in four stages [MTN+08]: Preparedness, Prevention, Rescue and Recovery. The tasks performed in the phases of Preparedness and Prevention (phases prior to the disaster) refer to the prevention and reduction

of risks and the efforts to be prepared for the possible occurrence of a disaster. The Rescue and Recovery phases happen during and after the event. The Rescue phase includes tasks such as searching and rescuing victims, among others [MTN+08]. The Rescue phase is also called the Response phase by some authors, as in [RHI+15]. The Recovery phase is intended to reduce the damage and long-term life risks caused by the event [MTN+08].

When a flooding disaster occurs, teams are called into action to assist in the rescue and recovery phases. According to [RHI+15], these teams are usually organised by following a command hierarchy model, with individuals playing different roles during a mission.

One of the first and foremost tasks to be carried out in a disaster situation is to obtain an overview of the affected region (situational awareness) in order to identify where help is needed and to define resource allocation [MTN+08]. The execution of this task, such as many others that need to be carried out, involves several risks since in a disaster there is a lot of uncertainty regarding the conditions of the environment. In fact, in a disaster environment, there is uncertainty about the infrastructure and logistics situation, resource availability, equipment operation and whether information remains valid (which may change over time) [FRR+15].

One way to reduce the risk to people involved in rescue tasks is to use robots. For example, unmanned aerial vehicles (UAVs) can be used to obtain an overview of an area affected by a disaster, helping to identify where aid is needed and the best way to reach the victims, thus reducing the risk to other individuals. Next, we present tasks that could be performed or assisted by robots in flood situations. Then we discuss the aspects that need to be considered for the allocation of these tasks to teams of heterogeneous and autonomous robots working in flood situations.

### 3.2.1 Tasks and Robots in Flood Scenarios

According to Murphy, there are several tasks that could be executed or aided by robots during flood disasters [Mur14], including the ones presented next (1 to 9). In addition to the tasks suggested by Murphy, Scerri [SKV+12] adds the task "Collection of data and water samples" (10) also described below.

1. **Awareness, recognition and mapping:** robots can be used to obtain an overview of the region affected by the disaster. The collected information can be useful to identify the extent and impact of the flood, discovering the state of streets, bridges, among others;

2. **Search:** robots can be used to assist in the search for people in danger or missing people. For this task, robots must have the ability to detect people, either through sensors or images;

3. **Delivery items:** in some situations the rescue of victims can be delayed, either because there are a large number of victims to be rescued or because the victims are in places that are difficult to reach. In these situations, robots can be used to deliver supplies, medicines, lifejackets, and other life support items to victims in hard-to-reach regions;

4. **Provide logistical support:** robots could also carry equipment and supplies;

5. **Risk monitoring for rescue teams:** there are situations where, for example, a team of rescuers may be in a flooded area rescuing a victim using a boat. If there are water currents, debris, tree branches, among others, those things expose the rescue team to risks. In this sense, robots can be used to monitor the water and warn a rescue team of any wreckage that is heading toward the team;

6. **Network signal propagation:** if it is necessary to maintain a communication network between different points (or regions), robots can act as repeaters, extending the range of the network communication;

7. **Analysis of dams:** robots can be used to obtain images of dams, helping to check for signs of rupture or overflow;

8. **Rescue:** robots can be used to aid in the rescue of victims;

9. **On-site medical evaluation and intervention:** it is often difficult or even impossible for medical staff to reach out to the victims. In this case, robots could be used to perform visual inspections or even telemedicine. According to [Mur14], this is one of the biggest challenges regarding the use of robots in disasters.

10. **Collection of data and water samples:** robots can be used to collect data on temperature, oxygen level among other possible measurements at the site, as well as collect samples of water for more detailed analysis, since flooded areas may be contaminated and thus put the health of the local population at risk.

Due to the characteristics of this type of disaster, the mentioned tasks could be performed or supported by autonomous robots with different types of locomotion such as Unmanned Surface Vehicles (USVs), Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs). Next, we make some considerations about each of these types of robot.

- **USVs:** these are vehicles that move over water. They can be used to obtain images from a different angle and different locations that a UAV cannot access, and can be used to identify victims. USVs can also be used to collect water samples or perform other measurements (they may have sensors capable of measuring water temperature, for example). Thus, different tasks could be performed by the USVs, such as mapping, searching, data collection and network signal propagation. It should be noted that in order to perform any task, one must take into account the physical and computational capabilities of the robots.

- **UAVs:** these are vehicles that operate in the air. Due to their type of locomotion, they can be used to obtain aerial images, identify victims, deliver items to isolated victims and monitor risks to the teams. Thus, UAVs would have the ability to perform the tasks such as mapping, searching, item distribution, risk monitoring, and network signal propagation.

- **UGVs:** these are land vehicles. Due to their type of locomotion, they can be used for tasks in locations near flooded areas, or in portions of lands isolated by floods. Vehicles of this type can, for example, assist in the transport of items or even obtain images from an angle different from the other types of vehicles. Thus, UGVs could be used to perform tasks such as mapping, distribution of items and network signal propagation.

According to [Cra17], small UAVs have already been used in some flood disasters or disasters that have had associated flooding, such as the floods in Thailand in 2011, Hurricane Haiyan in the Philippines in 2013 and floods in Serbia in the Balkan region in 2014. In these cases, a small number of manually controlled UAVs were used to obtain an overview of the affected areas, as well as carrying out search tasks for missing people.

The use of robots can mitigate the risks faced by the teams working in the Rescue phase, but adds other complexities to the process. To cover a large area, for example, a large number of robots may be required, which may make manual control impractical, since at least the same amount of operators as robots would be required. Other important aspects that must be considered are: which is the best robot to perform each task; how many tasks each robot can perform with the amount of energy available; what is the minimum distance that the controller should be from the robot (so as not to put the controller at risk), among other complicating aspects. Considering these aspects, the use of autonomous robots acting in a coordinated and cooperative way, allocating the tasks among themselves in order to be able to execute them properly, could be considered an appropriate solution for these scenarios.

### 3.2.2 Allocation of tasks in flood scenarios

Task allocation is the process of deciding which robots should perform which tasks. However, the allocation process may be impacted by constraints imposed by the domain problem. In this section we give an example of such constraints in the domain of the flood disaster scenario.

As mentioned earlier, during a rescue phase in a flooding disaster, teams with individuals performing different roles are called into action to work on various tasks. These individuals are usually part of an organisation responsible for managing the tasks that need to be performed during the rescue phase.

The capabilities of individuals define which roles each one can play and those roles constraints the tasks that individuals can perform. In the same way, we may have robots with different capabilities which allow the robots to play some roles and not others. The robots may only carry out tasks according to the roles they can play.

Also, each robot has a maximum number of subtasks that can allocate to itself. In the flooding scenario this constraint may be related to the amount of energy (fuel) available to a robot.

Regarding the Rescue tasks described above, we exemplify next how they can be defined according to the types of tasks presented in Section 3.1.

As we mentioned earlier, depending on the area where the flood occurs, the risks to people's lives can be considered major or minor and that can impact how the tasks are defined by the organisation. For example, in tasks related to the mapping of areas, robots can be used to obtain images of a region. Let's consider that during a flood, the organisation wants to know the situation of a dam that is a few kilometres from the city, to verify if there is a possibility of rupture of the dam. Thus, the organisation wants images of the dam and some specific locations around it. Then, for example, the organisation would create a task for each of the locations for which it needs images (including the dam).

However, the organisation may decide that it is not worth sending more than one robot to get these images, as other tasks need to be performed in more populated areas where more robots will be needed. In other words, the organisation wants all these tasks allocated to the same robot. That fits in the **DS** type of task described in Section 3.1, where a task is composed of subtasks that need to be performed by the same robot. In this case there would be a mapping task with several subtasks, each referring to one of the locations that the organisation wants to obtain images.

Now consider that the organisation wants to map an area by getting images of five locations. Then the organisation would create a task for each of the locations for which it needs images. In this case, the organization may decide that it does not need

to restrict the allocation of these tasks to only one robot, which could be allocated by one or more robots and that robots should decide the best allocation considering all the other tasks. That fits in the **CM** type of task described in Section 3.1, where a task is composed of subtasks that can be allocated from one up to $M$ (many) robots, where $M$ is the number of subtasks. In this case there would be a mapping task with several subtasks, each referring to one of the locations that the organisation wants to obtain images. The examples above are also applied to other tasks in flood scenarios such as search, delivery items, sample water, among others.

Let's now consider the task of network signal propagation. Consider that the organization wants to maintain communication network between different areas. In this case robots can act as signal repeaters, extending the range of communication. The organization has stated that to maintain this communication, it will be necessary to have repeaters in six locations. In this case, to maintain the communication channel it is not possible for only one robot to play the role of repeater, because repeaters are required in different locations at the same time. In other words, the organisation needs each of these tasks being necessarily allocated by a different robot. That fits in the **CN** type of task described in Section 3.1, where a task is composed of $N$ subtasks that need to be allocated to exactly $N$ different robots. In this case there would be a task with several subtasks, each referring to one of the locations that the organisation need a robot acting as reapeater.

## 3.3 Multi-agent Task Allocation Problem

In this section, we formally state our Multi-Agent Task Allocation optimisation problem. We assume that there are $n_r$ available robots $R = \{r_1, \ldots, r_{n_r}\}$, $n_t$ tasks $T = \{t_1, \ldots, t_{n_t}\}$, and $n_{st}$ subtasks $ST = \{st_1, \ldots, st_{n_{st}}\}$ where each subtask $st_k$ belongs to exactly one task $t_j$. Each task $t_j$ has one or more subtasks from $ST$, and we use $N_j$ for the specific number of subtasks that the $j$th task has. Furthermore, we use the binary variable $p_{jk}$ indicating whether $st_k$ belongs to $t_j$ to formalise the constraints above, as follows:

$$\sum_{j=1}^{n_t} p_{jk} = 1 \;\; \forall k = 1, ..., n_{st}. \tag{3.1}$$

$$\sum_{k=1}^{n_{st}} p_{jk} >= 1 \;\; \forall j = 1, ..., n_t. \tag{3.2}$$

Each subtask $st_k$ may be allocated to at most one robot, and each robot $r_i$ can perform at most $L_i$ subtasks (the task limit for robot $r_i$). We assume that a task $t_j \in T$ is considered allocated only if all of its subtasks were allocated to robots following the

constraints described in this section. Let $f_{ik}$ be a binary variable indicating whether $st_k$ is assigned to $r_i$, and let $u_{ik} \in \mathbb{R}$ be the utility value associated for the allocation of $st_k$ to $r_i$.

$$\sum_{i=1}^{n_r} f_{ik} \leq 1 \;\; \forall k = 1, \ldots, n_{st}. \tag{3.3}$$

$$\sum_{k=1}^{n_{st}} f_{ik} \leq L_i \;\; \forall i = 1, \ldots, n_r. \tag{3.4}$$

Let us consider further that there are $n_q$ types of tasks $Q = \{q_1, \ldots, q_{n_q}\}$ and that each task $t_j$ from $T$ is associated with exactly one type of task from $Q$. The binary variable $w_{jq}$ indicates whether $t_j$ is of type $q_n$.

$$\sum_{q=1}^{n_q} w_{jq} = 1 \;\; \forall j = 1, ..., n_t. \tag{3.5}$$

Each type of task $q \in Q$ has a minimum and maximum ($min_q$, $max_q$) number of subtasks that a robot must take on when allocating to itself subtasks of a single task of type $q$. We use $min_j$ and $max_j$ for the minimum and the maximum number of subtasks that a robot must take when allocating subtasks of task $t_j$.

Note that with the $min$ and $max$ constraints we can represent all task types $DS$, $CN$, and $CM$ as described earlier. For example, a robot trying to allocate a $DS$ task must take all of the $N_j$ subtasks, i.e. it must take a minimum of $N_j$ and a maximum of $N_j$ subtasks since the type $DS$ requires the allocation of all subtasks to *exactly one* robot. Similarly, a robot trying to allocate a $CN$ task with $N_j$ subtasks must take only *one* subtask, i.e. it must take a minimum and a maximum of *one* subtask. A robot trying to allocate a $CM$ task with $N_j$ subtasks must take a minimum of *one* and a maximum of $N_j$ subtasks. Equations (3.6) and (3.7) state, for each task, the constraints on the number of subtasks a robot must allocate to itself based on the type of that task.

$$\sum_{k=1}^{n_{st}} f_{ik}.p_{jk} \geq min_j \quad \forall j \text{ s.t. } t_j \in T;\; i = 1, \ldots, n_r. \tag{3.6}$$

$$\sum_{k=1}^{n_{st}} f_{ik}.p_{jk} \leq max_j \quad \forall j \text{ s.t. } t_j \in T;\; i = 1, \ldots, n_r. \tag{3.7}$$

In addition, there are also constraints related to the roles the robots may play in the organisation according to their capabilities. Assume that there are $n_c$ capabilities $C = \{c_1, \ldots, c_{n_c}\}$ and $n_e$ roles $E = \{e_1, \ldots, e_{n_e}\}$.

Each role $e$ is associated with the capabilities a robot must have in order for it to be able to play that role. Each robot $r_i$ has a set of capabilities, which determine the set of roles it is able to play. Each subtask $st_k$ is associated with a set of roles a robot must be able to play in order to execute it. We use the following binary variables to formalise the constraints above: $g_{yx}$ indicates whether role $e_y$ requires capability $c_x$; $h_{ky}$ indicates whether subtask $st_k$ requires role $e_y$; $v_{ix}$ indicates whether robot $r_i$ has capability $c_x$ and; $z_{iy}$ indicates whether robot $r_i$ is able to play role $e_y$.

$$\sum_{x=1}^{n_c} g_{yx} \leq n_c \;\; \forall y = 1, ..., n_e. \tag{3.8}$$

$$\sum_{y=1}^{n_e} h_{ky} \leq n_e \;\; \forall k = 1, ..., n_{st}. \tag{3.9}$$

$$\sum_{x=1}^{n_c} v_{ix} \leq n_c \;\; \forall i = 1, ..., n_r. \tag{3.10}$$

$$\sum_{y=1}^{n_e} z_{iy} \leq n_e \;\; \forall i = 1, ..., n_r. \tag{3.11}$$

$$\sum_{x=1}^{n_c} g_{yx}.v_{ix}.z_{iy} = \sum_{x=1}^{n_c} g_{yx}.z_{iy} \quad \forall i = 1, ..., n_r; \; y = 1, \ldots, n_e. \tag{3.12}$$

$$\sum_{y=1}^{n_e} h_{ky}.z_{iy}.f_{ik} = \sum_{y=1}^{n_e} h_{ky}.f_{ik} \quad \forall i = 1, ..., n_r; \; k = 1, \ldots, n_{st}. \tag{3.13}$$

Finally, the objective of our multi-agent task allocation problem is to find an allocation that maximises the sum of utilities of the agents while satisfying all the above constraints. The idea is that the process of maximising the sum of individual utilities simultaneously improves the global utility [DZKS06]. The objective function of our optimisation problem can be stated as follows:

*Objective:*

$$\max_{\{f_{ik}\}} \sum_{i=1}^{n_r} \sum_{k=1}^{n_{st}} u_{ik}.f_{ik} \tag{3.14}$$

# 4. DECENTRALISED TASK ALLOCATION

In this chapter we define the design of our decentralised task allocation mechanism, a domain-independent task allocation solution for multi-agent systems and describe our implementation of the mechanism in the JaCaMo MAS development platform. Our decentralised mechanism allows the allocation of different types of tasks to heterogeneous robot teams, considering that these robots may play different roles and they carry out tasks according to the roles they can play and their load capacities. We first provide a general view of the allocation process, focusing on the main elements of the proposed mechanism and then we provide details about the design and implementation. Some of the ideas presented here first appeared in [BB17]. In [CKB+18] we briefly describe our attempt to use our decentralised approach in the Multi-Agent Programming Contest in 2017. Our initial experiments were promising, but the integration was started late in development and it was not possible to complete and testing in a timely manner for the contest.

## 4.1 Overview of the Allocation Process

The main elements considered in the proposed mechanism are presented in Figure 4.1. Initially, we consider the existence of an *organisation* that is responsible for announcing the *tasks* (with their corresponding subtasks) that need to be carried out by the *agents* in a given mission. As mentioned before, we use the term agent to refer to the main control software of an individual robot of any kind. The tasks provided by the organisation are published on a *blackboard* that can be viewed by all the agents available for the mission. The organisation is also responsible for publishing the *roles* that agents may play in the organisation. Finally, the environment is the place where agents carry out the tasks. Blackboard is a widely known architecture (see [Hay85] and [RB07]) which works as a globally accessible space and can be used, for example, for sharing information among agents [RB07]. Note that we could use any other way to provide necessary information to the agents. For example, in [BMZB18] we propose an architecture for supporting the decentralised allocation of tasks built on the idea of having communication and coordination in a multi-agent system through a private blockchain.

Regarding the process itself, it is initially considered that an organisation has a set of agents to carry out a mission and that these agents are waiting for the tasks they will be asked to carry out (the agents start executing without having any assigned task). When necessary, the organisation publishes a set of tasks with their subtasks on the blackboard to which all agents have access. By identifying the new set of tasks

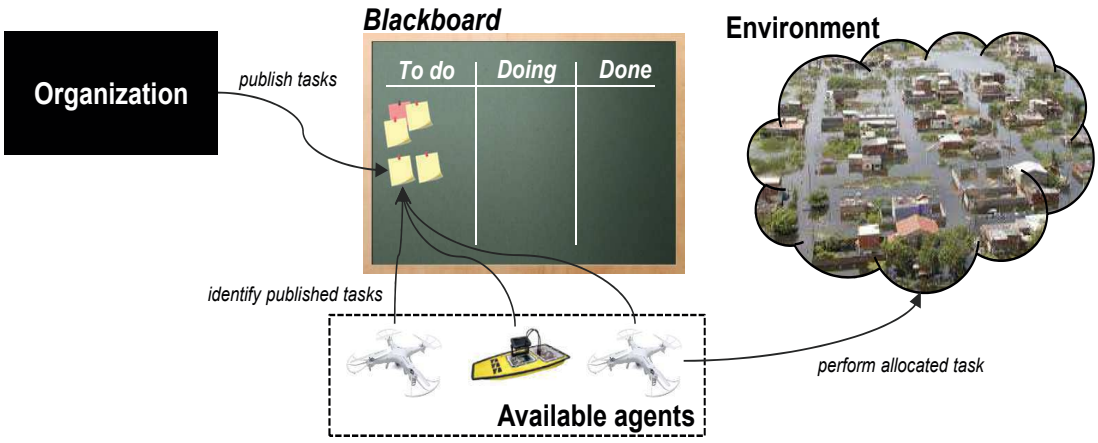


Figure 4.1 – Overview of the Task Allocation Process.

available, the agents begin the allocation process based on the mechanism we describe in this chapter. Information about the roles required by the organisation is also published on the blackboard.

Simply put, each agent initially identifies the roles available in the organisation and based on its capabilities it checks which roles it can possibly play in the organisation. The agent then identifies on the blackboard the subtasks it can carry out based on the roles it can play (each subtask is associated to a role which an agent must be able to play to perform that subtask). The agents then exchange bids for the subtasks they want to be allocated to (subtasks with the highest utilities for themselves). When an agent receives a bid that improves its bid for a subtask it wanted to allocate to itself, the agent withdraws that subtask from the list of its allocated subtasks and checks which subtask it will bid for in order to replace the subtask it withdrew. These steps are repeated until the robots agree on the overall allocation. It is important to mention that our mechanism allows for all the subtasks to be bid on asynchronously. When the agents finish the allocation process, the agents with allocated subtasks start to carry them out.

Note that at the end of the allocation process, there might be agents without any allocated subtask as well as subtasks that could not be allocated to any suitable/available agent. Such results depend on the constraints indicated and the features of available agents. For example, if the organisation announces tasks with more subtasks than the total limit (capacity) of all the agents together, clearly some subtasks will not be allocated. On the other hand, if the total limit of the agents together is higher than the number of subtasks to be allocated, there may be agents without any subtask allocated. Also, the available agents may not be able to play the roles required to carry out some subtasks, so those subtasks will not be allocated to any agent. If an agent is not able to play any of the roles required by the subtasks, it will not be able to allocate any of them. Furthermore, constraints related to the number of robots required for subtasks of certain types of tasks may also lead to incomplete allocations, i.e. tasks in which at least one subtask was not allocated to any agent.

Next, we exemplify our allocation process using the flooding disaster scenario. As discussed in Section 3.2.1, there are several tasks that can be performed or assisted by robots during flooding disasters. One of the key tasks to be accomplished is to obtain situational awareness of the affected region, which involves mapping the affected areas. In such a task, robots are asked to obtain images of specific areas. Let us consider, in this example, that to accomplish the subtasks related to this task a robot needs to have flight capability and a camera to obtain the images from the area. Note that it would be possible to have the same task for a robot with sailing capability to get images from a different perspective. Also, robots could be used to deliver supplies to victims who are waiting for rescue. To perform the subtasks related to this task a robot needs to have load capability. Another task in flood disasters is the collection of water samples for analysis. To perform the subtasks related to this task, in our example the robot must have water navigation capability and be able to collect water samples. In this example, we will focus on these specific tasks. Note that, for the sake of simplicity, the information about tasks, roles, capabilities, and robots described below are deliberately kept at a rather high level.

Regarding the process itself, consider that the organisation needs to work on a flooding disaster by performing the above tasks: mapping areas, delivery items and collecting water samples for analysis. The organisation has three robots available to carry out a mission: one USV (Unmanned Surface Vehicle) and two UAVs (Unmanned Aerial Vehicle), which we will call respectively $USV1$, $UAV1$, and $UAV2$. The robots start executing without having any assigned tasks. $USV1$ has water navigation capability[1] and resources to collect water samples, while $UAV1$ and $UAV2$ have flying capabilities and cameras to take pictures. $UAV1$ also has a loader. Figure 4.2 represents the information each robot has about itself.

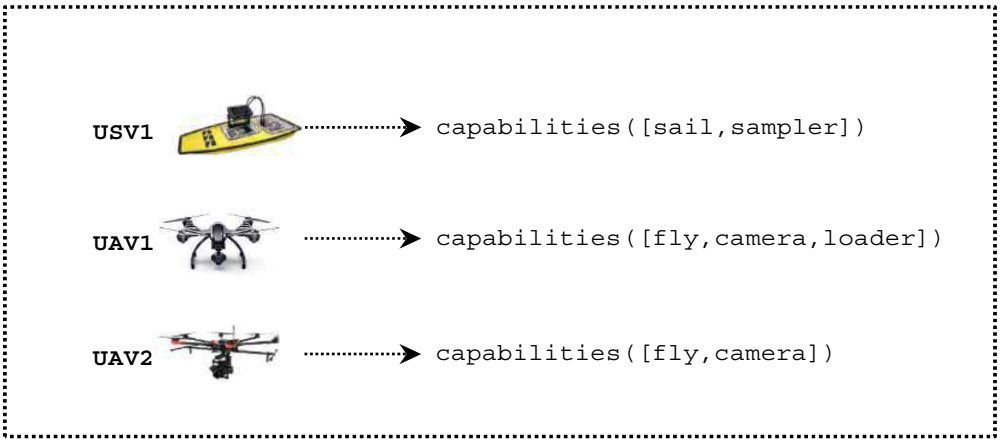


Figure 4.2 – Example of robots and their capabilities.

[1] In order to make the presentation shorter, we will use the term *sail* to mean any form of water navigation capability.

In the organisation, there are three possible roles to be played: $collector$, $deliverer$ and $mapper$. In order to play the $collector$ role, robots must have the capability to navigate on water and resources to collect water samples. For the $deliverer$ role, the robot must have the capability to fly and a loader to carry items. Finally, for the $mapper$ role, a robot must have the capability to fly and must have a camera to take pictures. The organisation publishes information about the roles on the blackboard to which all robots have access. Figure 4.3 represents the information about the roles.

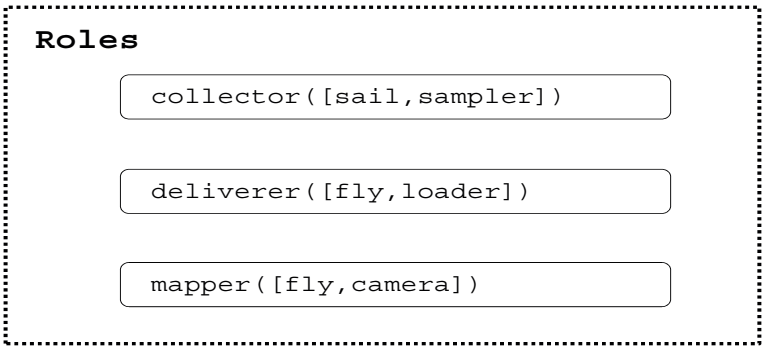


Figure 4.3 – Example of roles the their required capabilities.

Following our example scenario, the organisation publishes on the blackboard three tasks with their subtasks. Task $task1$ and $task2$ with three subtasks each, and a task $task3$ with two subtasks as available in Figure 4.4. The predicate representing each of the subtasks is composed of (and in this particular order): the subtask identifier, the action related to that subtask, the role required for a robot to perform that subtask, and the region where the task is to be performed. Note that as in the task $task3$, we can have a task with subtasks requiring different roles to be performed. Thus the role is at the subtask level and not at the task level.

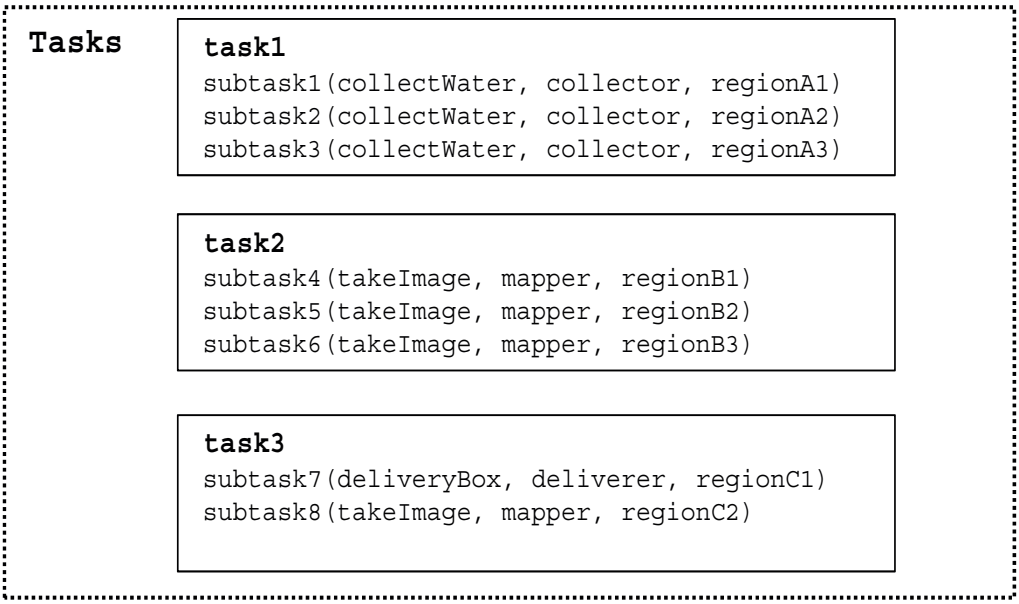


Figure 4.4 – Example of tasks and their required roles.

By perceiving the new set of tasks available, the robots begin the allocation process based on our mechanism. First, each robot will identify which roles it can play in the organisation. $USV1$ identifies it can play the $collector$ role while $UAV1$ identifies it can play the $deliverer$ and $mapper$ roles, and $UAV2$ identifies it can play only the $mapper$ role. Each robot is now able to identify the subtasks it can try to allocate based on the roles it can play. $USV1$ realises it can bid the three subtasks of $task1$ since it is able to play the role $collector$. $UAV1$ and $UAV2$ realise they can bid for the subtasks of $task2$ and for the subtask $subtask8$ of $task3$. Also, $UAV1$ can also bid for the $subtask7$ of $task3$ since it can also play the role of $deliverer$.

The robots then following our mechanism start biding for the subtasks they prefer, each one respecting the amount of subtasks it can allocate, which in this scenario may be related to the amount of energy (fuel) available to a robot. The process follows until all the robots agree on the allocated subtasks and the allocation process finishes after that (the way the end of the allocation process is determined is presented in section 4.2.4). The robots then can start the execution of the allocated subtasks, following their own plans and using the resources they have. Note that at any time the organisation may need to add new tasks or even new tasks might be discovered by the robots while executing the current tasks.

## 4.2 Decentralised Task Allocation

Our work aims at providing a decentralised solution for task allocation in environments with heterogeneous robots that are capable of carrying out various different tasks. We assume that an agent can have different capabilities. The capabilities of an agent can be related to its type of locomotion (e.g., the possibility of sailing or flying) or even to the resources available to the agent (i.e., the robot's payload such as cameras, sensors, etc.). An agent may play one or more roles. The roles are defined by the organisation the agents belong to, and each role is related to a set of capabilities that an agent needs to have in order to play that role. The organisation is also responsible for stating the tasks that are required for a given mission.

Remember that we are working with the type of tasks described in section 3.1: DS tasks, where all the subtasks need to be allocated to the same agent; CN tasks, where each subtask needs to be allocated to a different agent; CM tasks, where the subtasks can be allocated to different agents. The proposed mechanism allows us to work with those different type of tasks through the definition of the minimum and the maximum number of subtasks a robot must take from each type.

We also assume that each agent has a maximum number of subtasks that can allocate to itself. This constraint may be related, for example, to the amount of energy

(fuel) available to a robot. This may vary among robots as well as it may vary while the tasks are being carried out. The task allocation is based on utility values, and we consider that each agent is capable of calculating its own utility value for each task. As mentioned in the Section 2.3.2, the utility is a value that expresses how much a task contributes to the robot's objectives when executed by it. The utility function is dependent on the domain we are working, and the utility value can be quantified by combining several factors such as the energy spent to complete a task, distances, loads, among others. Furthermore, our approach assumes reliable communication. Next, we present aspects related to the definition of roles and tasks and how we use them in our task allocation mechanism. Then we continue with the presentation of the core algorithms of our mechanism.

### 4.2.1 Roles and Tasks Definition

The roles and tasks are defined by the organisation and announced on a blackboard. In this section, we describe our artifacts for announcing the roles and the tasks.

**Role definition:** in our multi-agent task allocation problem a role is related to a set of capabilities an agent must have in order to play that role. For instance, there could be a role $mapper$ requiring agents with the following capabilities to play the role:

$$role(mapper, [fly, camera]).$$

In Listing 4.1 we show the *roles* artifact for sharing information about roles. The *roles* artifact has two operations $announceRoleCapability$ and $removeRoleCapability$ which are used to announce and remove the capabilities for a role. The operation $announceRoleCapability$ receives two parameters as input, the role description and a capability description for that role and generate an observable property for that input. For instance, for the $mapper$ role mentioned above, the operation $announceRoleCapability$ would be executed twice:

$$announceRoleCapability(mapper, fly)$$
$$announceRoleCapability(mapper, camera)$$

The information defined in the observable properties generated by the operations will appear as beliefs for the agents who focus on the roles artifact as follow:

$$roleCapability(mapper, fly)$$
$$roleCapability(mapper, camera)$$

In the same way, the operation $removeRoleCapability$ would be executed twice for removing all the information about the $mapper$ role, or only once if only one of the capabilities is no more required for that role.

```
1  ...
2  @OPERATION void announceRoleCapability(String role, String capability){
3  try {
4  defineObsProperty("roleCapability",role, capability);
5  } catch (Exception ex){failed("role_announce_failed");}
6  }


9  @OPERATION void removeRoleCapability(String role, String capability){
10 try {
11 removeObsPropertyByTemplate("roleCapability",role, capability);
12 } catch (Exception ex){failed("role_remove_failed");}
13 }
14 ...
```

Listing 4.1 – Operations avaialable in the roles artifact.

**Task definition:** tasks are also announced by the organisation through an artifact.

Before explaining how the tasks are announced through the artifact, let us first explain how the tasks are defined using the example from Figure 4.5. Let us consider that the organisation needs agents to collect water for analyses in three different locations and decided to create a task with three subtasks for that. Also, consider that the organisation defines that the three subtasks can be executed by only one agent or at most three agents (see Figure 4.5.A). The information for each subtask in Figure 4.5.A is presented in this order: the action that needs to be performed, the role required to execute the subtask and a list of values (in brackets) which are related to the domain and are additionally needed to calculate the utility for the subtask.

The infomation available in Figure 4.5(A) can be mapped for the one presented in Figure 4.5(B). Every time the organisation wants to announce new tasks we call this new set of tasks as a job and give it an id ($jobId$). Suppose the organisation is going to announce at the same time the three tasks available in Figure 4.4. All the tasks would have the same $jobId$. The task and each of the subtasks also receive an identifier. Based on the MinAgents and MaxAgents values of the new task, in this example, the task is mapped for the type of task CM.

Thus each task has the following parameters that must be determined when a task will be announced through the artifact: jobId, taskId, taskType, [subtaskList].

**jobId:** is an identifier for the set of tasks which were announced together;

**taskId:** is the task identifier;

**taskType:** is the type of the task;

**subtaskList:** the list of subtasks for that task

Each subtask in the subtaskList also has a number of parameters: subtaskId, taskAction, role and [subtaskParams].

**subtaskId:** is an identifier for the subtask;

**taskAction:** is the action that must be performed;

**role:** is the role needed to perform that subtask;

**paramsSubtask:** the number of parameters here depends on the domain the task bellows. Values in this parameter can be related to, for instance, distance, execution time, weight and others. These values should be the ones necessary to calculate the utilities for the subtasks.

Based on the information available in Figure 4.5(B) the organisation announce the subtasks using the operations available in the *Blackboard* artifact (Listing 4.2). The operation $announceSubtask$ is used to announce the information necessary for the allocation process as shown in Figure 4.5(C). The operation $announceSubtaskParam$ is used to announce information necessary to calculate the utility values for the subtasks as shown in Figure 4.5(D).


Figure 4.5 – Example of task definition.

```
1  ...
2  @OPERATION void announceSubtask(String subTaskDescr, String TaskDescr, String job
       , String TaskType, String taskAction, String roleDescr){
3  try {
4  Literal lsubTaskDescr = createLiteral(subTaskDescr);
5  Literal lTaskDescr = createLiteral(TaskDescr);
6  Literal lJob = createLiteral(job);
7  Literal lTaskType = createLiteral(TaskType);
8  Literal ltaskAction = createLiteral(taskAction);
9  Literal lroleDescr = createLiteral(roleDescr);
10 this.defineObsProperty("subTask", lsubTaskDescr, lTaskDescr, lJob, lTaskType,
       ltaskAction, lroleDescr);
11 } catch (Exception ex){}
12 }

14 @OPERATION void announceSubtaskParam(String info, String subTask, String task,
       String job, String infoDescr){
15 try {
16 Literal lsubTask = createLiteral(subTask);
17 Literal lTask = createLiteral(task);
18 Literal lJob = createLiteral(job);
19 Literal linfoDescr = createLiteral(infoDescr);
20 this.defineObsProperty(info, lsubTask, lTask, lJob, linfoDescr);
21 } catch (Exception ex){}
22 }
23 ...
```

Listing 4.2 – Operations avaialable in the tasks artifact.

### 4.2.2 Algorithms for the Task Allocation Process

The proposed task allocation mechanism is based on algorithms that are executed by each agent in the organisation, characterising a decentralised solution. A general view of algorithms that constitute the core of the proposed mechanism is presented below. Note that during a mission new roles can be added or the existing roles can be updated regarding the required capabilities or a role can even be deleted from the roles list. Also, new tasks can be added during a mission. When the agents perceive a new set of tasks, they begin the allocation process by initially running the Algorithm 1.

**Starting the allocation process (Algorithm 1)**

The initial algorithm is Algorithm 1 which receives as input two parameters: the list of tasks to be carried out by the agents as currently available on the blackboard; and the current list of roles within the organisation.

In a new allocation process, the first step for an agent is to identify the roles it can play based on the current list of roles available in the organisation (line 5 shows the call to the Algorithm 2 which takes care of that). Next, the agent selects only the tasks that are compatible with the roles that it can play (line 6 shows the call to the respective function in Algorithm 3). Knowing the tasks that the agent can perform, it calls Algorithm 4 (line 7), which updates for each task the minimum and the maximum number of subtasks that a robot must take according to the type of the task. Line 8 shows the call to the function responsible for calculating the utilities for the possible subtasks. The utility function is dependent on the domain the agents are working on and can combine several factors.

Finally, it calls Algorithm 5 (line 9), which starts the bidding process for the tasks that can be allocated to that agent. The first time the Algorithm 5 is called, the input parameter $allocatedSubtasks$ is empty.

**Algorithm 1**
`startAllocation`($blackboardTasks$,$organisationRoles$)

1: $allocatedSubtasks = \emptyset$;
2: $candidates = \emptyset$;
3: $taskList \leftarrow blackboardTasks$;
4: $roleList \leftarrow organisationRoles$;
5: $possibleRoles \leftarrow getPossibleRoles(roleList)$;
6: $possibleTasks \leftarrow getPossibleTasks(taskList, possibleRoles)$;
7: $possibleTasks \leftarrow getMinMaxTaskType(possibleTasks)$;
8: $calculateUtilities(possibleTasks)$;
9: $taskAllocation(possibleTasks, allocatedSubtasks)$;

### Identifying the possible roles (Algorithm 2)

The purpose of this algorithm is to identify the roles the agent can play based on its own capabilities and on the current list of roles available in the organisation.

Algorithm 2 receives as input the list of all roles currently defined within the organisation with the required capabilities a robot must have to play each of the roles. The algorithm goes through the $roleList$ received as input and for each role in that list it checks if the agent has all the required capabilities. Each role the agent is able to play is added to the list of $possibleRoles$ which is the output from the algorithm. The identification of the possible roles is performed whenever new tasks are perceived by the agents because we consider that in some scenarios it may be possible that the roles may change during the mission.

**Algorithm 2**
`getPossibleRoles`($rolesList$)

```
1: Let agentCapabilities be the list of agent's capabilities;
2: possibleRoles = ∅;
3: for all role r_k in rolesList do
4:     validRole ← true
5:     for all capabilityRequired in r_k do
6:         if capabilityRequired not in agentCapabilities then
7:             validRole ← false
8:         end if
9:     end for
10:    if validRole = true then
11:        possibleRoles.add(r_k)
12:    end if
13: end for
14: return possibleRoles
```

**Identifying the possible tasks (Algorithm 3)**

This algorithm is used by the agent to identify the subtasks it can carry out based on the roles it can play (each subtask is associated to a role which an agent must be able to play to perform that subtask).

Algorithm 3 receives as input the list of all blackboard tasks that need to be carried out as well as the list of the roles the agent is able to play within the organisation. The algorithm goes through each task and for each task it goes through each subtask identifying if that subtask could possibly be allocated to the agent, i.e. if the agent is able to play the role associated to that subtask. The output of this algorithm is a list of tasks in which each subtask is updated with respect to whether or not it can be allocated by the agent.

**Algorithm 3**
`getPossibleTasks`($blackboardTaskList, possibleRoles$)

```
1: taskList ← blackboardTaskList;
2: for all task t_j in taskList do
3:     for all subtask st_k in t_j do
4:         if st_k.role in possibleRoles then
5:             st_k.possible ← true;
6:         else
7:             st_k.possible ← false;
8:         end if
9:     end for
10: end for
11: return taskList
```

**Update min/max subtasks by task type (Algorithm 4)**

The purpose of this algorithm is to update the minimum and the maximum number of subtasks that a robot must take from each task according to the type of the task.

Algorithm 4 receives as input the list of all tasks the agent can execute. It goes through each task and updates the minimum and the maximum number of subtasks that a robot must take from that task. Minimum and maximum values are based on the type of the task, and we assume they are defined a priori to the execution of the mechanism. We consider that minimum and maximum values can be expressed with a number or with the letter $N$ to represent all subtasks.

The algorithm goes through each task $t_j$ from the list of $possibleTasks$ performing the following: first, based on the type of the task $t_j$, the algorithm identifies the minimum and the maximum number of subtasks that a robot must take from that type of task (lines 2 to 3). The number of subtasks that task $t_j$ has is identified as $N_j$ (line 4). Next, the minimum and the maximum values for the task $t_j$ are identified from line 5 to line 14. If the $minType$ for the type of task is the letter $N$ then the minimum number of subtasks of $t_j$ is $N_j$, i.e. the agent must take all the subtasks of $t_j$. Otherwise, $minType$ is a number representing the minimum number of tasks should be taken.

Also, if the $maxType$ for the type of task is the letter $N$ then the maximum number of subtasks of $t_j$ is $N_j$, i.e. the agent can take all the subtasks of $t_j$. Otherwise, $maxType$ is a number representing the maximum number of tasks could be taken. The output from this algorithm is the list of $possibleTasks$ updated with the minimum and the maximum number of subtasks that a robot must take from each of the tasks.

**Algorithm 4**
`getMinMaxTaskType`($possibleTasks$)

1: **for all** $task\ t_j$ **in** $possibleTasks$ **do**
2: $minType \leftarrow taskTypes.getMin(t_j.type)$;
3: $maxType \leftarrow taskTypes.getMax(t_j.type)$;
4: $N_j \leftarrow t_j.subTasks.count$
5: **if** $minType = N$ **then**
6: $t_j.minSubTask \leftarrow N_j$;
7: **else**
8: $t_j.minSubTask \leftarrow minType$;
9: **end if**
10: **if** $maxType = N$ **then**
11: $t_j.maxSubTask \leftarrow N_j$;
12: **else**
13: $t_j.maxSubTask \leftarrow maxType$;
14: **end if**
15: **end for**
16: **return** $possibleTasks$

**Performing the task allocation (Algorithm 5)**

Algorithm 5 receives as input the list of tasks that can be allocated ($possibleTasks$) to agent $r_i$ (which is running the algorithm), and the list of subtasks already allocated in the current allocation process ($allocatedSubtasks$) – empty the first time the algorithm is called.

Initially, the algorithm checks if the agent still has the capacity to allocated more subtasks by comparing if the current load for the allocated subtasks ($currentLoad$) so far is lower than its load capacity ($loadCapacity$ – the task limit). If so, it begins the analysis of all possible tasks in order to identify the tasks that can still be allocated (lines 5 to 13). We use $l'_i$ to refer to the difference between $loadCapacity$ and $currentLoad$.

The analysis starts by identifying the list of candidate subtasks, which is based on the minimum and the maximum number of subtasks the agent can take from each task (line 5 shows the call to the Algorithm 6 which takes care of that). The algorithm follows by filtering the list of candidate tasks using the Algorithm 7 (called at line 6). The algorithm then uses the filtered candidates list ($filteredCandidates$) and the current load capacity of the agent ($l'$) as input to select the best candidates, i.e. the candidate subtasks that the agent choose to allocate (line 7 shows the call to the respective function). The $getBestCandidates$ is a version of an algorithm for the Knapsack problem (more detail in Section 4.2.3). The $getBestCandidates$ function returns a list with the best candidates subtasks selected for allocation.

**Algorithm 5**
`taskAllocation`($possibleTasks$, $allocatedSubtasks$)

1: Let $loadCapacity$ be the agent's max load to allocate new concurrent subtasks;
2: $currentLoad \leftarrow allocatedSubtasks.load$;
3: $l' \leftarrow loadCapacity - currentLoad$;
4: **if** $l' > 0$ **then**
5: $candidates \leftarrow getCandidates(possibleTasks, l')$;
6: $filteredCandidates \leftarrow filterCandidates(candidates, l')$;
7: $bestCandidates \leftarrow getBestCandidates(filteredCandidates, l')$;
8: $allocatedSubtasks \leftarrow allocatedSubtasks \cup bestCandidates$;
9: $calculateBids(bestCandidates)$;
10: **for all** $subtask\ st_k$ **in** $bestCandidates$ **do**
11: $bidList \leftarrow bidList \cup [st_k.id, st_k.bid, agentId]$;
12: $updateSubtaskOwner(st_k.id, st_k.bid, agentId)$;
13: **end for**
14: $communicateBids(bidList)$ //to all other agents;
15: **end if**

The best candidates are then added to the list of allocated subtasks (line 8). Next, the algorithm calculates the bids for the subtasks in the $bestCandidates$ list (line 9 shows the call to the Algorithm 8). Next, the identifier and bid values for each subtask

$st_k$ in $bestCandidates$ as well as the identifier of the agent are added to the list of bids $bidList$ (line 10 to 13). Note that the list of bids has bids only for the best subtasks selected in this iteration of Algorithm 5. In the same loop the algorithm updates locally the owner and the winner bid value for each subtask. Finally, the agent communicates the list of bids $bidList$ to all other agents (line 14).

**Getting the candidate subtasks (Algorithm 6)**

This algorithm is used to identify the list of candidate subtasks based on the minimum and the maximum number of subtasks the agent can take from each task. The algorithm receives as input the list of tasks the agent is able to execute ($possibleTasks$) and the current load capacity of the agent ($l'$). For each task $t_j$ in $possibleTasks$, the algorithm identifies the number of subtasks the agent can/must select as candidates for allocation. In order to get this value, the algorithm first identifies the minimum load required for the task (line 4). Then, it checks if the agent has the capacity to select the minimum load of subtasks required by task $t_j$ (line 5). After that, based on the maximum number of subtasks that the agent can allocate from task $t_j$ and the ones already allocated, the algorithm verifies if there are still subtasks of task $t_j$ that can be allocated (line 9).

Next, the algorithm selects as candidates the $nToAlloc$ best subtasks from task $t_j$ limited to the current load capacity $l'$ (line 12). The choice of candidates is carried out based on the utility of each subtask of that task. After analysing each task, the algorithm adds the subtasks selected as candidates to the list of candidate subtasks (line 15). Finally, the output of the algorithm is a list with the subtasks of each task selected as candidates (line 17).

**Filtering the candidate subtasks (Algorithm 7)**

The idea is to filter the number of subtasks that will be used in the next step of the Algorithm 5 based on the load of the subtasks.

The algorithm receives as input two parameters: the list of candidate subtasks ($candidates$) and the current load capacity of the agent ($l'$). The first step is to identify the distinct loads available in the subtasks from the $candidates$ list (line 2 shows the call to the function which takes care of that). Then, the algorithm goes through each distinct load (line 3 to 16) performing the following: first it adds to the $loadCapabilities$ list all the subtasks from $candidates$ list which have the load $l_i$ being processed in the iteration and then sorts that list from highest to lowest based on utility values (line 4 to line 5).

Then while the $loadCapabilities$ list is not empty and while the sum of the loads ($loadTotal$) for the subtasks added to the $filteredCandidates$ list is lesser than the current

**Algorithm 6**
`getCandidates`($possibleTasks$, $l'$)

```
1: candidates = ∅;
2: for all task t_j in possibleTasks do
3:     candidatesT = ∅;
4:     taskMinLoad ← getMinLoad(t_j);
5:     if l' >= taskMinLoad then
6:         minT ← t_j.minSubTask;
7:         maxT ← t_j.maxSubTask;
8:         nAllocatedT ← allocatedSubtasks.countSubtasksFrom(t_j);
9:         if maxT > nAllocatedT then
10:            toAllocT ← maxT − nAllocatedT;
11:            nToAlloc ← min[l', toAllocT];
12:            candidatesT ← getTaskCandidates(t_j.subTasks, nToAlloc, l');
13:        end if
14:    end if
15:    candidates ← candidates ∪ candidatesT;
16: end for
17: return candidates
```

load capacity $l'$, the algorithm goes through the $loadCapabilities$ list performing the following: for each subtask $st_k$ in $loadCapabilities$, it calculates the bid for the subtask and then verifies if the bid is higher than the current bid winner. If so, it adds the subtask to the $filteredCandidates$ list and sums the load value of that subtask to $loadTotal$. Then the first subtask is removed from the $loadCapabilities$ list, and the steps are repeated. The output for this algorithm is a list of subtasks filtered from the $candidates$ list.

The filtered candidates list ($filteredCandidates$) are then used by the Algorithm 5 to select the best candidates, i.e. the candidate subtasks that the agent choose to allocate (function called at line 7 of Algorithm 5). The $getBestCandidates$ function is a version of an algorithm for the Knapsack problem (more detail in Section 4.2.3), which returns a list with the best candidates subtasks selected for allocation.

### Calculate bids (Algorithm 8)

The purpose of this algorithm is to calculate the bid values for the subtasks that the agent is trying to allocate.

The input for this algorithm is the list of the subtasks that were selected for allocation ($bestCandidates$). The algorithm goes through each subtask $st_k$ in the $bestCandidates$ list performing the following: first the algorithm identifies the next best subtask for the subtask $st_k$ – $subtaskNext$ (line 2). The next best subtask is the one that would be selected if the agent was not possible to select $st_k$. The subtasks in the $allocatedSubtasks$ list are not eligible to be the next best subtask since they are currently allocated to the agent.

Next, the algorithm identifies the utility value and the current winner bid for the $subtaskNext$ (line 3 to 4). Then the algorithm calculates the bid value for the $st_k$ subtask to be sent to the other agents. This calculation was inspired by the bid calculation formula introduced in [LCS15a]. The bid for a subtask is its utility value minus the amount that would be lost if the next best subtask were taken instead. Line 5 refers to the calculation of the bid for the subtask $st_k$.

**Algorithm 7**
`filterCandidates`($candidates$, $l'$)

```
1: filteredCandidates = ∅;
2: loads ← selectDistinctLoads(candidates)
3: for all load l_i in loads do
4:     loadCandidates ← getCandidatesWithLoad(candidates, l_i);
5:     loadCandidates.sortByUtility;
6:     loadTotal = 0
7:     while loadTotal ≤ l' and loadCandidates ≠ ∅ do
8:         st_k ← loadCandidates.getFirstSubtask;
9:         st_k.bid ← calculateBid(st_k);
10:        if st_k.bid > currentWinnerBid(st_k) then
11:            filteredCandidates.add(st_k);
12:            loadTotal ← loadTotal + l_i;
13:        end if
14:        loadCandidates.removeFirstSubtask;
15:    end while
16: end for
17: return filteredCandidates;
```

**Algorithm 8**
`calculateBids`($bestCandidates$)

```
1: for all subtask st_k in bestCandidates do
2:     subtaskNext ← getNextBestSubtask(st_k);
3:     utilityNext ← getUtility(subtaskNext);
4:     winnerBidNext ← getCurrentWinnerBid(subtaskNext);
5:     st_k.bid = st_k.utility − (utilityNext − winnerBidNext) + 1;
6: end for
```

**Processing the received bids (Algorithm 9)**

Each agent processes the bids received from other agents by executing Algorithm 9. This algorithm is only triggered when the Algorithm 5 is not running in the agent. It means that while Algorithm 5 is running the agent may receive bids from one or more agents before start processing the bids. The $bidsQueue$ input is the list of bids received from other agents.

When the agent starts processing the bids it goes through each list of bids received from other agents ($bidList$) available in the $bidsQueue$ list performing the follow-

ing: for each $bid$ in the $bidList$ it checks if the received bid is greater than the current winner bid value for the subtask and, if so, it updates the owner and winner bid value for that subtask (line 6 to line 7). Note that the bids are processed in the order they were received. Next, if the bid is related to a subtask the agent allocated to itself, then the agent has to remove that subtask from its allocated subtasks (line 8 to line 9). After the bids in $bidList$ were processed, the algorithm calls Algorithm 11 to update locally the status of the agent who sent the bids in the $bidList$ (line 15).

After processing all the $bidList$ in $bidsQueue$ the algorithm verifies if the agent still has available load to allocate other subtasks. If so, it calls Algorithm 5 in order to try to allocate others subtasks.

Algorithms 5 and 9 are repeated until the agents agree on the allocation, that is, until the self-allocated subtasks do not undergo any further modifications. Section 4.2.4 provides more details about how is determined the end of the allocation process.

**Algorithm 9**
`processBids`($bidsQueue$)

```
1: if bidsQueue ≠ ∅ then
2:     for all bidList in bidsQueue do
3:         agentId ← bidList.agentId;
4:         agentLostBid ← false;
5:         for all bid in bidList do
6:             if bid.value > subtask.bidValue then
7:                 updateSubtaskOwner(bid.subtask, bid.value, agentId);
8:                 if bid.subtask in allocatedSubtasks then
9:                     allocatedSubtasks.remove(subtask)
10:                end if
11:            else
12:                agentLostBid ← true;
13:            end if
14:        end for
15:        updateAgentStatus(agentId, agentLostBid);
16:    end for
17: end if
18: Let loadCapacity be the agent's max load to allocate new concurrent subtasks;
19: currentLoad ← allocatedSubtasks.load;
20: if (loadCapacity − currentLoad) > 0 then
21:    allocatedSubtasks ← getCurrentAllocatedSubtasks();
22:    possibleTasks ← getCurrentPossibleTasks();
23:    taskAllocation(possibleTasks, allocatedSubtasks);
24: end if
```

### 4.2.3 Selecting the Best Subtasks Using a Knapsack Algorithm

In Algorithm 5, we call the *getBestCandidates* function (line 7) for selecting the best subtasks for allocation to an agent from its candidates list. Our *getBestCandidates* function is an algorithm for the Knapsack problem. In this section, we explain the basic idea behind it.

The basic Knapsack problem consists in placing items with different weights and values inside a knapsack, trying to maximise the total value of the items in the knapsack while respecting the maximum weight it can take. Analogous to the Knapsack problem, the agent's load capacity for subtasks corresponds to the weight limit of the knapsack; the amount of load that a subtask will occupy in the agent's load capacity corresponds to the weight of an item; and the utility value of a subtask corresponds to the value of an item (see Table 4.1).

In order to explain the use of the knapsack algorithm in this paper, let us consider an agent that has 20 as load capacity to allocate subtasks. Consider also the task samples available in Table 4.1. Recall that for a decomposable simple task (such as DS1 and DS2 in Table 4.1), the agent must take all or none of its subtasks. Task DS1, for example, has three subtasks that must all be taken by the same agent. To deal with this type of task while selecting the best subtasks, we consider those three subtasks as one, summing up the utility values of each subtask, and also summing up the load of each subtask, that is, their weight. Thus, the task DS1 would be considered as a task with the load equal to 15 (it would occupy fifteen load space of the agent's load capacity) and with a total utility value of 14 (the sum of the utilities of the individual subtasks).

Since the subtasks from CN and CM type of tasks can be independently allocated, they are individually considered here. Thus, each subtask will occupy only its load in the agent's load capacity, and its utility value will also be considered individually.

Table 4.1 – Example of Candidate Subtasks List

| Task | DS1 | | | DS2 | | CN1 | | CM1 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Subtask (item) | st1 | st2 | st3 | st4 | st5 | st6 | st7 | st8 | st9 | st10 |
| Load (weight) | 4 | 5 | 6 | 12 | 5 | 2 | 10 | 6 | 4 | 5 |
| Utility (value) | 4 | 3 | 7 | 8 | 3 | 6 | 7 | 8 | 1 | 6 |

At each iteration of Algorithm 5, the agent runs the knapsack algorithm (called on line 7) to select the best subtasks from the list of candidate subtasks up to its limit. That is, the algorithm should select subtasks such that the sum of their utilities is maximised while respecting the limit of subtasks the agent can take on at any given time.

Although it seems prohibitive to solve knapsack problems repeatedly and for each agent, it should be noted that the previous steps of Algorithm 5 ensure that only a

typically small selection of tasks take part in this step of the overall allocation process, and an agent typically have small task limits. In other words, even if a large number of subtasks is available, the process will filter the subtasks that will be sent to the *getBest-Candidates* function (which calls our Knapsack algorithm) and only a relatively small number of tasks will be considered for a small task limit.

### 4.2.4 Determining the End of the Allocation Process

In decentralised task allocation mechanisms, where agents place bids for the tasks they want to allocate, one of the problem to solve is for each agent to know when the other agents finished sending bids, hence determining that task allocation process is complete.

Some solutions to this problem, such as in [LCS15a], use the following approach to determine the end of the allocation process. At each iteration the agents send their list of bids for all tasks they wish to allocate, that is, if an agent wishes to allocate five tasks, a message with five bids is sent to the other agents at each iteration, even though there has been no change in the allocated tasks. Sending messages with the same content is used to control the end of the allocation process, that is, when the bids of all agents are the same for a number of iterations, it means that the task allocation has ended. Due to this feature, at each iteration a large number of messages with size proportional to the number of tasks each agent is allocating to itself is sent to all other agents. This can impact the communication infrastructure and unnecessarily waste resources, which can be crucial in a solution for real-world environments. [CBH09] also uses a similar approach.

Below, we describe our approach to identify the end of the allocation process. In our approach, each agent internally stores the winner for each subtask and its bid value in a list (the *SubtaskWinner* list). Each agent also keeps an agent status list with each agent participating in the allocation process (*AgentBidStatus* list). Consider an agent $a_i$ and its *SubtaskWinner* and *AgentBidStatus* lists.

When an agent $a_i$ receives bids from another agent $a_j$ for the first time, it will add that agent to the *AgentBidStatus* list, and in subsequent bids from that same agent $a_j$, $a_i$ will update the value associated with $a_j$ (see Algorithm 11). In our approach, when an agent $a_i$ processes the list of bids received from each of the other agents, it adds/updates to that agent one of the following values in the *AgentBidStatus* list:

- *false* – set this value when agent $a_j$ won all the subtasks for which it has bid. It means that the agent does not need to send further bids for now. The update in the *AgentBidStatus* list is performed by calling Algorithm 11 on line 15 of Algorithm 9;

- *true* – set this value to indicate that agent $a_i$ needs to wait for another bid from that agent $a_j$. This value is set in the following cases:
  - when $a_j$ does not win all of the subtasks for which it has placed bids, that is, at least one of the subtasks has already received a higher bid from another agent. For example, if agent $a_i$ is processing a bid list with bids for three subtasks, and it realises that the agent $a_j$ only won two of them, it means that $a_j$ will need to send further bids. The update in the *AgentBidStatus* list is also performed by calling Algorithm 11 on line 15 of Algorithm 9;
  - when $a_j$ is outbid by another agent, i.e., when it loses one of its allocated subtasks. It means that $a_j$ will need to send further bids. That can be checked when agent $a_i$ is processing bids from other agents and then update the *AgentBidStatus* list through Algorithm 10 called on line 7 of Algorithm 9, which calls Algorithm 11;
  - when $a_i$ allocates a subtask to itself that outbids $a_j$. It means that $a_j$ will need to send further bids. This update is done when $a_i$ calls Algorithm 10 on line 12 of Algorithm 5, , which calls Algorithm 11;

When an agent is outbid, it will always try to select another subtask to bid for. However, when that agent is not able to select another subtask, it will send a *done* message. When agent $a_i$ receives a *done* message from an agent $a_j$, agent $a_i$ will set its value for $a_j$ as *false* in its *AgentBidStatus* list, meaning that $a_j$ does not need to send another bid for now. That value may change if that agent is later outbid on another subtask.

Regardless of the value set in the *AgentBidStatus* list, each subtask a bidding agent wins will be associated to it in *SubtaskWinner* list. Also, when agent $a_i$ allocates a subtask to itself, it will be associated with that subtask in the *SubtaskWinner* list.

In order to control the end of the whole allocation process, we do as follows. When an agent has no more bids to process, it will check internally if the number of agents in the *AgentBidStatus* list is equal to the number of agents considered in the allocation process and if the values for all the agents is *false*, which means all agents sent their bids or *done* messages, and subtasks were allocated in accordance with the bids. These steps are performed by all the agents in the allocation process.

In summary, at the beginning of the allocation process, each agent will send a bid list for all subtasks that it wishes to allocate. Next, only the agents that were outbid will send bids for other subtasks again, but only for the newly selected subtasks. If the agent is not able to select another subtask, it will send a *done* message. That reduces the number and size of messages each agent needs to send to other agents, and still allows a precise procedure to check for termination. Our approach, however, has a higher storage cost, since it stores the winner of each subtask and the status of each agent

related to the bids they sent. Furthermore, like most other auction-based approaches, our approach assumes reliable communication among agents.

**Algorithm 10**
`updateSubtaskOwner`($subtask, bidValue, agentId$)

1: **if** $(subtask.owner \neq null)$ **and** $(subtask.owner \neq agentId)$ **then**
2: $updateAgentStatus(subtask.owner, true)$;
3: **end if**
4: $subtask.owner \leftarrow agentId$;
5: $subtask.winnerBid \leftarrow bidValue$;

**Algorithm 11**
`updateAgentStatus`($agentId, status$)

1: **if** $agentId$ **in** $AgentBidStatus$ **then**
2: $AgentBidStatus.update(agentId, status)$;
3: **else**
4: $AgentBidStatus.add(agentId, status)$;
5: **end if**

### 4.2.5 Coping with Partially Allocated Tasks

The algorithms previously described produced good results in the performed experiments, especially when the total capacity of the agents was greater than or equal to the total number of subtasks that need to be allocated. However, when the number of subtasks is greater than the total capacity of the agents, the final allocation of the algorithms can result in tasks not completely allocated, that is, tasks in which at least one subtask was not allocated to any agent. The algorithms can result in tasks not completely allocated also when none of the agents have the capability required to perform one of the subtasks of a task.

For example, if a task is composed of four subtasks, the allocation process may result in three subtasks allocated while one of them is not. This situation can occur because the agents always try to allocate the subtasks with higher utility values. Thus, if some subtasks of a compound task have higher utilities they probably will be allocated while the ones with lower values may end up not allocated. This may result in a compound task partially allocated, which should not be allowed. Thus, in order to avoid tasks not completely allocated, at the end of the allocation process previously described, it is necessary to perform a few more steps which are performed by Algorithm 12.

**Handling partial allocated tasks(Algorithm 12)**

First, the algorithm identifies the tasks that are not completely allocated, that is, the tasks in which at least one of their subtasks was not allocated to any agent (line 2 shows the call to the Algorithm 13 responsible for this process). Knowing which tasks are partially allocated, each agent checks whether it has any of their subtasks in its allocation list, and then removes those subtasks from its allocation (line 3 refers to this function in Algorithm 14).

After this step, there may be some completely unallocated tasks. Since the agents have removed previously allocated subtasks, they may now have space for new allocations. So we run our allocation process again, but one task at a time, that is, we run the process to fully allocate one task, then move on to the next task until all tasks have been allocated. The idea of allocating one task at a time is due to the fact that the preferences of each agent tend to be the same as those that resulted in partially allocated tasks.

The order in which the tasks will be allocated is relevant since the preference order for the allocation may be different for the agents and may impact the quality of the final allocation. Thus, the agents need to reach an agreement on the order in which the tasks will go through this stage of the allocation process. Thus, in this work, we use a social-choice algorithm based on voting to achieve such an agreement; in particular, we use Borda count as the voting method to decide the order in which the tasks will be allocated.

**Algorithm 12**
`handlePartialAllocatedTasks`

1: $taskList \leftarrow getTasks()$;
2: $partialTasks \leftarrow identifyPartialAllocated(taskList)$;
3: $removePartialTasks(partialTasks)$;
4: $preferenceList \leftarrow generatPreferenceOrder(partialTasks)$;
5: $sendPreferenceList(PreferenceList)$;

In Borda count, each voter submits a full preference ordering on the candidates. Each place in the ordered list provides points to the candidates. The first candidate receives $n - 1$ points, the next receives $n - 2$, and so on (where $n$ is the number of candidates). The function $generatePreferenceOrder$ in line 4 represents the function responsible for generating the preference ordering list. The global ordering is determined by the sum of points from all the voters [Con10]. Simply put, in our use of Borda count the agents submit their order of preference to all other agents (this is represented in line 5).

**Identifying partially allocated tasks(Algorithm 13)**

The algorithm receives as input a list with all the tasks being allocated ($taskList$). For each task in $taskList$ the algorithm goes through each subtask verifying if the subtask was allocated to some agent. If some subtask was not allocated to any agent the task is set as partial (line 4 to 8). After verifying all the subtasks the algorithm checks if the task is set as partial and then add that task to the $partialTasks$ list (line 9 to 11). It means that the task was not completely allocated. The output from this algorithm is a list of tasks that were not completed allocated.

**Algorithm 13**
`identifyPartiallyAllocated`($taskList$)

```
 1: partialTasks = ∅;
 2: for all task t_j in taskList do
 3:     partialT_j = false;
 4:     for all subtask st_k in t_j do
 5:         if st_k.owner = null then
 6:             partialT_j = true;
 7:         end if
 8:     end for
 9:     if partialT_j = true then
10:         partialTasks = partialTasks ∪ t_j;
11:     end if
12: end for
13: return partialTasks
```

**Removing partially allocated tasks(Algorithm 14)**

The algorithm receives as input the list with the tasks that were not completely allocated ($partialTasks$). For each task in $partialTasks$ the algorithm goes through each subtask performing the following: since the subtask will no more be allocated to any agent, the algorithm first updates the owner and the winner bid values for the subtask (line3). Then it checks if the subtask is in the list of allocated subtasks ($allocatedSubtasks$) and then remove the subtask from that list.

**Processing agents' preferences(Algorithm 15)**

The algorithm receives as input the preference lists (voting) received from the other agents. These preference lists can have different tasks and different lengths for each agent, because some subtasks may not be possible to be executed by some agents due to the roles required by the subtask. All tasks not included in the list by an agent are assumed to be equally least preferred by that agent.

**Algorithm 14**
removePartialTasks($partialTasks$)

```
1: for all task t_j in partialTasks do
2:     for all subtask st_k in t_j do
3:         updateSubtaskOwner(st_k, null, null);
4:         if st_k in allocatedSubtasks then
5:             allocatedSubtasks.remove(st_k);
6:         end if
7:     end for
8: end for
```

Based on the voting lists received, each agent individually calculate the global ordering for the allocation process (line 3 shows the call to the function $computePreferences$ which takes care of that). The tasks will be allocated in that computed ordering. For each task being processed the algorithm first verifies if the agent has capacity to allocate more subtasks (line 6). The agent then start providing bids on the subtasks of that task and, after receiving the bids, the winner for each subtask is known. The allocation is performed by running Algorithm 5 (line 9).

After this process, the agent checks whether the task has been completely allocated or not. If it is fully allocated, the agent maintains its allocated subtasks, otherwise they are considered not allocated (line 13 calls the Algorithm 14 which takes care of that).

Then the agents start bidding on the subtasks of the next task in the global ordering previously defined and this is repeated until all tasks have been processed. At the end of this process, there may still be tasks that were not allocated. This may happen because the agents have no space to allocate more subtasks or do not have the capability required to perform one of the subtasks of a task.

**Algorithm 15**
processPreferences($preferenceLists$)

```
1: Let loadCapacity be the agent's max load to allocate new concurrent subtasks;
2: allocatedSubtasks = ∅;
3: tasksOrder ← computePreferences(preferenceLists);
4: for all task t_j in tasksOrder do
5:     currentLoad ← allocatedSubtasks.load;
6:     if (loadCapacity − currentLoad) > 0 then
7:         allocatedSubtasks ← getCurrentAllocatedSubtasks();
8:         possibleTasks ← getCurrentPossibleTasks(t_j);
9:         taskAllocation(t_j, allocatedSubtasks);
10:    end if
11:    partialTask ← identifyPartialAllocated(t_j);
12:    if (partialTask ≠ ∅) then
13:        removePartialTasks(partialTasks);
14:    end if
15: end for
```

# 5. EVALUATION

This chapter presents the evaluation of the proposed mechanism. First, we compare the performance of our mechanism with the optimal solution. We use the GLPK (GNU Linear Programming Kit) [Mak16] to obtain (centralised) optimal solutions for comparison with our results. We also compare the performance of our mechanism with other solutions. First, we compared with the Iterative Consensus-Based Auction Algorithm – ICBAA [CBH09] and the Sequential Single-Item Auction algorithm – SSIA [KKT10]. Then we compare our mechanism with two other task allocation approaches that handle only one of the types of tasks in our approach, more specifically atomic tasks: a task allocation algorithm (we call it TAA) used in [GKZ17] and a role-based task allocation (we call it RBTA) available in [GA15]. Finally, we present the impact of our task allocation approach in the execution of tasks in a complex scenario like the flooding disaster scenario, first describing some features of the simulator for the evaluation, and then we present the results of the evaluation. All experiments were conducted on a computer with the following specification: Intel(R) Core(TM) i5-2520M, 2.50GHz, 4 GB of memory, Windows 8 64-bit operating system, and Java 8.

## 5.1 Comparison with the optimal solution

In this section, we present a comparison of the performance results of our mechanism with the optimal solutions obtained using GLPK [Mak16].

### 5.1.1 Evaluation Measures

In this section, we describe the evaluation metrics used to analyse the performance of our mechanism with the optimal solution.

- **Performance:** by performance we mean the overall utility obtained by all the agents to take on all the subtasks that they can, i.e. the sum of the utilities obtained by the individual agents;

- **Coefficient of variation:** the coefficient of variation (standard deviation divided by the mean) was used as a measure of dispersion (i.e., the amount of variability relative to the mean). The lower the coefficient of variation, the more homogeneous the data, that is, the dispersion in the data is smaller;

- **Number of bid messages:** the number of bid messages sent by our mechanism is also measured to assess the impact of the different variations on the network traffic;
  - **Average number of bid messages per subtask:** it is the average number of bid messages provided by the agents for each of the subtasks;
  - **Average number of bid messages by agent:** it is the average number of bid messages placed by each agent during the allocation process;

### 5.1.2 Simulation settings

The simulations were run by varying a single parameter at each setting (Table 5.1). In all simulations, the subtasks of different types of task (CN, CM, DS) were uniformly distributed. Also, for each agent, we randomly selected the utility values for each subtask from the utility range in the respective setting. For settings 1 to 4, we ran simulations with the total capacity of the agents greater than or equal to the total number of subtasks. The results for each variation in these simulations were averaged over one hundred iterations each.

There were also simulations where we considered agents with capabilities to play any role and thus able to carry out any task, and there were also simulations where we varied the number of capabilities from 1 to 4 for each agent. In that case, some agents may not be able to play some roles, thus limiting the tasks they are able to carry out. For setting 5, the simulations were run with more subtasks than the total capacity of the agents. Thus, after the first part of task allocation, we may have tasks partially allocated and others completely unallocated, which the agents will try to allocate again, using the approach described above. The average results for these simulations were calculated from twenty iterations for each variation.

Table 5.1 – Settings used in the simulations.

| Setting | Varying | Agents | Subtasks | Limit | Utility range |
|---|---|---|---|---|---|
| 1 | agents | 5, 10, 15, 20, 25, 30, 35 | 24 | 5 | 1-6 |
| 2 | subtasks | 10 | 15,30,45,60 | 7 | 1-15 |
| 3 | limit | 5 | 24 | 6, 8, 10, 12, 14, 16, 20, 24 | 1-6 |
| 4 | utility | 10 | 42 | 6 | 1-6, 1-12, 1-24, 1-48 |
| 5 | subtasks | 3 | 21, 28, 35 | 6 | 1-6 |

### 5.1.3 Varying the Number of Agents (Setting 1)

In order to understand the impact of varying the number of agents, these simulations were performed using the values shown in Setting 1 of Table 5.1. For comparison with the optimal results, the simulations were run with 5, 10, 15, 20, 25, 30, and 35 agents, with 5 as the limit on the number of tasks to be allocated to each agent. The number of subtasks available to be allocated in these simulations was 24 (including subtasks of CN, CM, and DS task types). For each agent, we randomly selected the utility values from a range of 1 to 6 for each subtask.

First, we performed simulations considering agents with capabilities to play any role, that is, the agents are able to carry out any task. Figure 5.1 shows the results of these simulations. Figure 5.1.a shows that the performance of the proposed solution improves and is closer to the optimal solution (100%) as we increase the number of agents.

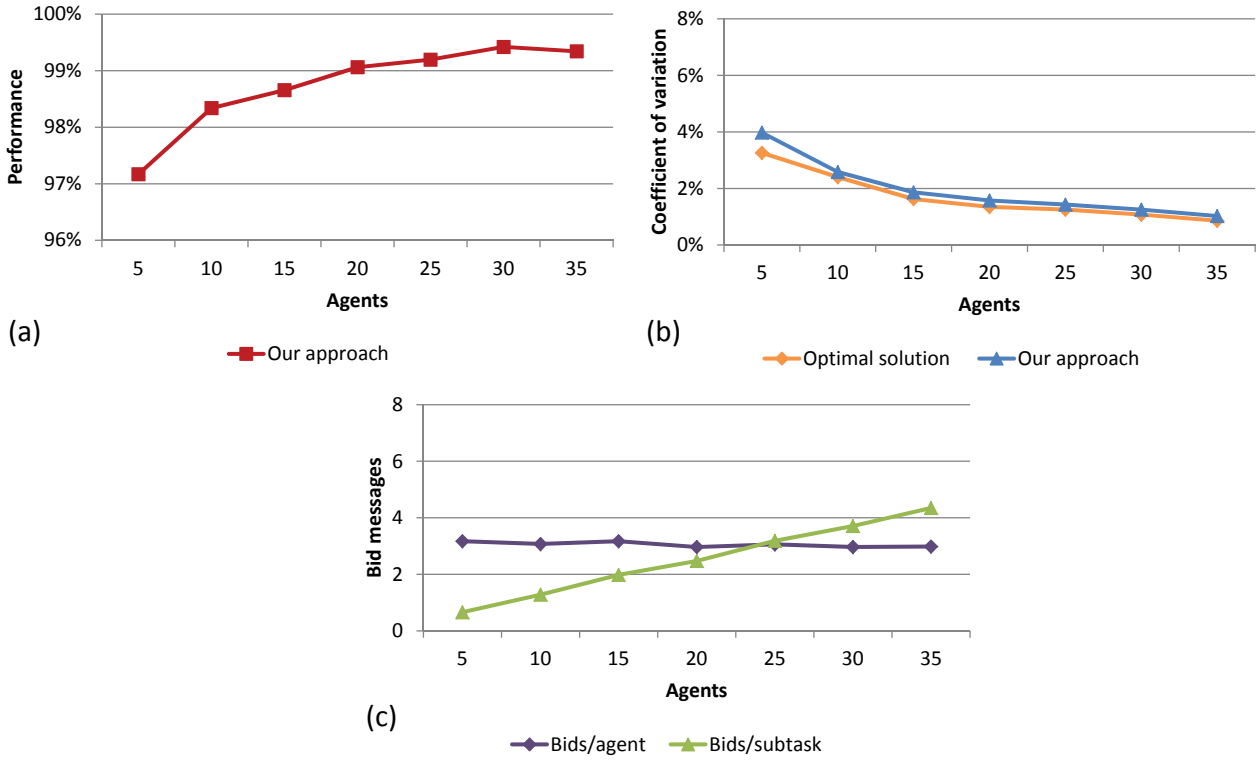


Figure 5.1 – Performance results varying the number of agents.

Besides, the coefficient of variation indicates that the consistency of the results obtained by both solutions is good and basically the same, with the results of the optimal solution being a little more stable when compared to the results of the proposed solution 5.1.b). However, this difference becomes smaller for larger agent teams. Regarding the number of bid messages, Figure 5.1.c shows that the average number of bid messages placed by an agent remains stable for the larger number of agents while the average

number of bid messages by subtask increases since more agents are bidding for the same subtasks. Note that a bid message may contain bids for more than one subtask.

Then we ran simulations by randomly assigning from 1 to 4 capabilities to each agent. Thus, some agents may not be able to play some roles, limiting the tasks they are able to carry out. Figure 5.2 shows the results of these simulations. Although the results when we consider agents capable of playing any role are somewhat better, Figure 5.2.a shows that the performance of the proposed solution is also close to the optimal solution and also better for the higher number of agents. The coefficient of variation between both solutions is basically the same 5.2.b), again with the difference becoming smaller for larger agent teams. Regarding the number of bid messages, Figure 5.2.c shows that the average number of bid messages placed by an agent decreases slightly for a larger number of agents and the average number of bid messages by subtask also increases slightly but not so much as when agents are able to play any role. These results are somewhat different from those in Figure 5.1.c because although there are more agents bidding for the same subtasks, here the agents were not able to bid on all subtasks.

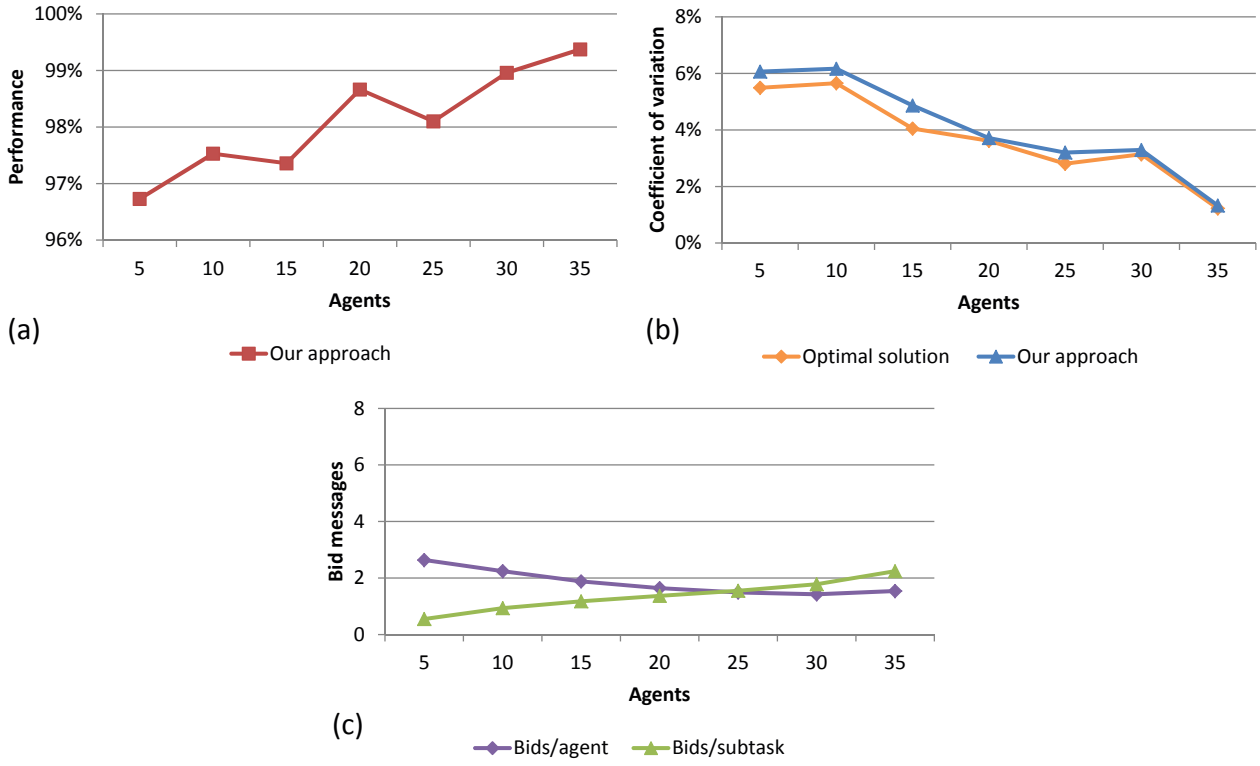


Figure 5.2 – Performance results varying the number of agents and agents capabilities.

### 5.1.4 Varying the Number of Subtasks (Setting 2)

The simulations varying the number of subtasks were performed using the values available in Setting 2 of Table 5.1. The simulations were run with 15, 30, 45, and 60

subtasks to be allocated to 10 agents, each one with a task limit of 7. In these simulations, we uniformly distributed subtasks of CN, CM, and DS task types. The utility values for each subtask were randomly selected from a range of 1 to 15.

First simulations considered agents with capabilities to play any role. Figure 5.3.a shows that although the performance has decreased somewhat with more subtasks, it is still close to the optimal solution. Even though the difference between the coefficients of variation of both solutions increases with more subtasks, it is still relatively small (Figure 5.3.b). The average number of bid messages that each agent provides increases with more subtasks while the average number of bid messages per subtask decreases (Figure 5.3.c). The results in Figure 5.3.c are because the agents have more subtasks to bid. For example, if there are fewer subtasks and an agent is outbid by another agent, it may not be able to bid another subtask because of bids that other agents have already provided to the subtasks, thereby reducing the number of bids each agent provides. When more subtasks are available, and an agent is outbid by another agent, it has more subtasks to be able to bid and then the average number of bid messages that each agent provides increases. Also, with more subtasks, the agents may bid on different subtasks then reducing the average number of bid messages per subtask.

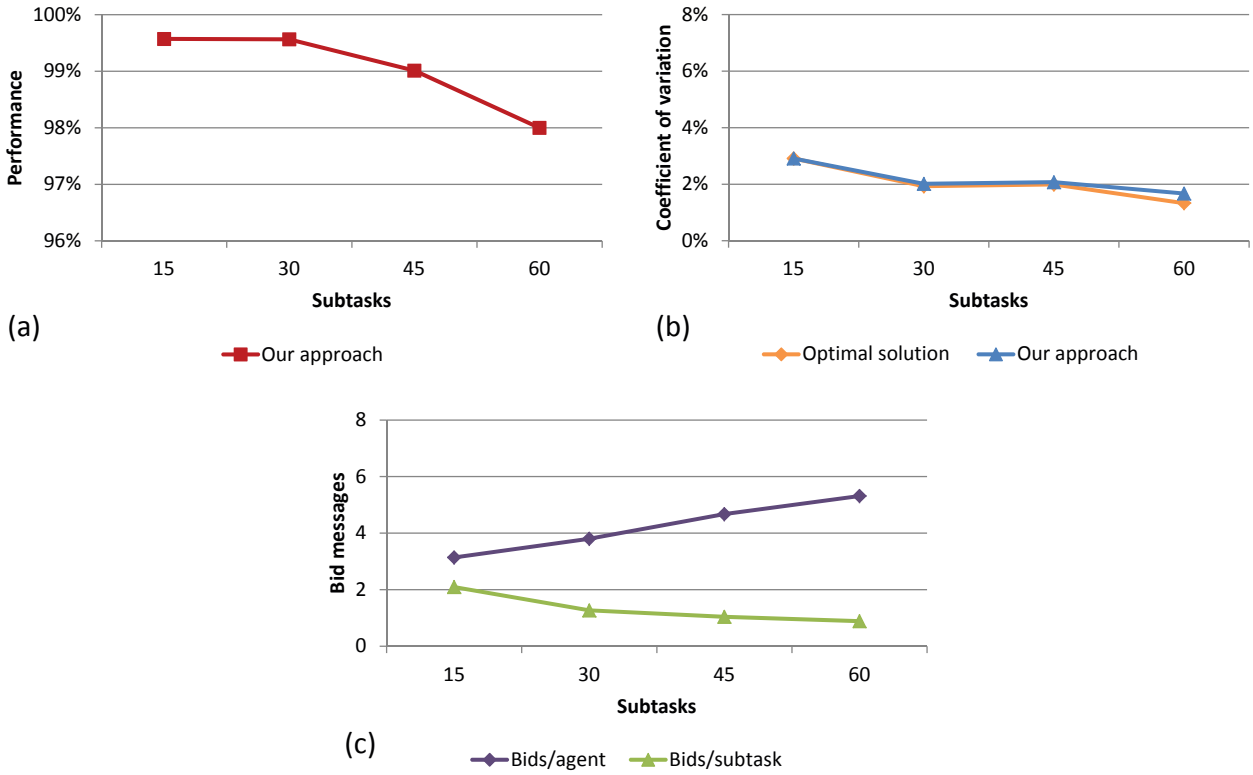


Figure 5.3 – Performance results varying the number of subtasks.

Then, for the next simulations, we randomly assign from 1 to 4 capabilities to each agent. Figure 5.4 shows the results with similar performance to those achieved when we considered agents with capabilities to play any role. Figure 5.4.a shows the results close to the optimal solution, although it has decreased with more subtasks.

The average number of bid messages that each agent provides here also increases with more subtasks while the average number of bid messages per subtask decreases (Figure 5.3.c). However, the average number of bid messages per subtask is lower here because, in addition to having more subtasks to provide bids, the agents were not able to bid on all subtasks due to their capabilities.

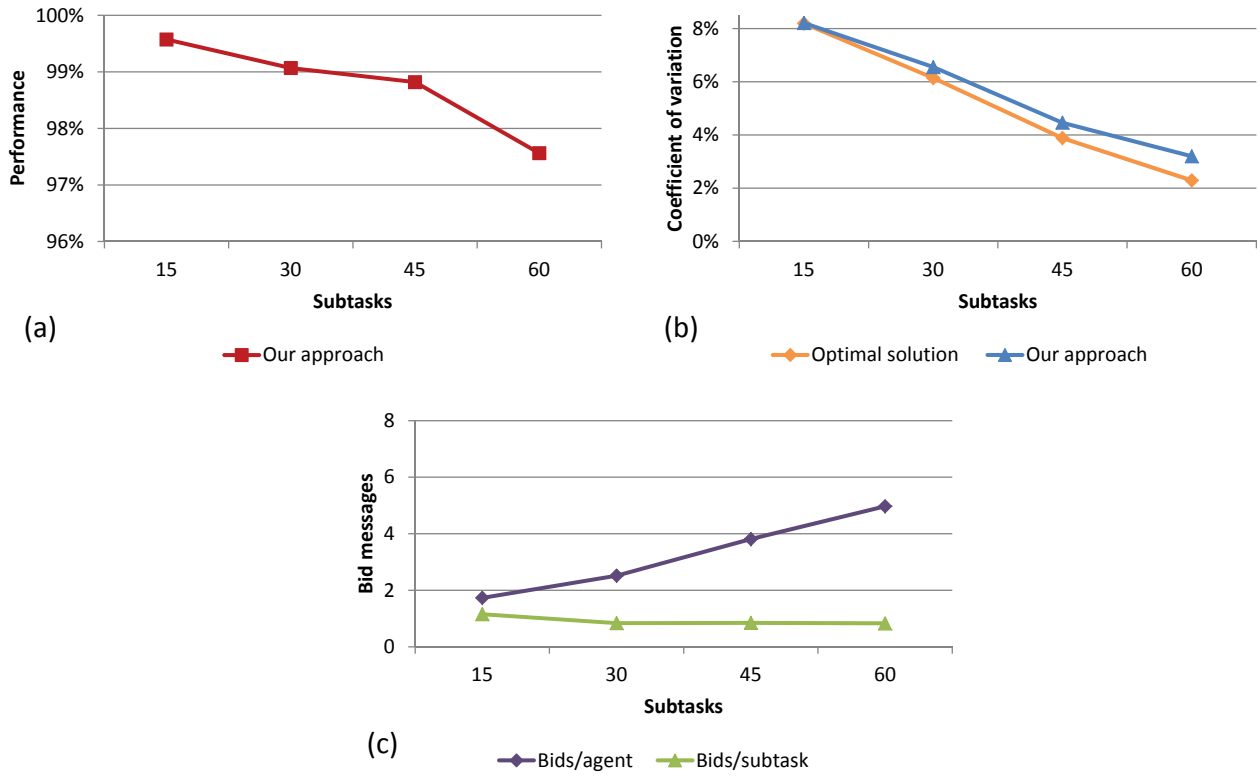


Figure 5.4 – Performance results varying the number of subtasks and agents capabilities.

### 5.1.5 Simulations for each Type of Task Individually

The above results show that the performance of the proposed mechanism decreases somewhat with the increase in the number of subtasks. However, since CN, CM, and DS types of tasks were uniformly distributed in the simulations, it is not clear the contribution of each type of task in the results. For this reason, we also ran simulations for each type of task individually.

For each type of task we ran simulations with 12, 24, 36, and 48 subtasks. The number of agents was kept 10 with 6 as the limit on the number of tasks to be allocated to each agent. For each agent we randomly selected the utility values from a range of 1 to 15 for each subtask. The results were averaged over one hundred simulations for each different number of subtasks.

Figure 5.5.a shows that both type of tasks, when increased, have an impact on the performance of our mechanism. However, we can see that although CM and

DS tasks had decreased somewhat, the CN type had more impact when the number of subtasks was increased. The DS type had more regular results for the different amounts of subtasks, having better performance for the greater number of tasks compared with the other types.

Regarding the number of bids required to complete the allocation, Figures 5.5.b and 5.5.c show the average number of bid messages placed for each subtask and the average number of bid messages placed by each individual agent. As we can see in Figure 5.5, the CN type of task required the highest average number of bid messages to complete the allocation for the different numbers of subtasks, while the DS type required fewer bid messages than the others.

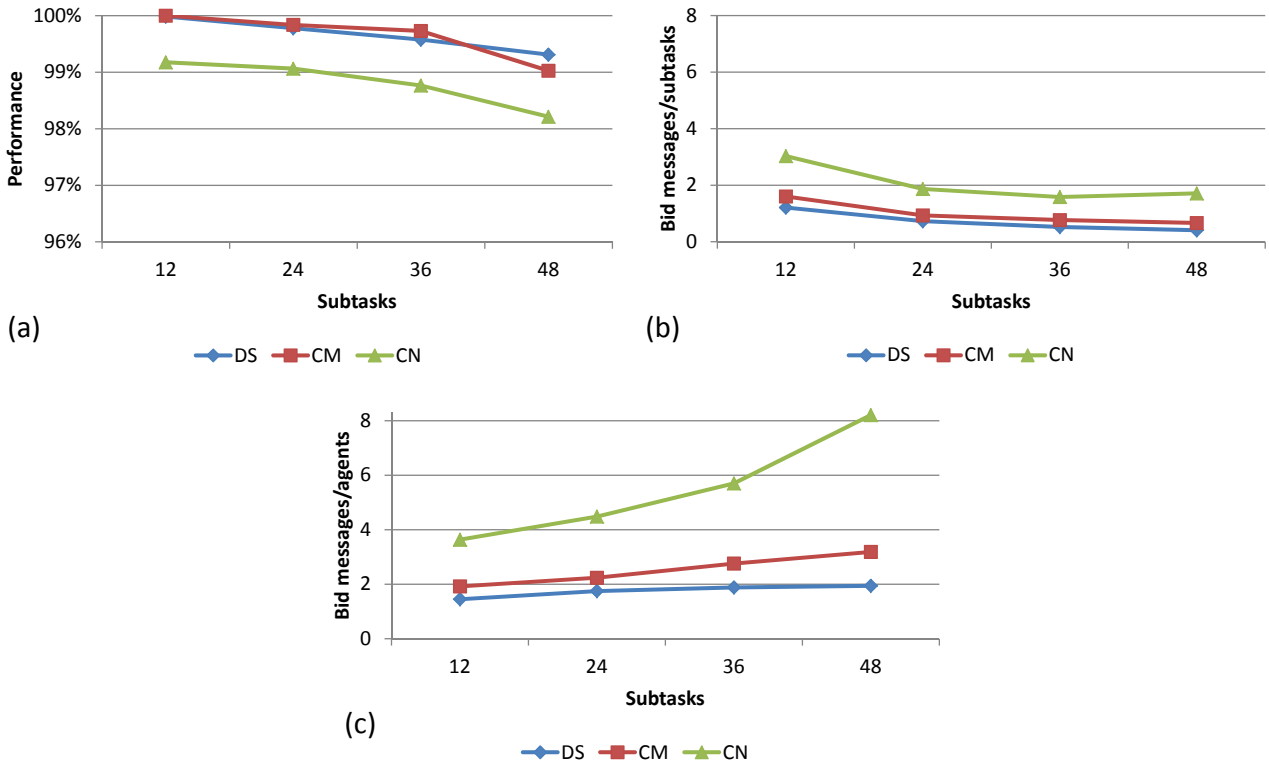


Figure 5.5 – Performance results varying the number of subtasks for each type of task.

### 5.1.6 Varying the Task Limit (Setting 3)

This section shows the performance results when we varied the limit of subtasks the agents can take using the values of Setting 3 in Table 5.1. In the simulations, the agents were set up with limits from 6 to 24. The number of agents and subtasks were kept 5 and 24, respectively, in all the simulations. The utility values were randomly selected from a range of 1 to 6.

Figure 5.6.a shows that the performance of our approach increased when agents are able to carry out more subtasks (higher agent limits). For the variation in the number

of subtasks that the agents can take, the coefficients of variation of our proposed solution and the optimal solution are very close to each other. Regarding the bids, the average number of bid messages remains stable, that is, the limit of subtasks an agent can take does not impact the number of exchanged bid messages.

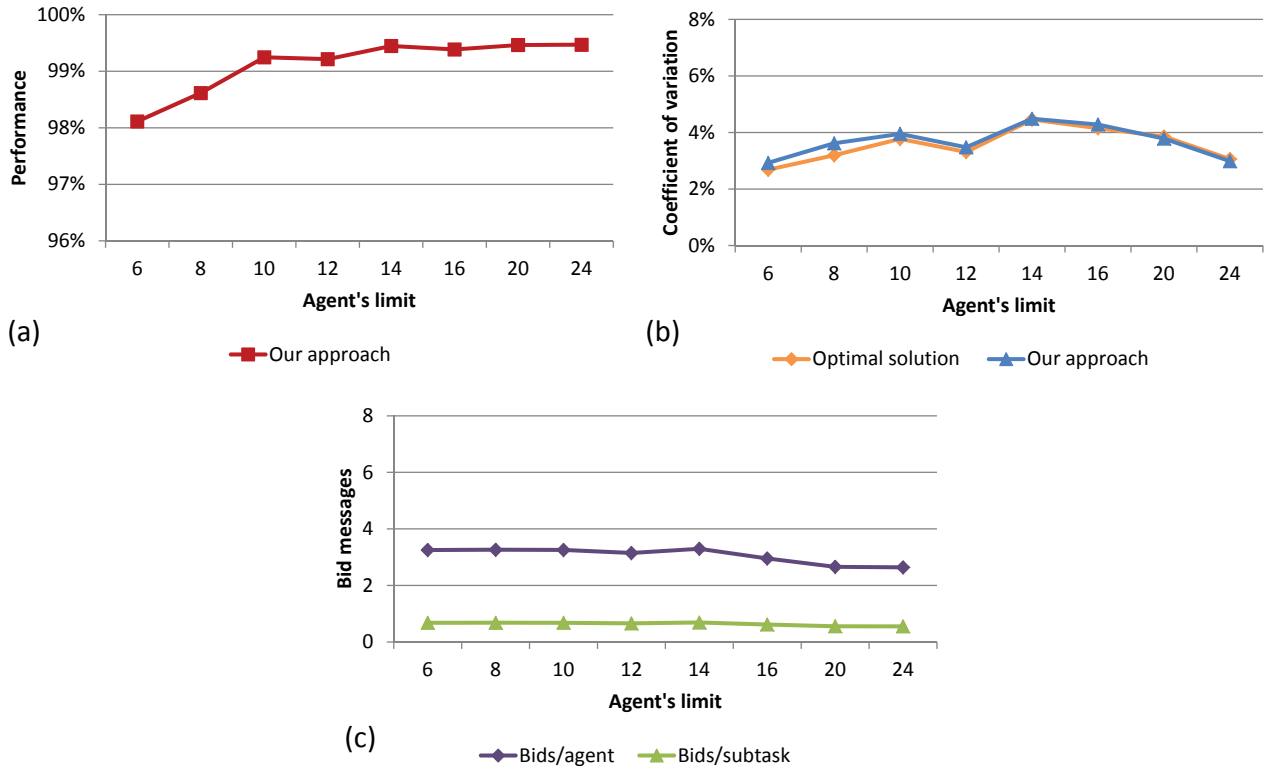


Figure 5.6 – Performance results varying the limit of subtasks the agents can take.

### 5.1.7 Varying the Utility Range (Setting 4)

In order to evaluate the impact of different ranges of utilities, we ran simulations with the utilities varying from 1 up to 6, 12, 24, and 48. The utility values for each subtask were randomly selected from each of those ranges. The number of agents and subtasks were kept 10 and 42 respectively.

Figure 5.7.a shows that the performance of our approach is better with broader utility ranges, although it is very close to the optimal solution for all available ranges.

For the variation of the utility ranges, the coefficients of variation of the proposed solution and the optimal solution are very close to each other. Regarding the bids, the average number of bid messages remains almost stable, that is, the utility ranges have a small impact on the number of exchanged bid messages.

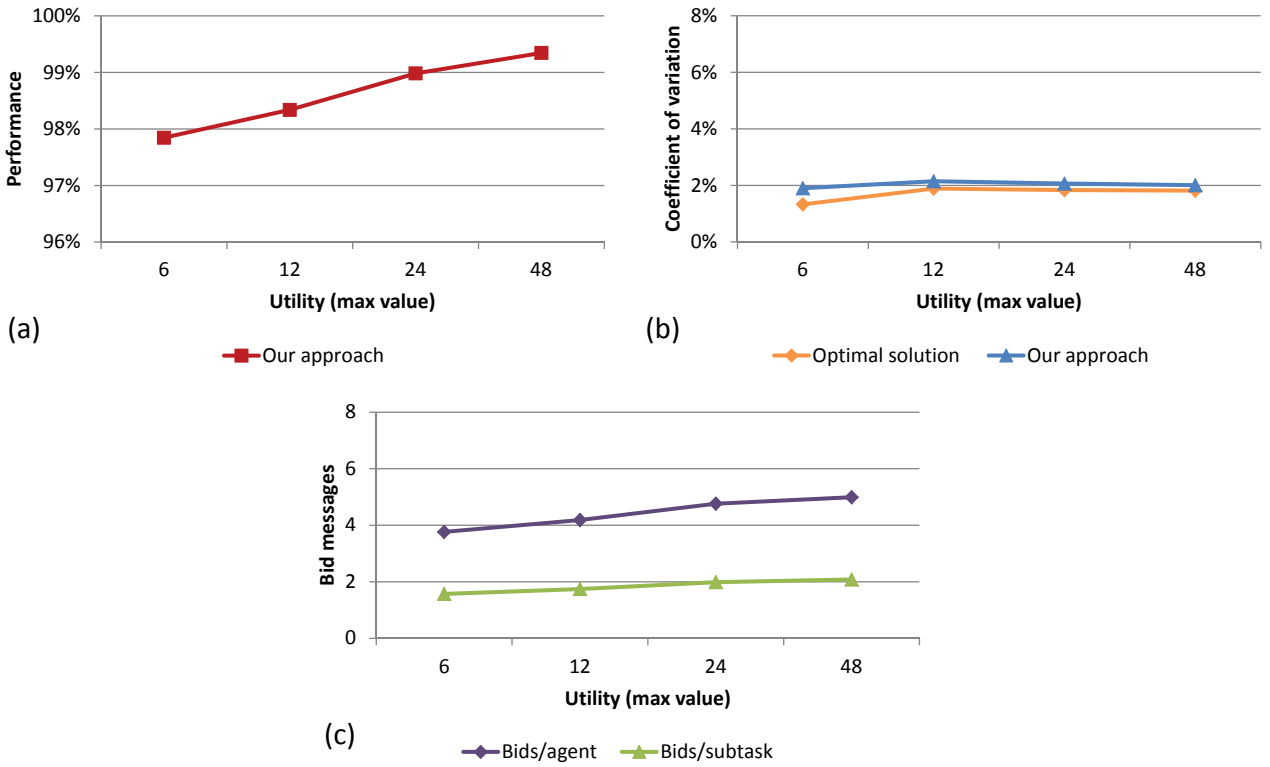


Figure 5.7 – Performance results varying the utility range.

### 5.1.8 Coping with Partially Allocated Tasks (Setting 5)

Previous simulations were performed considering that the total capacity of the agents is greater than or equal to the total number of subtasks that need to be allocated.

The next simulations we report were run with the number of subtasks greater than the total capacity of the agents. Thus, at the end of the allocation, we may have tasks partially allocated and others completely unallocated, which the agents will try to allocate again using the mechanism we proposed for this. For these simulations, the number of agents was 3 and the number of subtasks to be allocated was 21, 28, and 35. We used 5 as the limit on the number of tasks each agent can take which means that the agents are able to take up to 15 tasks during the allocation process.

Table 5.2 shows reasonable performance results on reallocating partially allocated tasks, where PA is the number of partially allocated tasks, NA is the number of completely unallocated tasks, and EA is the number of entirely allocated tasks (i.e., all subtasks were allocated). The average results were calculated from twenty iterations for each variation.

Table 5.2 – Simulations with the number of tasks greater than the total capacity of agents.

| Tasks | Subtasks | Phase | PA | NA | EA | Phase | PA | NA | EA |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 21 | 1 | 3 | 1 | 5 | 2 | 0 | 3 | 6 |
| 12 | 28 | | 5 | 4 | 3 | | 0 | 6 | 6 |
| 15 | 35 | | 6 | 5 | 4 | | 0 | 10 | 5 |

## 5.2 Comparison with SSIA and ICBAA

In this section, we compare the performance of our mechanism with ICBAA [CBH09] and SSIA [KKT10]. The simulations were run by varying the number of subtasks to be allocated and also varying the number of agents. The results for each variation of simulation parameters were averaged over fifty repetitions each.

### 5.2.1 Evaluation Measures

In this section, we describe the evaluation metrics used to analyse the performance of our mechanism with the ICBAA and the SSIA solutions.

- **Performance:** as mentioned before, here performance is the overall utility obtained by all the agents to take on all the subtasks that they can;
- **Coefficient of variation:** the coefficient of variation was used as a measure of dispersion;
- **Number of bid messages:** the average number of bid messages sent during the allocation process is also measured to assess the impact of the different solutions on the network traffic;
- **Computing time:** refers to the time taken to complete the allocation process.

We also statistically analysed the results of the experiments through paired t-tests. The t-Student test (or t-test only) is a hypothesis test that allows us to verify whether the data of a sample produces evidence that supports a hypothesis or not [BM03, Mey70].

The t-test for paired samples tests the difference between the means of two dependent populations, that is, the paired t-test is useful for analysing the same set of data submitted to two different conditions. For example, one can compare the results of two algorithms when applied to the same data set. Note that our simulations for all approaches were performed for the same problem instances.

### 5.2.2 Varying the Number of Subtasks

The simulations were run varying the number of subtasks from 17 up to 68 subtasks to be allocated to 10 agents. Table 5.3 shows the results of the simulations for the three algorithms, where the Utility represents the average utility obtained by the solutions during the simulations. Our algorithm is the one with higher utility values for all simulations when compared with the others. For all algorithms, the coefficients of variation are very low, indicating that the dispersion of the results is very small.

Table 5.4 shows the average number of bid messages that agents place during the execution and the time taken to complete the allocation process. Our algorithm is the one requiring fewer bid messages in all configurations.

In order to statistically analyse the results, we performed paired t-tests to determine that a significant difference does exist between the results from our solution and the results from ICBAA and SSIA algorithms. We statistically analysed the utility values obtained and also the number of bid messages by setting the significance level $\alpha = 0.05$. The t-tests showed that the differences were both statistically significant with p-values lesser than 0.05.

Table 5.3 – Performance results varying the number of subtasks

| | **ICBAA** | | **SSIA** | | **Our approach** | |
|---|---|---|---|---|---|---|
| Subtasks | Utility | Coef. Var | Utility | Coef. Var | Utility | Coef. Var |
| 17 | 301 | 3,76% | 306 | 3,38% | 311 | 2,94% |
| 34 | 608 | 2,00% | 618 | 2,08% | 625 | 1,85% |
| 51 | 920 | 1,83% | 931 | 1,49% | 939 | 1,54% |
| 68 | 1232 | 1,31% | 1238 | 1,27% | 1249 | 1,17% |

Table 5.4 – Average number of bids and time to complete the allocation

| | **ICBAA** | | **SSIA** | | **Our approach** | |
|---|---|---|---|---|---|---|
| Subtasks | Bid messages | Time(sec) | Bid messages | Time(sec) | Bid messages | Time(sec) |
| 17 | 208 | 10 | 35 | 4 | 26 | 3 |
| 34 | 401 | 25 | 55 | 5 | 37 | 4 |
| 51 | 501 | 56 | 72 | 9 | 43 | 5 |
| 68 | 689 | 122 | 91 | 10 | 50 | 7 |

### 5.2.3 Varying the Number of Agents

For these simulations, we varied the number of agents from 5 to 25 agents trying to allocate 68 subtasks. Since ICBAA requires a large number of bid messages

to complete the allocation (see Table 5.4), which could not be acceptable for real-world scenarios, here we decide to focus only in the comparison with SSIA.

Table 5.5 shows the results from the simulations for our mechanism and SSIA algorithm. Our algorithm performs better (higher utility values) than SSIA algorithm for the different number of agents. The coefficients of variation for both algorithms are basically the same. Table 5.6 shows the average number of bid messages that agents place during the execution and the time taken to complete the allocation process. Our algorithm requires fewer bid messages in all configurations.

We also statistically analyse the results our solution and the results from SSIA (utility values and number of bid messages) by performing paired t-tests setting the significance level $\alpha = 0.05$. The paired t-test showed that the differences were statistically significant with p-values less than 0.05.

Table 5.5 – Performance results varying the number of agents

| | **SSIA** | | **Our approach** | |
|---|---|---|---|---|
| Agents | Utilty | Coef. Var | Utility | Coef. Var |
| 5 | 1117 | 2,45% | 1129 | 2,24% |
| 10 | 1218 | 1,49% | 1230 | 1,41% |
| 25 | 1300 | 0,78% | 1307 | 0,65% |

Table 5.6 – Average number of bids and time to complete allocation

| | **SSIA** | | **Our approach** | |
|---|---|---|---|---|
| Agents | Bid messages | Time(sec) | Bid messages | Time(sec) |
| 5 | 73 | 9 | 36 | 6 |
| 10 | 90 | 10 | 53 | 8 |
| 25 | 126 | 14 | 100 | 17 |

## 5.3 Comparison with TAA and RBTA

In this section, we compare the performance of our mechanism with the task assignment approaches used in two frameworks: a task assignment algorithm (we call it TAA) used in [GKZ17] and a role-based task assignment (we call it RBTA) introduced in [GA15]. Both approaches deal only with part of the task structures we deal with, more specifically atomic tasks. Our idea is to compare the performance of approaches developed for specific types of tasks with the performance of our mechanism, which is broader, given that we are not aware of other algorithms that deal with the same types of tasks as in our approach. The results for each variation of simulation parameters below were averaged over fifty repetitions each.

### 5.3.1 Evaluation Measures

In this simulations analyse the performance of our mechanism using the same evaluation metrics used in the previous section: performance, coefficient of variation, number of bid messages and computing time. We also statistically analysed the results of the experiments through paired t-tests.

### 5.3.2 Comparison with TAA

In this approach the number of agents and the number of tasks must be the same, each agent can allocate only one task, and the agents are homogeneous. Thus, these simulations were run with 5, 10, and 15 agents and tasks. Table 5.7 shows the results of the simulations for the algorithms, where Utility represents the average utility of the simulations. Our algorithm has higher utility values in all simulations when compared with TAA. The coefficients of variation indicate that the dispersion of the results is small and similar between both solutions.

Table 5.7 – Performance results varying the number of agents/tasks

| | **TAA** | | **Our approach** | |
|---|---|---|---|---|
| Agents/Tasks | Utilty | Coef. Var | Utility | Coef. Var |
| 5 | 23 | 10,59% | 25 | 8,83% |
| 10 | 53 | 5,80% | 56 | 3,57% |
| 15 | 83 | 3,61% | 87 | 2,10% |

Table 5.8 shows the average number of bid messages that agents place during the execution and the time taken to complete the allocation process. Our algorithm requires fewer bid messages in all configurations and perform faster than TAA.

We performed paired t-tests to analyse the results statistically. We analysed the utility values obtained and also the number of bid messages by setting the significance level $\alpha = 0.05$. The t-tests showed that the differences were both statistically significant with p-values less than 0.05.

Table 5.8 – Average number of bids and time to complete allocation

| | **TAA** | | **Our approach** | |
|---|---|---|---|---|
| Agents/Tasks | Bid messages | Time(sec) | Bid messages | Time(sec) |
| 5 | 15 | 6 | 8 | 2 |
| 10 | 55 | 7 | 19 | 2 |
| 15 | 120 | 8 | 30 | 3 |

### 5.3.3 Comparison with RBTA

In the RBTA approach, each agent can allocate more than one task (restricted to a certain limit). Also, the agents may have different capabilities that limit the tasks they can allocate. Thus, the simulations were run by varying the number of tasks to be allocated and also varying the number of agents.

First, the simulations were run with 40, 50, and 60 tasks to be allocated to 10 agents. Table 5.9 shows the results of the simulations for the algorithms. Our algorithm obtained higher utility values for all simulations, especially when the number of tasks was increased. The coefficients of variation indicate that the dispersion of the results is small and similar between both solutions. Table 5.10 shows the average number of bid messages that agents place during the execution and the time taken to complete the allocation process. The RBTA approach required a much larger number of bid messages in all configurations.

Table 5.9 – Performance results varying the number of tasks

| | **RBTA** | | **Our approach** | |
|---|---|---|---|---|
| Tasks | Utilty | Coef. Var | Utility | Coef. Var |
| 40 | 218 | 6,50% | 219 | 5,68% |
| 50 | 280 | 3,29% | 282 | 2,47% |
| 60 | 333 | 3,78% | 340 | 3,25% |

Table 5.10 – Average number of bids and time to complete allocation

| | **RBTA** | | **Our approach** | |
|---|---|---|---|---|
| Tasks | Bid messages | Time(sec) | Bid messages | Time(sec) |
| 40 | 284 | 11 | 60 | 5 |
| 50 | 409 | 12 | 94 | 6 |
| 60 | 531 | 14 | 158 | 12 |

Then we run the simulations by varying the number of agents (10, 12, 15, and 18 agents) to allocate 60 tasks. Table 5.11 shows the results from the simulations for our mechanism and the RBTA algorithm. Our algorithm performs better (higher utility values) than RBTA for the different numbers of agents, except for the simulation with 18 agents, which had the same results. The coefficients of variation for both algorithms are basically the same. Table 5.12 shows the average number of bid messages that agents place during the execution and the time taken to complete the allocation process. Again, the RBTA approach required a much larger number of bid messages.

For both variations (agents and tasks) we statistically analyse the results of our solution and the results from RBTA (utility values and number of bid messages) by

Table 5.11 – Performance results varying the number of agents

| | **RBTA** | | **Our approach** | |
|---|---|---|---|---|
| Agents | Utilty | Coef. Var | Utility | Coef. Var |
| 10 | 333 | 3,78% | 340 | 3,25% |
| 12 | 338 | 3,80% | 340 | 2,95% |
| 15 | 334 | 4,68% | 337 | 3,49% |
| 18 | 338 | 4,51% | 338 | 3,14% |

Table 5.12 – Average number of bids and time to complete allocation

| | **RBTA** | | **Our approach** | |
|---|---|---|---|---|
| Agents | Bid messages | Time(sec) | Bid messages | Time(sec) |
| 10 | 531 | 14 | 158 | 10 |
| 12 | 537 | 13 | 126 | 10 |
| 15 | 542 | 14 | 110 | 9 |
| 18 | 608 | 14 | 113 | 11 |

performing paired t-tests setting the significance level $\alpha = 0.05$. The paired t-test showed that the differences were statistically significant with p-values less than 0.05, except for the simulation with 18 agents (see Table 5.11) which had no statistical difference between the results.

### 5.3.4 Discussion

We have empirically evaluated our approach by comparing its results with the optimal solution and the results from other four approaches. Our simulation results show that our approach provides near-optimal solutions in all simulation variations (settings 1 to 4 in Table 5.1). In fact, the results are close to the optimal solution when the task types were simulated individually (see Figure 5.5), although even together the results were clearly close to the optimal solution. The coefficient of variation from all settings indicates that the consistency of the results obtained is reasonably close to the centralised optimal solution.

Regarding the comparison with SSIA and ICBAA algorithms, the results show that our approach performs better than these two algorithms. Also, there is a significant difference in the number of bid messages that were required, especially when compared to ICBAA. The number of bid messages sent by ICBAA shows that it tends not to be suitable for real-world scenarios. Although SSIA requires fewer bid messages than ICBAA, it still sent on average 40% more bid messages than our approach, going up to over 70% in some cases, and that can also be considered an important difference for applications to real-world scenarios. The results show that our approach also performs better than the TAA and RBTA algorithms. Again, there is a significant difference in the number of bid messages that were required by TAA and RBTA when compared with our approach.

Although our approach showed good performance in the experiments we carried out, we evaluate only the performance of the task allocation itself, but we do not measure the impact of the task allocation results in the execution of the tasks. Thus, the next section presents such evaluation in a complex scenario as the flooding disaster scenario.

## 5.4 Evaluation in the Flooding Disaster scenario

In the last sections, we presented the evaluation of our task allocation mechanism regarding global team utility, the number of bid messages and the time taken to complete the allocation. In this section, we evaluate the impact of the task allocation results in the execution of the tasks in a complex scenario like the flooding disaster scenario. First, we describe some features of the simulator used for this evaluation, and then we present the results of the evaluation.

### 5.4.1 Simulator

We extended the Simulation platform for the Multi-Agent Programming Contest (MASSim) [1] to work on a flooding disaster scenario. MASSim is a simulation server used in the Multi-Agent Programming Contest. The communication between agents and the MASSim server is performed through the EISMASSim, which is based on the [Environment Interface Standard][2](EIS).

In MASSim the agents connect to the server, receive percepts and send their actions, which are executed by the simulator (see Figure 5.8). The simulation is divided into discrete steps, and each agent can execute at most one action per step. If an agent does not submit an action before a configured timeout, the simulator considers the agent had no action to perform at that step.

Our flooding disaster scenario consists of a number of agents, which can move through flooded regions over the map of a realistic city, either by water or air. The purpose of the agents is to perform as many tasks as possible and in the best possible way. The agents perform tasks by executing actions in the environment through the simulator. The tasks are provided by an organisation (CDM) which is also an agent connected in the simulator.

In each simulation, the CDM will provide a set of tasks to perform. Tasks differ by their type and actions that need to be performed in the environment. Agents are initially positioned near to the CDM station, and tasks will be distributed on the map. During the execution of the tasks, the agents can discover other tasks to be executed.

[1] https://multiagentcontest.org/2016/
[2] https://github.com/eishub/

For each simulation, there are many configurations that may be defined. Most of the configurations in the MASSim simulator are performed through json files. Figure 5.8 presents the main configurations that are loaded by the simulator when the simulation starts. Next, we describe the actions that agents may perform during the simulations as well as the initial configurations.

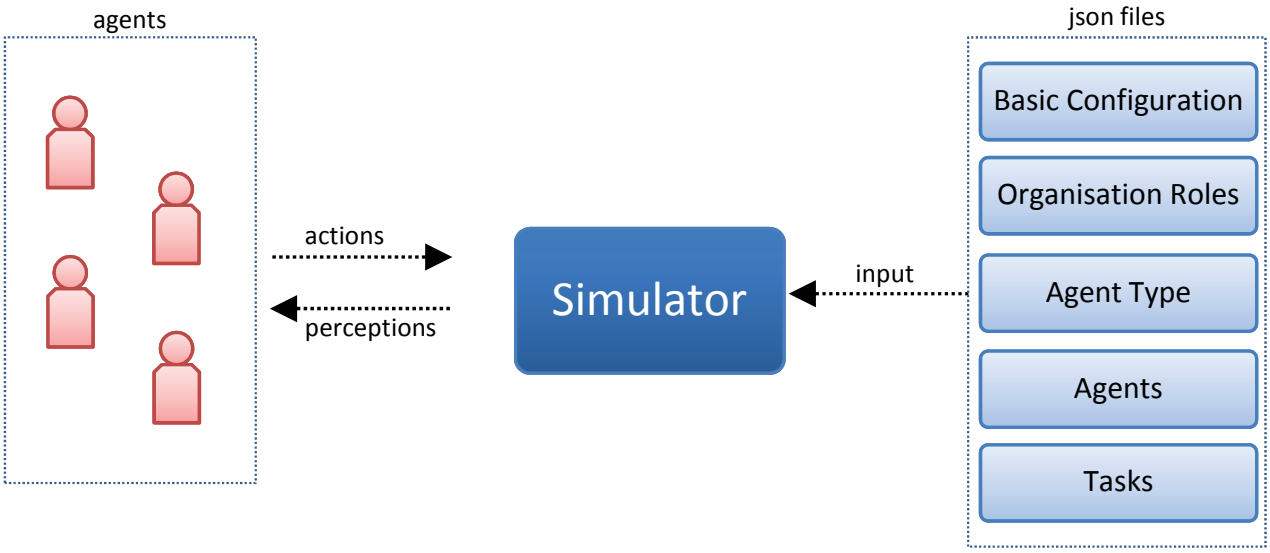


Figure 5.8 – Overview of the simulator interactions.

**Basic Configuration**

There are many parameters available in the basic configuration of the MASSim. Listing 5.1 provides a snippet of the json file with the basic configuration. We describe here some of the parameters which are necessary for our simulations:

```
...
{
   "steps" : 500,
   "map" : "paris",
   "minLon" : 2.26,
   "maxLon" : 2.41,
   "minLat" : 48.82,
   "maxLat" : 48.90,
   "gotoCost" : 10,
}
"agentTimeout" : 4000,
"entities" : [{"drone" : 6}, {"car" : 0}, {"boat" : 4}, {"cdm" : 1}]
...
```

Listing 5.1 – A snippet of the basic configuration file

Where:

- **steps:** defines the number of steps of the simulation;
- **map:** refers to the map of the city used during the simulation;
- **min/max Lon/Lat:** the area of the map where the simulation will happen;
- **gotoCost:** the amount of battery spent on each goto action performed by an agent;
- **agentTimeout:** the time the simulation wait for the agents to send their actions before processing received actions and moving to the next step of the simulation;
- **entities:** defines the number of agents per type (see entity Type);

### Organisation Roles

Here we describe how we define the roles available in the organisation in the json file loaded by the simulator. This configuration is part of our extension to the MASSim. Listing 5.2 provides a snippet of the json file with the definition of the organisation roles. Each role can be defined by a description (such as "collector" in the example) and the list of capabilities an agent must have to play that role.

```
{
...
   "collector" : {
   "capabilities" : ["sail","collector"]
   },
   "mapper" : {
   "capabilities" : ["fly","camera"]
   },
...
}
```

Listing 5.2 – A snippet of the organisation roles configuration file

### Agent Type

We describe here how we define the type of agents that will be available during the simulation. This configuration is similar to the one available in the MASSim but with the addition of the victimLoad parameter. Listing 5.3 provides a snippet of the json file with the definition of the agent types.

Where the first information is the description of the type, and the following parameters are:

- **speed:** the speed at which the agent moves in the environment;

```
{
...
"boat" : {
   "speed" : 3,
   "load" : 550,
   "victimLoad" : 2,
   "battery" : 800,
   "roads" : ["water"]
},
"drone" : {
   "speed" : 5,
   "load" : 100,
   "victimLoad" : 1,
   "battery" : 400,
   "roads" : ["air"]
}
...
}
```

Listing 5.3 – A snippet of the agent types configuration file

- **load:** how much volume the agent may carry;
- **victimLoad:** how much victims the agent may carry;
- **battery:** the agent's amount of battery;
- **roads:** which roads the agent can navigate.

**Agents**

Here we describe how the agents are defined for the simulations. This configuration was added as part of our extension to the MASSim. The current version of the extended simulator loads a json file with information about the agents and their capabilities. This json file can be manually created or generated by the simulator.

**Generation of agent data:** the generation of agent data is performed through a java code added to the simulator. The generation code requires that the organisation roles be defined a priori since they are used to generate the data. Also, the number of agents per type (defined in the basic configuration) is used as input. The code uses the capabilities required by the organisation roles to randomly attribute capabilities for the agents. Listing 5.4 provides a snippet of the json file generated with the agents' definitions.

The first information is the description of the agent, and the following parameters are:

- **capabilities:** the list of capabilities available for that agent;

```
{
...
"agentA1": {
   "capabilities": [
      "router",
      "sail",
      "collector"
   ],
   "entityName": "agentA1",
   "entityType": "boat"
},
"agentA2": {
      "capabilities": [
      "sail",
      "camera"
   ],
   "entityName": "agentA2",
   "entityType": "boat"
},
...
}
```

Listing 5.4 – A snippet of the agents configuration file

- **entityName:** the name of the entity (used to map agents to an entity in the simulator);
- **entityType:** indicates the type of the agent (see Agent type described above);

**Tasks**

The tasks that need to be performed during the simulation are also loaded by the extended simulator through a json file. This json file can be manually created or generated by the simulator. Here we describe how the tasks with their subtasks are defined for the simulations. This configuration was also added as part of our extension to the MASSim.

**Generation of task data:** the generation of task data is performed through a java code added to the simulator. Each of the flooding tasks has a (configurable) probability of being generated. Tasks are generated with a random number of subtasks, limited by a minimum and a maximum number of subtasks also configurable. It is also random the task type definition (DS, CM, CN). Besides, it is possible to set a maximum distance that the subtasks of the same task can be in the simulation. Subtasks also have a probability of having an associated victim. Having an associated victim means that when the agent performs the action related to the subtask, it will receive the perception

of a victim on the spot. The generation of task data is divided into two parts: the initial tasks and the step tasks. Initial tasks is the set of tasks that will be announced at the beginning of the simulation. The step tasks are the ones that will be announced during the steps of the simulation.

Listing 5.5 provides a snippet of the json file generated with the initial tasks. In the example, there is a task "sampleWaterJ1T1" with two subtasks: "sampleWaterJ1T1S1" and "sampleWaterJ1T1S2".

```
{
...
"sampleWaterJ1T1S1": {
   "jobid": "job1",
   "tasktype": "ds",
   "subtaskaction": "sampleWater",
   "roleid": "collector",
   "subtaskid": "sampleWaterJ1T1S1",
   "location": {
      "lon": 2.37115,
      "lat": 48.88306
   },
   "victim": 0,
   "taskid": "sampleWaterJ1T1"
},
"sampleWaterJ1T1S2": {
   "jobid": "job1",
   "tasktype": "ds",
   "subtaskaction": "sampleWater",
   "roleid": "collector",
   "subtaskid": "sampleWaterJ1T1S2",
   "location": {
      "lon": 2.3719,
      "lat": 48.88535
   },
   "victim": 0,
   "taskid": "sampleWaterJ1T1"
},
...
}
```

Listing 5.5 – A snippet of the initial tasks file

The first information is the description of the subtask (such as "sampleWaterJ1T1S1"), and the following parameters are:

- **jobid:** is an identifier for the set of tasks which were announced together;
- **tasktype:** is the type of the task to which the subtask belongs. It can be one of the task types described in section 3.1;
- **subtaskaction:** is the action that needs to be performed by the agent in the environment;

- **roleid:** is the organisation role that an agent must be able to play to perform that subtask;
- **subtaskid:** is an identifier for the subtask;
- **location:** is the location in the map where the subtask is to be executed. It is composed by the latitude and longitude of the subtask;
- **victim:** indicates if the subtask has a victim associated with it;
- **taskid:** is an identifier for the task to which the subtask belongs.

Listing 5.6 provides a snippet of the json file generated with the step tasks. The difference for the initial tasks is that in this case the tasks are grouped by the step which they need to be announced. In the example, there are two steps of the simulation were tasks will be announced: step 23 and step 37. In step 37, for instance, there is a task "mappingJ8T1" with two subtasks: "mappingJ8T1S1" and "mappingJ8T1S2".

**Actions**

During the simulation, at each step, an agent may perform only one action. The actions are always executed by the simulator (no fail occurs). Next, we describe the actions the agents may perform in our flooding scenario. The first nine actions were defined in our scenario, and the last five actions were reused from the original MASSim.

- **take_picture:** used to take a picture from some location. When an agent executes this action it will receive a perception that is has a picture;
- **communicate_picture:** used to send a picture to the CDM. The agent needs to be in the CDM location.
- **sample_water:** used to collect water samples at some location. When an agent performs this action it will receive a perception that is has a sample of water;
- **drop_sample:** used to deliver a sample of water at the CDM. The agent needs to be in the CDM location;
- **pickup_box:** action used to get a box in the CDM. The agent needs to be in the CDM location;
- **drop_box:** action used to deliver a box at the current agent’s location;
- **rescue_victim:** used to rescue a victim in the current agent’s location;
- **delivery_victim:** used to delivery the victim at CDM;

```
...
{
"23": {
   "mappingJ2T1S1": {
      "jobid": "job2",
      "tasktype": "ds",
      "subtaskaction": "mapping",
      "roleid": "mapper",
      "subtaskid": "mappingJ2T1S1",
      "location": {"lon": 2.34049, "lat": 48.87968},
      "taskid": "mappingJ2T1"
   }
},
"37": {
   "mappingJ8T1S1": {
      "jobid": "job8",
      "tasktype": "tcl",
      "subtaskaction": "mapping",
      "roleid": "mapper",
      "subtaskid": "mappingJ8T1S1",
      "location": {"lon": 2.3858,"lat": 48.86534},
      "taskid": "mappingJ8T1"
   },
   "mappingJ8T1S2": {
      "jobid": "job8",
      "tasktype": "tcl",
      "subtaskaction": "mapping",
      "roleid": "mapper",
      "subtaskid": "mappingJ8T1S2",
      "location": {"lon": 2.38186,"lat": 48.86843},
      "taskid": "mappingJ8T1"
   }
}
...
```

Listing 5.6 – A snippet of the step tasks file

- **route_net:** used to extend the range of the network communication through the propagation of network signal;
- **goto:** used to move an agent to a destination. Each time this action is called, it consumes the amount of battery as defined in the basic configuration. The parameters are the latitude and the longitude of the agent's desired destination. If no parameters are used, the agent wants to follow the current route (if it exists).
- **charge:** used to charge the agent's battery. The agent needs to be at the CDM location for charging.
- **continue & skip:** follows the current agent's route or does nothing if the agent has no route;

- **abort:** clears the current agent's route (if it exists);
- **noAction:** this action is considered when an agent did not send an action in time.

**Percepts**

The simulator sends percepts with information about the current simulation. There are two type of percepts: Initial percepts and Step percepts.

**Initial percepts:** This percept contains the information the agents need to begin the simulation, and they will not change during the simulation. The initial percepts include information about the simulation, such as the name of the map used and its bounds, the number of simulation steps and others. These percepts contain also details about the type of agents, such as speed, load, victim load, and battery. Note that each agent will receive only the information about its type.

For our scenario, we added to the initial percepts a set of tasks which are loaded from the initial tasks json file. We also added the organisation roles which are loaded from a json file. Since the organisation (CDM) is responsible for announcing the tasks and roles, this initial percept will be available only for the organisation. Each agent will also receive in the initial percepts the information about its capabilities and the organisation location.

**Step percepts:** In the step percepts, the information is related to the simulation state before the simulator moves to the next step of the simulation. Each agent receives information about itself such as its current battery charge, used capacities, position and others. If the agent is in the same location as the CDM, CDM will be listed as a percept. The result for the last action executed by an agent is also available.

For our scenario, when an agent takes a picture in the same position as a victim, it will receive a percept about that victim. Also, as available in the steps task file, tasks are added to the step percepts. Again, since the organisation is responsible for announcing the tasks, this percepts will be available only for the organisation.

### 5.4.2 Using the Simulator

In this section, we provide a brief overview of how we use the simulator. First, we mention how we integrate our agents developed in JaCaMo platform with the simulator. Next, we describe the agents' behaviour for executing the allocated subtasks.

**Agent environment artifact**

To act and receive perceptions from the simulator we used the solution used in [CPK+18] and [CKB+18]. The solution uses a CArtAgO artefact called EISArtifact, which is responsible for register the agents, receive (filter) the perceptions into observable properties, and send actions to the simulator. Each agent has it own EISArtifact artifact. At every step of a simulation new perceptions are received, and to deal with them, the artefacts filter any useful perception into an observable property by adding/updating/removing them in the artefact's observable properties [CPK+18].

**Task Execution**

To evaluate the impact of the task allocation on the execution of subtasks, we encode the expected behaviour of the agents regarding the execution of subtasks. When an agent has new subtasks allocated, it starts the task execution phase, which is represented by Algorithm 16 at a rather high level. Algorithm 16 was developed in Jason, except where noted below. Note that new subtasks may be allocated by the agent even though the agent is performing other tasks. Thus, when new subtasks are allocated, the first step performed by Algorithm 16 is to stop the current execution of actions, which is defined on line 1. If the agent was already running Algorithm 16, it will percept that in the while loop at line 8 and then will stop that execution of actions. Next, based on the new subtasks, the current subtasks, and the current list of actions not yet performed, the algorithm defines the order (route) in which the subtasks must be performed (that is performed in by calling function defineSubtaskRoute at line 4). The function $defineSubtaskRoute$ is an algorithm for the salesman problem developed in Java. Next, the algorithm estimates the battery needed to perform the subtasks in the defined route. This is necessary to reserve battery for the allocated subtasks and avoid that the allocation process of allocating more subtasks than the agent would be able to carry out. Since new subtasks were allocated the current order in which the actions would be executed are not valid anymore. Thus, the algorithm deletes the actions that were not executed until now (line 6). Next, knowing the order in which the subtasks will be executed, the algorithm determines the actions that need to be performed and the order of execution of those actions (line 7 call the function responsible for that). In the sequence, the algorithm starts the execution of the actions (line 8 to 13).

**Algorithm 16**
`taskExecution`($newSubtasks$)

1: $stopActionExecution \leftarrow true$;
2: $currentSubtasks \leftarrow getCurrentSubtasks$;
3: $actionOrderList \leftarrow getCurrentActionOrderList$
4: $subtasksOrder \leftarrow defineSubtaskRoute(newSubtasks, currentSubtasks, actionOrderList)$;
5: $estimateBattery(subtasksOrder)$
6: $deletePendingActions$;
7: $actionOrderList \leftarrow defineActionsOrder(subtasksOrder)$;
8: **while** $stopActionExecution \neq true$ **and** $actionOrderList \neq \emptyset$ **do**
9: $action_i \leftarrow actionOrderList.getFirstAction$;
10: $executeAction(action_i)$;
11: $updateReservedBattery$;
12: $actionOrderList.removeFirstAction$;
13: **end while**

### 5.4.3 Evaluation Measures

In this section, we describe the evaluation metrics used to analyse the performance of our mechanism with the ICBAA and the SSIA solutions in the flooding scenario.

- **Performance:** as mentioned before, here performance is the overall utility obtained by all the agents to take on all the subtasks that they can;
- **Number of bid messages:** the average number of bid messages sent during the allocation process is also measured to assess the impact of the different solutions on the network traffic;
- **Battery amount:** refers to the total amount of battery the agents spent to perform the subtasks.

### 5.4.4 Varying the Number of Subtasks

The simulations were run varying the number of subtasks from 17 up to 68 subtasks to be allocated to 10 agents. For these simulations, we defined the 10 agents with different capabilities as specified in Listing 5.7. There are also five different type of agents (entityType): two type of boats and three type of drones. Both boats have battery level at 2400 and speed 3 and 4, respectively for boat and boat2. All drones have battery level at 900 and speed 5, 7 and 9, respectively for the drone, drone2 and drone3. There were also five organisation roles defined in the organisation: collector, mapper, rescuer, router and deliverer. Each of the agents was able to play one or more of the organisation roles.

```
{
   "agentA1": {
      "capabilities": ["router","sail","collector"],
      "entityName": "agentA1",
      "entityType": "boat2"
   },
   "agentA2": {
      "capabilities": ["router", "sail", "collector" ],
      "entityName": "agentA2",
      "entityType": "boat"
   },
   "agentA3": {
      "capabilities": [ "sail", "collector" ],
      "entityName": "agentA3",
      "entityType": "boat"
   },
   "agentA4": {
      "capabilities": [ "router", "sail" ],
      "entityName": "agentA4",
      "entityType": "boat"
   },
   "agentA5": {
      "capabilities": [ "fly", "load_box", "load_victim", "camera" ],
      "entityName": "agentA5",
      "entityType": "drone"
   },
   "agentA6": {
      "capabilities": [ "fly", "load_box", "load_victim", "camera" ],
      "entityName": "agentA6",
      "entityType": "drone"
   },
   "agentA7": {
      "capabilities": [ "fly", "load_box", "load_victim", "camera" ],
      "entityName": "agentA7",
      "entityType": "drone2"
   },
   "agentA8": {
      "capabilities": [ "fly", "load_victim", "camera" ],
      "entityName": "agentA8",
      "entityType": "drone3"
   },
   "agentA9": {
      "capabilities": [ "fly", "camera", "load_box" ],
      "entityName": "agentA9",
      "entityType": "drone3"
   },
   "agentA10": {
      "capabilities": [ "fly", "camera", "load_victim" ],
      "entityName": "agentA10",
      "entityType": "drone2"
   }
}
```

Listing 5.7 – A snippet of the agent types configuration file

Table 5.13 shows the average results of the simulations for both algorithms. In these simulations, our objective was to minimise the amount of battery spent by the agents during the mission, thus the utility values were related to the distances to each task. Our algorithm is the one with lower average utility, bid and battery values for the simulations in comparison with SSIA.

Figure 5.9 shows the number of bid messages that agents place during the execution to complete the allocation process. Simulations 1 to 10 are simulations with 17 subtasks, 11 to 20 with 34 subtasks, 21 to 30 with 51 subtasks, and 31 to 40 with 68 subtasks. Except by one simulation with 17 subtasks, our algorithm requires fewer bid messages in all configurations, especially when the number of subtasks increases.

To evaluate the impact of the task allocation on the execution of subtasks we also measured the total amount of battery the agents spent to perform the subtasks. Figure 5.10 shows the amount of battery spent by the agents during the execution of the subtasks. Except by two simulations with 17 subtasks where both results were equal, the agents spent less battery with the allocation results of our algorithm. The difference was up to 43% in some simulations with the average of 17%. The results show that a small difference in the task allocation can significantly impact the results of the task execution.

In order to statistically analyse the results, we performed paired t-tests to determine that a significant difference does exist between the results from our solution and the results from the SSIA algorithm. We statistically analysed the utility values, the number of bid messages and the battery values obtained by setting the significance level $\alpha = 0.05$. The t-tests showed that the differences were both statistically significant with p-values lesser than 0.05.

Table 5.13 – Performance results varying the number of subtasks

| | **Our approach** | | | **SSIA** | | |
|---|---|---|---|---|---|---|
| **Subtasks** | **Utility** | **Bids** | **Battery** | **Utility** | **Bids** | **Battery** |
| 17 | 101,9 | 22,6 | 568 | 103,6 | 45,1 | 667 |
| 34 | 226,3 | 49,5 | 1161 | 234,5 | 78,5 | 1393 |
| 51 | 349,7 | 51,3 | 1638 | 358,3 | 127,7 | 1945 |
| 68 | 500,1 | 71,5 | 2200 | 506 | 282,4 | 2524 |

**Final Remarks**

In this chapter, we have empirically evaluated our approach by comparing its results with the optimal solution and the results from other approaches. Our simulation results show that our approach provides near-optimal solutions in all simulation variations. Regarding the comparison with SSIA, ICBAA, TAA and RBTA algorithms, the

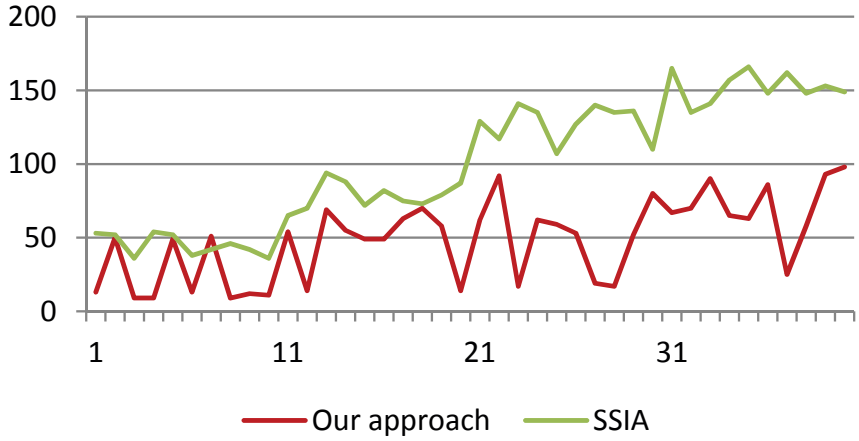


Figure 5.9 – Number of bid messages by simulation.

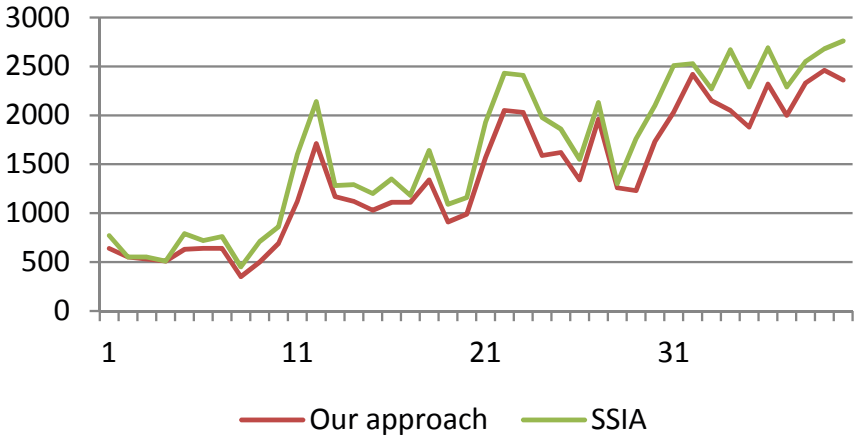


Figure 5.10 – Amount of battery spent by simulation.

results show that our approach performs better than these algorithms. We evaluated the impact of the task allocation results in the execution of the tasks in a complex scenario like the flooding disaster scenario by comparing results from our algorithm with the results from the SSIA algorithm. In all simulations, there was a significant difference in the number of bid messages that were required, and that can be considered an essential difference in some real-world scenarios.

In real-world scenarios, such as flooding disasters, robots may be damaged while executing their allocated tasks, and it may be necessary to reallocate the tasks that were assigned to the failed robot. In some other approaches, identifying the tasks that were allocated to the failed robots can be a challenge. Since at the end of our allocation process each robot knows which robot was responsible for which tasks, in our approach the available robots can easily identify the tasks that were assigned to the failed robot and therefore need to be reallocated.

Although our approach showed good performance in the experiments we carried out, there are real-world scenarios where other approaches in the literature might be more appropriate. For example, in some scenarios agents should be prepared to cooperate and collaborate without pre-coordination, that is, agents are being developed by different organisations and should interact with other agents in the absence of prior knowledge, agreements, and possibly not sharing the same communication protocols and world models [ALS17, SKKR10].

In [MS16], for instance, learning agents must coordinate with other agents to complete a cooperative task, identifying the teammates' strategies and also the tasks to be completed, which is achieved by observing the actions of other agents.

[ALS17] and [SKKR10] provide more details about the problem of collaboration without pre-coordination. [ALS17] also refers to research related to multi-agent interaction without prior coordination, such as [HLZT$^+$17, LV17], while [SKKR10] creates a challenge to the AI community to develop research related to ad hoc agent teams.

# 6. RELATED WORK

There is vast literature related to task allocation in multi-agent or multi-robot systems. Some such work aims at allocating an initial set of tasks to a set of robots, while others focus on the allocation of tasks that arise during the execution (for instance, tasks perceived in the environment or even made available by some type of organisation). Accordingly, we split this section into these two main types of related work.

## 6.1 Allocation of Tasks Perceived or Provided at Runtime

The work by [MSRJ11] introduces a distributed algorithm for task allocation where new tasks can appear, and the set of agents can change at any time. The allocation is performed by forming coalitions, with the objective of finding coalitions which maximise the global utility. The algorithm considers that coalitions are not overlapping, i.e., each agent is allocated to one coalition at a time. The algorithm does not consider constraints between tasks and subtasks.

In the work by [COBT17], task allocation is addressed through distributed planning in each robot using Monte Carlo Tree Search (MCTS). In the scenario they use, the tasks are single item orders that robots need to gather and deliver, where items can have different costs and are distributed in a warehouse. New tasks can be requested at any given time, but the robots have a limited capacity to store items before delivering them. The solution does not consider tasks with constraints and subtasks. Also, they assume that the distribution of the orders is known, which allows them to model the probabilities of tasks appearing at each location.

[KP13] introduce a dynamic task allocation approach where the tasks are grouped based on their distributional information, and then agents are allocated to the groups of tasks instead of tasks directly. The task space is reduced to the same number of agents, and then each subgroup is associated with one of the agents. The solution uses a centralised mediator that coordinates agents through task subgrouping and the allocation of agents to groups through a cost permutation process. Only the decision-making is distributed, with each agent communicating its decisions directly to the mediator. The approach differs from our since there is a central mediator responsible for the allocation, while we are interested in a distributed solution.

[LJGM06] put forward a mathematical model for dynamic task allocation using emergent coordination. In emergent coordination, the robots use only local sensing, and there is little or no direct communication between robots. In the proposal, based on repeated local observations, the robots estimate the state of the environment and choose (based on probabilities given by transition functions) the task to execute. In

that approach, the robots are equally capable of performing any task, and can only be allocated to a single task at a time.

[CMKJ09] propose a decentralised solution for planning agent schedules using a Markov game formulation for tasks with hard deadlines. Each agent is allocated to perform a sequence of tasks. Tasks are discovered during the mission execution, but it is assumed that the task deadlines are always known. All the agents can be allocated to any task (i.e., agents are homogeneous) and have the same costs to perform a given task. The solution does not consider subtasks, only tasks as atomic units.

[CsTPcP14] propose the allocation of new tasks to groups of heterogeneous robots. When new tasks arise, the allocation is initially performed by a centralised algorithm which distributes the tasks to the groups of robots. That approach considers that only one new task should be allocated to a group at a time and that all new tasks need to be allocated. Then the tasks are allocated within each group in a decentralised manner using an auction-based algorithm, and each robot can allocate only one task.

In [GKZ17] a task allocation approach is used in the context of a forest fires fighting scenario, where the tasks (fire spots) are known a priori and provided to the algorithm. The approach uses an auction-based algorithm to allocate the tasks to the agents (UAVs). Assumes that the number of agents is the same as the number of tasks and that agent is able to allocate only one task. The approach does not consider heterogeneous agents.

[US13] consider the allocation of tasks that can be divided into subtasks. The agents are organised in a hierarchical structure composed of manager agents and worker agents (the latter appear only at the bottom level of the hierarchy) which are those that have the resources to execute the tasks. When the first manager in the hierarchy receives a new task for allocation, it divides the task into subtasks and allocates them to the managers directly connected to it in the hierarchy. Each of those managers divides the subtasks into smaller subtasks, and the process is repeated until the subtasks reach the worker agents. The worker agents form a team when they are allocated to perform the subtasks of the same task, and each worker can participate in only one team at a time.

In the work by [GA15], a solution for allocating new tasks discovered by robots during their missions is put forward. The authors only consider atomic tasks. In that work, each task has a specification of the minimum requirements for its execution and robots can play different roles, and each role is based on types of tasks that could be carried out by robots playing that role. The paper focuses on disaster scenarios and proposes the use of heterogeneous robots, in which the robot with the best computational resources plays the role of the coordinator and, consequently, becomes responsible for the coordination of the task allocation process [GA15]. Thus, it could be said of that ap-

proach that there is still a single point of failure within each team, so it is not precisely a decentralised solution like ours.

A decentralised solution for task allocation in multi-robot systems that considers robot resources is presented in [LZK15]. There are scattered consumers in the environment who order services by creating an auction in which robots offer their services through bids. The task is allocated to the provider who can execute it in a shorter time. Each robot tries to allocate one task at a time, and the task is executed in the order in which it was allocated. Thus, the time to execute the new task is added to the time of the other tasks already in the robot execution list.

[TMS15] propose a distributed solution for the reallocation of tasks in order to maximise the number of allocated tasks. The solution focuses on search and rescue scenarios, considering heterogeneous vehicles and tasks with deadlines to start their execution. An algorithm is proposed for the exchange of tasks which aims to increase the number of allocated tasks, even if some of the tasks need to wait a long time to be executed. The basic idea is to 'move' the tasks in the agents' schedules and also change tasks between the agents in order to create spaces for the inclusion of new tasks. In that approach, each vehicle has a limit of tasks that it can perform and each task is allocated to only one vehicle.

[SZB16] proposes an approach for allocating tasks in disaster environments. When an agent finds a task in the environment, it becomes the coordinator for that task, and it shares the information with other agents. A task requires agents with certain work efficiency to complete it in the deadline. Thus agents can be able to partially execute a task, which can be performed by more than one agent at a time. The agents share their work efficiency with the coordinator which chose the agents to perform the task. After allocating one task, the coordinator tries to allocate the next one if any. Tasks also have a different urgent degree.

In [WLL+16] the robots move by the environment and try to allocate each new task they find by using the proposed market-based task allocation algorithm. The approach considers urgency of the time in the task evaluation function. Each robot can allocate at most one task.

[ZMC16] proposes a distributed task allocation method for allocation of atomic tasks to multiple heterogeneous vehicles in a search and rescue scenario. The algorithm iterates between three phases: task inclusion phase, and a consensus and task removal phase. Agents can allocate a maximum number of tasks. Temporal constraints are considered.

In the work by [CYYS16], an auction-based approach considering multiple constraints is proposed. To deal with multiple constraints, the approach computes the cost in four layers (one for each constraint). The auction algorithm is based on the Sequential

Single-Item Auction algorithm. Here only one task is allocated at a time, but the agents can have more tasks allocated according to their capacities.

[AHG17] proposes a decentralised coalition formation approach for allocating tasks with different priorities. When a new task appears the agent near the task becomes the initiator, which is responsible for creating the coalition by contracting other agents who have the resources to complete that task. Tasks have a deadline and a list of subtasks. Each subtask has a resource need to be performed. One task is allocated at time and each agent can allocate at most one task.

Another approach for allocating tasks with different priorities is introduced in [MVDB17]. The idea is to allocate responders to different locations during incidents. There is a limit on the number of responders that can be allocated to a location. Each responder can also be allocated to a limited number of locations. The approach is centralised and does not consider the heterogeneity of the responders.

An auction-based task allocation approach is presented in [IF16]. The idea is to form coalitions with heterogeneous robots to complete the tasks. The allocation works in two steps. First, when the task is announced, and the robot provides bids. The winner becomes the leader for that task. Next, the leader analyses the task and creates an auction for it according to the effort needed to complete the task. Each task always needs more than one robot to be completed forming a coalition. Each robot can participate in only a coalition at a time.

## 6.2 Allocating an Initial Set of Tasks

In the work by [SP13], a distributed solution for task allocation to a set of heterogeneous robots is presented in which robots' capabilities are considered. Unlike our approach, the solution presented in [SP13] requires that only one task is allocated to each robot and that each task is allocated to one robot only.

In [GSW+14], the authors propose a decentralised mechanism for task allocation along with an architecture that focuses on exploring disaster scenarios. Task allocation follows a simple approach based on an auction, where any robot can carry out any task. Each robot can be allocated to more than one task.

A decentralised solution for task allocation among multiple robots based on combinatorial auctions has been introduced by [SGSTS14]. According to the authors, the solution cannot avoid the computational overload characteristic of combinatorial auctions. The use of combinatorial auctions means that the robots provide bids for a combination of tasks rather than individually for each task. At the end of the allocation process, each task must be allocated to exactly one robot, and all tasks must be allocated. The solution considers heterogeneous vehicles with different load capacities.

[SGSTS15] presents a decentralised algorithm for allocating tasks to a set of robots. The solution first relaxes the problem, so that it can be solved optimally or with some approximation through techniques such as linear programming [SGSTS15]. In this step, a maximum consensus-based algorithm is used. The algorithm then converts the found solution back to the original domain.

In [LMB+13] an algorithm is presented to allocate tasks, with each task being allocated to only one robot. The solution proposes the simultaneous execution of auctions. Agents use policies computed through individual Markovian decision processes to calculate the bids for each task. Many approaches for task allocation are based on Markov Decision Processes (MDPs). When it is not possible to have a global view of the environment, Partially Observable MDPs (POMDPs) can be used, which allows each agent to make decisions based only on their observations [LMB+13]. In the case of distributed multi-agent systems, decentralised POMDPs (Dec-POMDPs) are used. However, Dec-POMDPs are known to be intractable in general settings [BZI00].

The solutions presented by [SP13], [GSW+14], [SGSTS14], and [LMB+13], focus specifically on atomic tasks, unlike the approach put forward in this thesis, which comprises other types of tasks as well.

Luo et al. present a similar set of algorithms that focus on different aspects of the task allocation process [LCS13, LCS15b, LCS15a]. In the work by [LCS13], the authors describe a distributed algorithm for the allocation of tasks with deadline to multiple robots. It is considered that all tasks have the same duration, represented by exactly one unit of time. The robots' batteries limit the amount of time available to perform tasks and, consequently, the number of tasks that can be allocated to each agent. The solution works for a number of tasks less than or equal to the sum of the limit of tasks the robots can take. In that approach, any robot can be allocated to any task, and the tasks are independent of each other. The work by [LCS15b] proposes an extension where tasks with different durations are considered. The work presented by [LCS15a] served as an initial inspiration for the mechanism proposed here, although we have departed in many ways from the limitations of that work. The authors present a distributed algorithm focusing on the allocation of groups of tasks. The constraints are in the number of total tasks that a robot can carry out in the mission as well as in the number of tasks carried out by each group. It is assumed that any robot can be allocated to any task. Unlike our work, that work does not consider the allocation of different types of tasks at the same time, aspects related to capabilities of robots, the use of roles associated with tasks is not considered either. We have also changed the way to check for termination of the allocation process as well as changed the bid messages so that our approach requires the exchange of significantly fewer messages.

[FGC14] propose a centralised approach for the allocation of tasks to robots, where the ordering of execution of the allocated tasks is also defined for each robot.

Tasks are considered independent of each other and are classified as atomic or non-atomic tasks. Non-atomic tasks are tasks that can be partially executed, at different times and by different robots, until they are completed. The solution considers that not all robots are able to perform all tasks and that time constraints may prevent all tasks from being executed.

Further, [DMC14] introduce a centralised approach for task allocation that needs to be carried out in parallel, forming temporary coalitions of robots. It considers heterogeneous robots with different capabilities, and each task is divided into a set of subtasks that need a set of capacities to be carried out. Each robot can be allocated only two subtasks at the same time: the subtask being currently carried out and the next subtask to be carried out.

The iterative consensus-based auction algorithm [CBH09]. ICBAA is the execution of the CBAA algorithm iteratively until all the tasks are allocated. CBAA is a single-assignment approach that uses parallel single item auctions, where each agent allocates at most one task. Running CBAA iteratively allow each agent to allocate more tasks. CBAA running iteratively has been used for comparison with new proposed approaches such as in [DMCB11b, IS17, TSM17, DMCB11a].

The approach in [MNG16] considers the allocation of tasks with precedence constraints by iterating over a version of the sequential single-item auction algorithm. At each iteration, a batch of tasks is allocated, more specifically, a batch of tasks that have no precedence other than the tasks already allocated in previous iterations. That is performed by assigning a numerical priority to the tasks in a precomputation step. Tasks without precedence tasks receive high-priority values and so on. The solution allows each agent to be allocated to only one task.

In [Ma16] is proposed an auction algorithm for the allocation of resources. The subsidiaries of a company, represented by agents, submit bids and the resources are allocated in proportion to their bids. The approach considers that the capacity of each agent allows it to allocate resources according to the amount of capacity spent to allocate each resource. Unlike our proposal, the proposed solution is centralised, and it does not consider heterogeneous agents. Another method for the allocation of resources is presented in [ZML18]. Agents try to allocate parts of the resources they want to use, that is, the resources can be allocated in proportion to the bid of the agents. At each iteration, a single agent is randomly chosen to update its best bid. The proposed solution does consider heterogeneous agents.

In sequential single-item auction algorithm [KKT10], the tasks are allocated in multiple rounds. All the tasks are known at the beginning of auctions, and the auctioneer offers all unallocated tasks to the agents. Each agent bids on only one task in each round. The auctioneer determines as the winner only the best bid at each round, i.e. only one task is allocated at each iteration. Due to the efficiency and simplicity of SSIA,

the approach has been used as the basis for several proposals to deal with heterogeneous agents, task precedence constraints, temporal constraints, task reallocation, task allocation with multiple constraints, and so on [MNG16, HP13, WHJ15, NG15, CYYS16, NMG16]. For the same reason, this approach is also often used for comparison with new approaches such as in [LZK15, CNC13, LZK14, SSPO15].

[IS17] presents a decentralised task allocation algorithm based on the Hungarian approach. The solution works for the one-to-one allocation, that is, each agent allocates only one task, and each task is allocated to one agent. The solution only works with atomic tasks, and it does not consider heterogeneous agents. [ZLZ+16] also propose an algorithm based on the Hungarian approach, but to solve the many to many (M–M) allocation problem, where one task can be allocated to many different agents, and one agent can allocate many tasks. However, here the proposed solution is centralised. The solution only works with atomic tasks, and it does not consider heterogeneous agents.

[MSG+17] proposes a coalition formation algorithm for the allocation of tasks to robots in natural disasters. For each task, the solution forms a coalition. Each task has a set of subtasks, and the number of agents in the coalition is the same as the number of subtasks. Each agent is allocated to only one subtask. The solution considers agents' capabilities when forming the coalitions and allocating subtasks to the agents. Unlike our proposal, the proposed solution is centralised.

[dMNdMM16] propose a decentralised algorithm for the allocation of tasks in a swarm of robots. Agents are distributed between the available tasks, and they are always allocated to one task at a time. There are more agents than tasks, and thus many agents are allocated to perform the same task. The approach does not consider heterogeneous agents.

[LLX15] proposes a method to multi-robot task allocation where each robot can allocate multiple tasks. The number of tasks is driven by the capacity of each agent and the amount of capacity needed for each task. Tasks have priority and deadline to be completed. There is a manager responsible for sends the task information to the robots. Each robot computes its own plans and shares it with the manager. The manager allocates the tasks based on the received plans. Heterogeneous robots are not considered.

[LSM14] presents a market-based task allocation algorithm where the price of the items is raised by the merchant, instead of by the bidders as in an auction algorithm. The approach only deal with atomic tasks, heterogeneous agents are not considered, and each agent is able to allocate at most one task.

[DMCB15] proposes a distributed algorithm for the allocation of tasks to heterogeneous robots in a healthcare facility. The idea is based on providing bids in parallel with the execution of a task. While a robot is performing a task, it provides bids to another task. Only when the execution of a task finishes, the agent is allocated to the next

task. In other words, each agent is able to allocate at most one task at a time. Agents are heterogeneous, and tasks are atomic.

[FMPU16] presents a gossip-based algorithm for allocation of tasks in a decentralised way. The algorithm starts from an unfeasible solution, and at each iteration, a node solves a Local-Integer Linear Programming problem. The node is randomly chosen at each iteration. Each agent can allocate up to M tasks and can perform any task (homogeneous agents). Deal only with atomic tasks.

[RM16] presents a centralised approach for allocation of tasks through coalition formation. The solution forms a coalition of robots to each task, i.e. every task needs to be executed by multiple robots. Each robot is allowed to participate in one coalition at a time. The robots are heterogeneous, and the tasks require different capabilities.

## 6.3 Summary of the characteristics of the approaches

Regarding the allocation of tasks perceived or provided at runtime, most of the work mentioned above propose decentralised approaches. Also, most of them consider only the allocation of atomic tasks or non-atomic tasks, that is, tasks which can be partially executed by different agents. Only [AHG17] and [US13] considers compound tasks, that is, tasks with subtasks. Most of the described research deals with heterogeneous entities (agents or robots) considering at least some aspect related to their physical capabilities, except [KP13], [CMKJ09], [LJGM06], [WLL+16], and [MVDB17] which consider homogeneous agents. The agents' capacities also vary between approaches. Many of them consider that each agent can allocate only one task. In others, the agents are able to allocate up to a certain number of tasks (for instance, five tasks). Only [CYYS16] considers that the capacity of each agent allows it to allocate tasks according to the amount of capacity spent to perform each of the tasks. For example, one task can occupy 5% of the agent's capacity, another can occupy 15%, and so on. However, in some of the approaches where agents have the capacity to allocate more than one task such as [GA15], [CYYS16], and [LZK15], the approach restricts to one the number of tasks offered at a time, which also reduces the complexity of the allocation process. Regarding the use of roles in the system, [GA15] defines the leader role within each group of robots, while [US13] use managers and workers in a hierarchical structure. Only [MSRJ11] considers new robots can be added during the process, but the variation in resource availability is not taken into consideration. The approach presented in [GA15] makes considerations on that aspect. Only the approaches proposed in [TMS15], [ZMC16] and [GA15] consider that, at the end of the allocation process, tasks may not have been allocated, either by temporal constraints as described in [TMS15] and [ZMC16], or because there is no robot capable of performing the task as in [GA15]. Some of them consider tasks with priorities, such as [GA15, MVDB17, SZB16, WLL+16, AHG17].

Regarding the allocation of an initial set of tasks, most of the approaches mentioned above propose decentralised solutions and consider a set of tasks being allocated at a time. Also, most of them consider only the allocation of atomic tasks or non-atomic tasks. Only [LCS13], [LCS15a], [LCS15b], [MSG+17] and [DMC14] consider compound tasks, that is, tasks with subtasks. Some of the described research deals with heterogeneous entities considering at least some aspect related to their physical capabilities, but most of them consider homogeneous agents. The agents' capacities also vary between approaches. Some of them consider that each agent can allocate only one task while some consider agents capable of allocating up to a certain number of tasks. Other approaches consider that the capacity of each agent allows it to allocate tasks according to the amount of capacity spent to perform each of the tasks. None of the approaches considers the use of roles. Few approaches also consider tasks with priorities, such as [LLX15, DMCB15, MNG16] and some other, such as [LLX15, MNG16, FGC14, LCS15b, LCS13] consider temporal constraints. Only the approaches proposed in [FGC14, ZML18, DMC14, LSM14, DMCB15] consider that, at the end of the allocation process, tasks may not have been allocated. Unlike our solution, none of the approaches which consider compound tasks perform some control over whether the tasks were not completely allocated, that is, if there is a task in which at least one subtask was not allocated to any agent.

Table 6.1 presents a summary of the characteristics of the approaches presented in this chapter. The "Availability" column refers to whether tasks are delivered as an initial set of tasks (Initial) or whether they arrive during the mission being perceived or provided (Arrive). Column "Dec." indicates whether the approach is decentralised or not. "Type of task" column indicates what type of task the approach deals with. "Task at time" column indicates whether the task allocation process receives only one task at a time for allocation (one) or if it receives multiple tasks at a time (many). "Agent's capacity" column refers to the capability of the agents considered in the approach. If each agent is able to allocate only one task (one), if each agent are able to allocate up to a certain number of tasks (n tasks), or if the capacity of each agent allows it to allocate tasks according to the amount of capacity spent to perform each of the tasks (amount). Column "Heterog. Agents" indicates whether or not the approach considers heterogeneous agents, that is, the approach considers at least some aspect related to the agent's capabilities, which may impact on the tasks the agents can allocate. Column "Roles" indicates whether or not the approach considers that agents may play different roles. "Tasks not allocated" column indicates whether the approach allows unallocated tasks at the end of the allocation process, either because the agents do not have the required capabilities to perform all the tasks or because there are more tasks than the agents' capacities. "Temporal constraint" column indicates whether the approach considers tasks with temporal constraints. "Task Priority" column indicates whether the approach considers tasks with different priorities.

Table 6.1 – Characteristics of the approaches.

| Approach | Availability | Dec. | Type of task | Tasks at a time | Agent's capacity | Heterog. agents | Roles | Tasks not allocated | Temporal constraint | Task priority |
|---|---|---|---|---|---|---|---|---|---|---|
| [CYYS16] | arrive | √ | atomic | one | amount | √ | | | √ | |
| [GA15] | arrive | √ | atomic | one | n tasks | √ | √ | √ | | √ |
| [LZK15] | arrive | √ | atomic | one | no limit | √ | | | √ | |
| [AHG17] | arrive | √ | compound | one | one | √ | | | √ | √ |
| [SZB16] | arrive | √ | non-atomic | many | one | √ | | | √ | √ |
| [WLL+16] | arrive | √ | non-atomic | one | one | | | | | √ |
| [MSRJ11] | arrive | √ | non-atomic | many | one | | | √ | | |
| [KP13] | arrive | | atomic | many | n tasks | | | | | |
| [LJGM06] | arrive | √ | atomic | many | one | | | | | |
| [CMKJ09] | arrive | √ | atomic | many | one | | | √ | √ | |
| [COBT17] | arrive | √ | atomic | many | n tasks | | | √ | | √ |
| [IF16] | arrive | √ | non-atomic | one | one | √ | | | | |
| [CsTPcP14] | arrive | mixed | atomic | many | one | √ | | | | |
| [MVDB17] | arrive | | non-atomic | many | n tasks | | | | √ | √ |
| [ZMC16] | arrive | √ | atomic | many | n tasks | √ | | √ | √ | |
| [TMS15] | arrive | √ | atomic | many | n tasks | √ | | √ | √ | |
| [US13] | arrive | √ | compound | many | one | √ | √ | | √ | |
| [LLX15] | initial | √ | atomic | many | amount | | | | √ | √ |
| [MNG16] | initial | √ | atomic | many | one | | | | √ | √ |
| [ZML18] | initial | √ | non-atomic | many | amount | | | √ | | |
| [FGC14] | initial | | atomic non-atomic | many | amount | √ | | √ | √ | |
| [MSG+17] | initial | | compound | many | one | √ | | | | |
| [DMC14] | initial | | compound | many | one | √ | | √ | | |
| [Ma16] | initial | | non-atomic | many | amount | | | | | |
| [ZLZ+16] | initial | | non-atomic | many | n tasks | | | | | |

**Table 6.1 continued from previous page**

| Approach | Availability | Dec. | Type of task | Tasks at a time | Agent's capacity | Heterog. agents | Roles | Tasks not allocated | Temporal constraint | Task priority |
|---|---|---|---|---|---|---|---|---|---|---|
| [RM16] | initial | | non-atomic | many | one | √ | | | | |
| [LMB+13] | initial | √ | atomic | many | n tasks | | | | | |
| [SGSTS14] | initial | √ | atomic | many | n tasks | √ | | | | |
| [SGSTS15] | initial | √ | atomic | many | n tasks | √ | | | | |
| [FMPU16] | initial | √ | atomic | many | n tasks | | | | | |
| [SP13] | initial | √ | atomic | many | one | √ | | | | |
| [GKZ17] | initial | √ | atomic | many | one | | | | | |
| [IS17] | initial | √ | atomic | many | one | | | | | |
| [LSM14] | initial | √ | atomic | many | one | | | √ | | |
| [DMCB15] | initial | √ | atomic | many | one | √ | | √ | | √ |
| [GSW+14] | initial | √ | atomic | one | n tasks | | | | | |
| [LCS15b] | initial | √ | compound | many | amount | | | | √ | |
| [LCS13] | initial | √ | compound | many | n tasks | | | | √ | |
| [LCS15a] | initial | √ | compound | many | n tasks | √ | | | | |
| [dMNdMM16] | initial | √ | non-atomic | many | one | | | | | |
| Our approach | initial arrive | √ | compound | many | amount | √ | √ | √ | | |

# 7. CONCLUSION

In this thesis, we described a decentralised mechanism for the allocation of different types of tasks to heterogeneous agent teams, considering that they can play different roles and carry out tasks according to their own capabilities. Task allocation is an important and challenging aspect considered in coordination problems in multi-agent systems. A process to allocate tasks efficiently is crucial in many domains, especially if we consider that different tasks can be performed by agents with different capabilities. Efficiently allocating tasks among multiple agents with multiple constraints requires solving a challenging optimisation problem. In chapter 3 we formally state the Multi-Agent Task Allocation optimisation problem addressed in this thesis.

As described in Chapter 6, even though there are several approaches proposed for the allocation of tasks in both MASs and multi-robot systems, there is still much to be done in this area, since the available solutions only include a subset of characteristics required to allocate tasks in many domains efficiently. The complexity in dealing with different characteristics in an allocation mechanism was perceived during the literature review, where it was found that the approaches are characterised by dealing with one or few characteristics. Approaches that deal with more characteristics, usually consider rather restrictive premises or have high computational cost.

The main objective of the proposed mechanism is to find an assignment that maximises the sum of the utilities obtained by each agent individually while satisfying all constraints. This is based on the idea that the process of maximising individual utilities simultaneously improves the global utility. The agents agree on the allocation of tasks by exchanging messages with bid values for the tasks they intend to execute.

The proposed mechanism deals with the allocation of a set of tasks being made available at the same time. The mechanism considers decomposable simple tasks, for which all the subtasks need to be allocated to the same agent, compound tasks where each subtask needs to be allocated to a different agent and compound tasks where subtasks can be allocated to any agent. Also, the proposed mechanism supports the allocation of atomic tasks by creating a compound task with all the atomic tasks. The proposed mechanism allows us to use all those types of tasks and also to express other constraints through the definition minimum and maximum values for the number of subtasks an agent can take from a task type.

Regarding the agents, our approach deals with heterogeneous agents, i.e. agents with different capabilities and resources. The capabilities constraint the roles the agents are able to play, which are necessary for the execution of certain tasks. Also, each agent in the proposed solution has a limit on the number of subtasks that it can allocate to itself, and this limit may be different for individual agents. We consider that the capacity of each agent allows it to allocate subtasks according to the capacity required to perform each of the subtasks. Thus, each agent needs to select the best tasks to allocate based on its capacity and on the capacities required for each task, which is itself a combinatorial optimisation problem. We have solved it with the knapsack approach.

When the total capacity of the agents is greater than or equal to the total capacity number of subtasks that need to be allocated, the base algorithms yield very good results. When the capacities necessary to perform the subtasks is greater than the to-

tal capacity of the agents, our approach performs extra steps in order to identify tasks which are not entirely allocated, and then we use a social-choice algorithm based on voting followed by our allocation algorithms to allocate at least some of those outstanding tasks entirely.

We also described our approach to identify the end of the allocation process, which is important in a decentralised mechanism where each agent needs to identify when the other agents finished sending bids. Our approach reduces the number of bid messages each agent needs to send to other agents, and still allows a precise procedure to check for termination.

## 7.1 Summary of Results

The results of the proposed mechanism were compared with the optimal solutions obtained with GLPK. We also compare the performance of our mechanism with other solutions: ICBAA, SSIA, TAA, and RBTA. We also evaluate the impact of our task allocation approach in the execution of tasks in a complex scenario like the flooding disaster scenario.

Our results show that our approach provides near-optimal solutions when compared with results obtained with GLPK. Regarding the comparison with ICBAA, SSIA, TAA, and RBTA algorithms, the results show that our approach performs better than these algorithms. Also, there is a significant difference in the number of bid messages that were required during the allocation process. Our approach required a much smaller number of bids in comparison with the other approaches, which can be considered an important aspect for applications in real-world scenarios.

For the evaluation in the flooding disaster scenario, we described the features of the simulator we extended for evaluating our approach. The scenario consists of a number of agents, which can move through flooded regions over the map of a realistic city and execute some actions to complete the tasks, such as take_picture, sample_water, pickup_box, rescue_victim and others. We compared the performance of our mechanism with the SSIA algorithm.

The results show that our approach performs better in the flooding scenario. Again, our approach required a much smaller number of bids, which can be considered an essential aspect for this scenario. The results show that even a small difference in the task allocation results can significantly impact the results of the task execution.

## 7.2 Future Work

Future work aims to consider aspects such as task prioritising, temporal constraints and task precedence constraints. We also intend to evaluate the approach with real agents as mentioned earlier or on ROS-based simulations that are currently being developed.

Also, in real-world scenarios, such as flooding disasters, robots may be damaged while executing their allocated tasks, and it may be necessary to reallocate the tasks

that were assigned to the failed robot. Although in our approach identifying the tasks that were allocated to the failed robots is facilitated by the fact that at the end of our allocation process each robot knows which robot was responsible for which tasks, we need to define strategies and the best way to reallocate the tasks.

During the allocation process, in decentralised approaches, there are often communication requirements, as participants need to share information. Maintaining data integrity, resilience, and security in data access are some of the important features of this type of solution. In that direction, we plan to integrate and evaluate the architecture we propose in [BMZB18] for supporting the dynamic and decentralised allocation of tasks built on the idea of having communication and coordination in a multi-agent system through a private blockchain.

# REFERENCES


[AAT13] Aghaeeyan, A.; Abdollahi, F.; Talebi, H. A. “Robust cooperative control in the presence of obstacles”. In: Proceedings of the Iranian Conference on Electrical Engineering, 2013, pp. 1–6.

[AHG17] Ayari, E.; Hadouaj, S.; Ghedira, K. “A dynamic decentralised coalition formation approach for task allocation under tasks priority constraints”. In: Proceedings of the International Conference on Advanced Robotics, 2017, pp. 250–255.

[ALS17] Albrecht, S. V.; Liemhetcharat, S.; Stone, P. “Special issue on multiagent interaction without prior coordination: guest editorial”, *Autonomous Agents and Multi-Agent Systems*, vol. 31-4, Jul 2017, pp. 765–766.

[BA95] Balch, T.; Arkin, R. C. “Communication in reactive multiagent robotic systems”, *Autonomous Robots*, vol. 1-1, Feb 1995, pp. 27–52.

[BB17] Basegio, T. L.; Bordini, R. H. “An algorithm for allocating structured tasks in multi-robot scenarios”. In: Proceedings of the KES International Conference on Agent and Multi-Agent Systems, 2017, pp. 99–109.

[BBH+13] Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; Santi, A. “Multi-agent oriented programming with jacamo”, *Science of Computer Programming*, vol. 78-6, Jun 2013, pp. 747–761.

[BBO+02] Brazier, F.; Brazier, F. M. T.; Overeinder, B.; Wijngaards, N.; Mobach, D.; Overeinder, B. J.; Wijngaards, N. J. E. “Supporting life cycle coordination in open agent systems”. In: Proceedings of the Multi-agent Systems Problem Spaces Workshop, 2002, pp. 4.

[BBT02] Bennewitz, M.; Burgard, W.; Thrun, S. “Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots”, *Robotics and Autonomous Systems*, vol. 41-2/3, Nov 2002, pp. 89–99.

[BD13] Bordini, R. H.; Dix, J. “Programming multiagent systems”. MIT Press, 2013, chap. 13, pp. 587–639.

[BHH+10] Behrens, T.; Hindriks, K.; Hübner, J.; Behrens, T.; Hindriks, K.; Hübner, J.; Dastani, M. “Putting apl platforms to the test: agent similarity and execution performance”, Technical report, Clausthal University of Technology, 2010, 23p.

[BHW07] Bordini, R. H.; Hübner, J. F.; Wooldridge, M. “Programming multi-agent systems in agentspeak using jason”. John Wiley & Sons, 2007, 292p.

[BIP88] Bratman, M. E.; Israel, D. J.; Pollack, M. E. “Plans and resource-bounded practical reasoning”, *Computational Intelligence*, vol. 4-4, Sep 1988, pp. 349–355.

[BM03] Bussab, W. d. O.; Moretin, P. A. “Estatística básica”. Saraiva, 2003, 526p.

[BMZB18] Basegio, T. L.; Michelin, R. A.; Zorzo, A. F.; Bordini, R. H. “A decentralised approach to task allocation using blockchain”. In: Proceedings of the International Workshop on Engineering Multi-Agent Systems, 2018, pp. 75–91.

[BPL05] Braubach, L.; Pokahr, A.; Lamersdorf, W. “Jadex: A bdi-agent system combining middleware and reasoning”. In: *Software Agent-Based Applications, Platforms and Development Kits*, Birkhäuser Basel, 2005, chap. 7, pp. 143–168.

[BRHL99] Busetta, P.; Ronnquist, R.; Hodgson, A.; Lucas, A. “Jack intelligent agents - components for intelligent agents in java”, *AgentLink*, vol. 2-1, Jan 1999, pp. 2–5.

[BZI00] Bernstein, D. S.; Zilberstein, S.; Immerman, N. “The complexity of decentralized control of markov decision processes”. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2000, pp. 32–37.

[CBH09] Choi, H.-L.; Brunet, L.; How, J. P. “Consensus-based decentralized auctions for robust task allocation”, *IEEE Transactions on Robotics*, vol. 25-4, Aug 2009, pp. 912–926.

[CFNM15] Casadei, G.; Furci, M.; Naldi, R.; Marconi, L. “Quadrotors motion coordination using consensus principle”. In: Proceedings of the American Control Conference, 2015, pp. 3842–3847.

[CGP15] Cepeda-Gomez, R.; Perico, L. F. “Formation control of nonholonomic vehicles under time delayed communications”, *IEEE Transactions on Automation Science and Engineering*, vol. 12-3, Jul 2015, pp. 819–826.

[CHB13] Cardoso, R. C.; Hubner, J. F.; Bordini, R. H. “Benchmarking communication in actor- and agent-based languages”. In: Proceedings of the International Workshop on Engineering Multi-Agent Systems, 2013, pp. 58–77.

[CKB+18] Cardoso, R. C.; Krausburg, T.; Baségio, T.; Engelmann, D. C.; Hübner, J. F.; Bordini, R. H. “Smart-jacamo: an organization-based team for the multi-agent programming contest”, *Annals of Mathematics and Artificial Intelligence*, vol. 84-1/2, Oct 2018, pp. 75–93.

[CMKJ09] Chapman, A. C.; Micillo, R. A.; Kota, R.; Jennings, N. R. “Decentralised dynamic task allocation: a practical game: theoretic approach”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2009, pp. 915–922.

[CNC13] Cavalcante, R. C.; Noronha, T. F.; Chaimowicz, L. “Improving combinatorial auctions for multi-robot exploration”. In: Proceedings of the International Conference on Advanced Robotics, 2013, pp. 1–6.

[COBT17] Claes, D.; Oliehoek, F.; Baier, H.; Tuyls, K. “Decentralised online planning for multi-robot warehouse commissioning”. In: Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 492–500.

[Con10] Conitzer, V. “Comparing multiagent systems research in combinatorial auctions and voting”, *Annals of Mathematics and Artificial Intelligence*, vol. 58-3/4, Apr 2010, pp. 239–259.

[Cor14] Correa, A. “Distributed team formation in urban disaster environments”. In: Proceedings of the IEEE Symposium on Intelligent Agents, 2014, pp. 57–64.

[Cou07] Council, E. “Eu directive of the european parliament and of the european council on the estimation and management of flood risks (2007/60/eu)”. Available at: http://ec.europa.eu/environment/water/flood_risk/index.htm, May 2017.

[CPK+18] Cardoso, R. C.; Pereira, R. F.; Krzisch, G.; Magnaguagno, M. C.; Baségio, T.; Meneguzzi, F. “Team pucrs: a decentralised multi-agent solution for the agents in the city scenario”, *International Journal of Agent-Oriented Software Engineering*, vol. 6-1, Jan 2018, pp. 3–34.

[Cra17] Crasar. “Center for robot-assisted search and rescue”. Available at: http://crasar.org, May 2017.

[CsTPcP14] Cao, L.; shun Tan, H.; Peng, H.; cong Pan, M. “Multiple uavs hierarchical dynamic task allocation based on pso-fsa and decentralized auction”. In: Proceedings of the IEEE International Conference on Robotics and Biomimetics, 2014, pp. 2368–2373.

[CSVJRC14] Corona-Sánchez, J. J.; Vargas-Jacob, J. A.; Rodríguez-Cortés, H. “Decentralized real time implementation of a leader-follower coordination strategy for quadrotors”. In: Proceedings of the International Conference on Unmanned Aircraft Systems, 2014, pp. 237–243.

[CYYS16] Cheng, Q.; Yin, D.; Yang, J.; Shen, L. “An auction-based multiple constraints task allocation algorithm for multi-uav system”. In: Proceedings of the International Conference on Cybernetics, Robotics and Control, 2016, pp. 1–5.

[CZHB13] Cardoso, R. C.; Zatelli, M. R.; Hübner, J. F.; Bordini, R. H. “Towards benchmarking actor- and agent-based programming languages”. In: Proceedings of the International Workshop on Programming Based on Actors, Agents and Decentralized Control, 2013, pp. 115–126.

[Das08] Dastani, M. “2apl: a practical agent programming language”, *Autonomous Agents and Multi-Agent Systems*, vol. 16-3, Jun 2008, pp. 214–248.

[DC98] Drogoul, A.; Collinot, A. “Applying an agent-oriented methodology to the design of artificial organizations: a case study in robotic soccer”, *Autonomous Agents and Multi-Agent Systems*, vol. 1-1, Jan 1998, pp. 113–129.

[dCFS13] de Campos, G. R.; Falcone, P.; Sjöberg, J. “Autonomous cooperative driving: a velocity-based negotiation approach for intersection crossing”. In: Proceedings of the International IEEE Conference on Intelligent Transportation Systems, 2013, pp. 1456–1461.

[Dec96] Decker, K. S. “TÆms: a framework for environment centered analysis & design of coordination mechanisms”. John Wiley & Sons, 1996, chap. 16, pp. 429–448.

[DMC14] Das, G. P.; McGinnity, T. M.; Coleman, S. A. “Simultaneous allocations of multiple tightly-coupled multi-robot tasks to coalitions of heterogeneous robots”. In: Proceedings of the IEEE International Conference on Robotics and Biomimetics, 2014, pp. 1198–1204.

[DMCB11a] Das, G.; McGinnity, T.; Coleman, S.; Behera, L. “A fast distributed auction and consensus process for allocation of prioritised tasks in multi-robot systems”. In: Proceedings of the Irish Conference on Artificial Intelligence and Cognitive Science, 2011, pp. 244–253.

[DMCB11b] Das, G. P.; McGinnity, T. M.; Coleman, S. A.; Behera, L. “A fast distributed auction and consensus process using parallel task allocation and execution”. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 4716–4721.

[DMCB15] Das, G. P.; McGinnity, T. M.; Coleman, S. A.; Behera, L. “A distributed task allocation algorithm for a multi-robot system in healthcare facilities”, *Journal of Intelligent & Robotic Systems*, vol. 80-1, Oct 2015, pp. 33–58.

[dMNdMM16] de Mendonça, R. M.; Nedjah, N.; de Macedo Mourelle, L. “Efficient distributed algorithm of dynamic task assignment for swarm robotics”, *Neurocomputing*, vol. 172-1, Jan 2016, pp. 345–355.

[DP13] Dignum, V.; Padget, J. “Multiagent organizations”. MIT Press, 2013, chap. 2, pp. 51–98.

[DS08] Dresner, K.; Stone, P. “A multiagent approach to autonomous intersection management”, *Journal of Artificial Intelligence Research*, vol. 31-1, Mar 2008, pp. 591–656.

[DZKS06] Dias, M. B.; Zlot, R.; Kalra, N.; Stentz, A. “Market-based multirobot coordination: a survey and analysis”, *Proceedings of the IEEE*, vol. 94-7, Jul 2006, pp. 1257–1270.

[FGC14] Flushing, E. F.; Gambardella, L. M.; Caro, G. A. D. “A mathematical programming approach to collaborative missions with heterogeneous teams”. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014, pp. 396–403.

[FMPU16] Fanti, M. P.; Mangini, A. M.; Pedroncelli, G.; Ukovich, W. “Discrete consensus for asynchronous distributed task assignment”. In: Proceedings of the IEEE Conference on Decision and Control, 2016, pp. 251–255.

[FRD98] Friedrich, H.; Rogalla, O.; Dillmann, R. “Integrating skills into multi-agent systems”, *Journal of Intelligent Manufacturing*, vol. 9-2, Mar 1998, pp. 119–127.

[FRR+15] Fischer, J. E.; Reeves, S.; Rodden, T.; Reece, S.; Ramchurn, S. D.; Jones, D. “Building a birds eye view: collaborative work in disaster response”. In: Proceedings of the Annual ACM Conference on Human Factors in Computing Systems, 2015, pp. 4103–4112.

[FUMP13] Fanti, M. P.; Ukovich, W.; Mangini, A. M.; Pedroncelli, G. “A quantized consensus algorithm for a multi-agent assignment problem”. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2013, pp. 1063–1068.

[GA13] Gunn, T.; Anderson, J. “Effective task allocation for evolving multi-robot teams in dangerous environments”. In: Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2013, pp. 231–238.

[GA15] Gunn, T.; Anderson, J. “Dynamic heterogeneous team formation for robotic urban search and rescue”, *Journal of Computer and System Sciences*, vol. 81-3, May 2015, pp. 553–567.

[GKZ17] Ghamry, K. A.; Kamel, M. A.; Zhang, Y. “Multiple uavs in forest fire fighting mission using particle swarm optimization”. In: Proceedings of the International Conference on Unmanned Aircraft Systems, 2017, pp. 1404–1409.

[GM04] Gerkey, B. P.; Mataric, M. J. “A formal analysis and taxonomy of task allocation in multi-robot systems”, *International Journal of Robotics Research*, vol. 23-9, Sep 2004, pp. 939–954.

[GSW+14] Gernert, B.; Schildt, S.; Wolf, L.; Zeise, B.; Fritsche, P.; Wagner, B.; Fiosins, M.; Manesh, R. S.; Muller, J. P. “An interdisciplinary approach to autonomous team-based exploration in disaster scenarios”. In: Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics, 2014, pp. 1–8.

[Hay85] Hayes-Roth, B. “A blackboard architecture for control”, *Artificial Intelligence*, vol. 26-1, Jul 1985, pp. 251–321.

[HBHM01] Hindriks, K. V.; Boer, F. S. d.; Hoek, W. v. d.; Meyer, J.-J. C. “Agent programming with declarative goals”. In: Proceedings of the International Workshop on Intelligent Agents, 2001, pp. 228–243.

[HLZT+17] Hernandez-Leal, P.; Zhan, Y.; Taylor, M. E.; Sucar, L. E.; Munoz de Cote, E. “Efficiently detecting switches against non-stationary opponents”, *Autonomous Agents and Multi-Agent Systems*, vol. 31-4, Jul 2017, pp. 767–789.

[HP13] Heap, B.; Pagnucco, M. “Repeated auctions for reallocation of tasks with pickup and delivery upon robot failure”. In: Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems, 2013, pp. 461–469.

[HS99] Huhns, M.; Stephens, L. M. “Multiagent systems and societies of agents”. MIT Press, 1999, chap. 2, pp. 79–120.

[HSB07] Hübner, J. F.; Sichman, J. S.; Boissier, O. “Developing organised multi-agent systems using the moise+ model: programming issues at the system and agent levels”, *International Journal of Agent-Oriented Software Engineering*, vol. 1-3/4, Jan 2007, pp. 370–395.

[Hui10] Hui, N. B. “Coordinated motion planning of multiple mobile robots using potential field method”. In: Proceedings of the International Conference on Industrial Electronics, Control Robotics, 2010, pp. 6–11.

[IF16] Irfan, M.; Farooq, A. “Auction-based task allocation scheme for dynamic coalition formations in limited robotic swarms with heterogeneous capabilities”. In: Proceedings of the International Conference on Intelligent Systems Engineering, 2016, pp. 210–215.

[IFR15] IFRC. “World disasters report 2015: focus on local actors and humanitarian action”. Available at: http://ifrc-media.org/interactive/world-disasters-report-2015/, November 2017.

[ILS07] Innocenti, B.; López, B.; Salvi, J. “A multi-agent architecture with cooperative fuzzy control for a mobile robot”, *Robotics and Autonomous Systems*, vol. 55-12, Dec 2007, pp. 881–891.

[IS17] Ismail, S.; Sun, L. “Decentralized hungarian-based approach for fast and scalable task allocation”. In: Proceedings of the International Conference on Unmanned Aircraft Systems, 2017, pp. 23–28.

[Jia16] Jiang, Y. “A survey of task allocation and load balancing in distributed systems”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 27-2, Feb 2016, pp. 585–599.

[JM13] Jia, X.; Meng, M. Q. H. “A survey and analysis of task allocation algorithms in multi-robot systems”. In: Proceedings of the IEEE International Conference on Robotics and Biomimetics, 2013, pp. 2280–2285.

[KKT10] Koenig, S.; Keskinocak, P.; Tovey, C. “Progress on agent coordination with cooperative auctions”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2010, pp. 1713–1717.

[KP13] Keshmiri, S.; Payandeh, S. “Multi-robot, dynamic task allocation: a case study”, *Intelligent Service Robotics*, vol. 6-3, Mar 2013, pp. 137–154.

[KSD13] Korsah, G. A.; Stentz, A.; Dias, M. B. “A comprehensive taxonomy for multi-robot task allocation”, *International Journal of Robotics Research*, vol. 32-12, Oct 2013, pp. 1495–1512.

[LCS13] Luo, L.; Chakraborty, N.; Sycara, K. “Distributed algorithm design for multi-robot task assignment with deadlines for tasks”. In: Proceedings of the IEEE International Conference on Robotics and Automation, 2013, pp. 3007–3013.

[LCS15a] Luo, L.; Chakraborty, N.; Sycara, K. “Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks”, *IEEE Transactions on Robotics*, vol. 31-1, Feb 2015, pp. 19–30.

[LCS15b] Luo, L.; Chakraborty, N.; Sycara, K. “Distributed algorithms for multirobot task assignment with task deadline constraints”, *IEEE Transactions on Automation Science and Engineering*, vol. 12-3, Jul 2015, pp. 876–888.

[LHK13] Lee, D.-H.; Han, J.-H.; Kim, J.-H. “Market-based multiagent framework for balanced task allocation”. Springer Berlin Heidelberg, 2013, chap. 2, pp. 549–559.

[LJGM06] Lerman, K.; Jones, C. V.; Galstyan, A.; Mataric, M. J. “Analysis of dynamic task allocation in multi-robot systems”, *The International Journal of Robotics Research*, vol. 25-3, Mar 2006, pp. 225–241.

[LLX15] Liu, F.; Liang, S.; Xian, X. “Multi-robot task allocation based on utility and distributed computing and centralized determination”. In: Proceedings of the Chinese Control and Decision Conference, 2015, pp. 3259–3264.

[LMB+13] Lozenguez, G.; Mouaddib, A. I.; Beynier, A.; Adouane, L.; Martinet, P. “Simultaneous auctions for "rendez-vous" coordination phases in multi-robot multi-task mission”. In: Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2013, pp. 67–74.

[LMP03] Luck, M.; McBurney, P.; Preist, C. “Agent technology: enabling next generation computing - a roadmap for agent based computing”. AgentLink, 2003, 102p.

[LSM14] Liu, L.; Shell, D. A.; Michael, N. “From selfish auctioning to incentivized marketing”, *Autonomous Robots*, vol. 37-4, Dec 2014, pp. 417–430.

[LTL+15] Li, G.; Tong, S.; Li, Y.; Cong, F.; Tong, Z.; Yamashita, A.; Asama, H. “Hybrid dynamical moving task allocation methodology for distributed multi-robot coordination system”. In: Proceedings of the IEEE International Conference on Mechatronics and Automation, 2015, pp. 1412–1417.

[LV17] Liemhetcharat, S.; Veloso, M. “Allocating training instances to learning agents for team formation”, *Autonomous Agents and Multi-Agent Systems*, vol. 31-4, Jul 2017, pp. 905–940.

[LZ03] Lau, H. C.; Zhang, L. “Task allocation via multi-agent coalition formation: taxonomy, algorithms and complexity”. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence, 2003, pp. 346–350.

[LZK14] Lee, D.; Zaheer, S. A.; Kim, J. “Ad hoc network-based task allocation with resource-aware cost generation for multirobot systems”, *IEEE Transactions on Industrial Electronics*, vol. 61-12, Dec 2014, pp. 6871–6881.

[LZK15] Lee, D. H.; Zaheer, S. A.; Kim, J. H. “A resource-oriented, decentralized auction algorithm for multirobot task allocation”, *IEEE Transactions on Automation Science and Engineering*, vol. 12-4, Oct 2015, pp. 1469–1481.

[Ma16] Ma, R. T. B. “Efficient resource allocation and consolidation with selfish agents: an adaptive auction approach”. In: Proceedings of the IEEE International Conference on Distributed Computing Systems, 2016, pp. 497–508.

[Mak16] Makhorin, A. “Gnu linear programming kit reference manual”. Available at: http://www.gnu.org/software/glpk/glpk.html, April 2018.

[Mey70] Meyer, P. L. “Introductory probability and statistical applications”. Addison Wesley, 1970, 367p.

[MNG16] McIntire, M.; Nunes, E.; Gini, M. “Iterated multi-robot auctions for precedence-constrained task scheduling”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2016, pp. 1078–1086.

[MS16] Melo, F. S.; Sardinha, A. “Ad hoc teamwork by learning teammates’ task”, *Autonomous Agents and Multi-Agent Systems*, vol. 30-2, Mar 2016, pp. 175–219.

[MSG+17] Mouradian, C.; Sahoo, J.; Glitho, R. H.; Morrow, M. J.; Polakos, P. A. “A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters”. In: Proceedings of the International Conference on Wireless Communications and Mobile Computing, 2017, pp. 1909–1914.

[MSRJ11] Macarthur, K. S.; Stranders, R.; Ramchurn, S. D.; Jennings, N. R. “A distributed anytime algorithm for dynamic task allocation in multi-agent systems”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2011, pp. 701–706.

[MTN+08] Murphy, R. R.; Tadokoro, S.; Nardi, D.; Jacoff, A.; Fiorini, P.; Choset, H.; Erkmen, A. M. “Springer handbook of robotics”. Springer Berlin Heidelberg, 2008, chap. 50, pp. 1151–1173.

[Mur14] Murphy, R. R. “Disaster robotics”. The MIT Press, 2014, 240p.

[MVDB17] Mukhopadhyay, A.; Vorobeychik, Y.; Dubey, A.; Biswas, G. “Prioritized allocation of emergency responders based on a continuous-time incident prediction model”. In: Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 168–177.

[NG15] Nunes, E.; Gini, M. “Multi-robot auctions for allocation of tasks with temporal constraints”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2015, pp. 2110–2216.

[NMG16] Nunes, E.; McIntire, M.; Gini, M. “Decentralized allocation of tasks with temporal and precedence constraints to a team of robots”. In: Proceedings of the IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots, 2016, pp. 197–202.

[NS14] Nazar, M.; Sadik, S. “Ontology based goal and task allocation for autonomous agents”. In: Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery, 2014, pp. 924–929.

[Par14] Parker, J. “Coordination in large scale multi-agent systems for complex environments”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2014, pp. 1751–1752.

[PGCM+13] Pujol-Gonzalez, M.; Cerquides, J.; Meseguer, P.; Rodríguez-Aguilar, J. A.; Tambe, M. “Engineering the decentralized coordination of uavs with limited communication range”. In: Proceedings of the Conference of the Spanish Association for Artificial Intelligence, 2013, pp. 199–208.

[PGL86] P. Georgeff, M.; Lansky, A. “Procedural knowledge”. In: Proceedings of the Institute of Electrical and Electronics Engineers (Special Issue on Knowledge Representation), 1986, pp. 1383–1398.

[RB07] Rudenko, D.; Borisov, A. “An overview of blackboard architecture application for real tasks”. In: Scientific Proceedings of the Riga Technical University, 2007, pp. 50–57.

[RG95] Rao, A. S.; Georgeff, M. P. “Bdi agents: from theory to practice”. In: Proceedings of the International Conference on Multi-agent Systems, 1995, pp. 312–319.

[RGG14] Rodriguez, S.; Gaud, N.; Galland, S. “Sarl: a general-purpose agent-oriented programming language”. In: Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2014, pp. 103–110.

[RHI+15] Ramchurn, S. D.; Huynh, T. D.; Ikuno, Y.; Flann, J.; Wu, F.; Moreau, L.; Jennings, N. R.; Fischer, J. E.; Jiang, W.; Rodden, T.; Simpson, E.; Reece, S.; Roberts, S. J. “Hac-er: a disaster response system based on human-agent collectives”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2015, pp. 533–541.

[Ric14] Ricci, A. “From actor event-loop to agent control-loop: impact on programming”. In: Proceedings of the International Workshop on Programming Based on Actors, Agents and Decentralized Control, 2014, pp. 121–132.

[RM16] Rauniyar, A.; Muhuri, P. K. “Multi-robot coalition formation problem: task allocation with adaptive immigrants based genetic algorithms”. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2016, pp. 137–142.

[RN09] Russell, S.; Norvig, P. “Artificial intelligence: a modern approach”. Prentice Hall Press, 2009, 1132p.

[RPF+10] Ramchurn, S. D.; Polukarov, M.; Farinelli, A.; Truong, C.; Jennings, N. R. “Coalition formation with spatial and temporal constraints”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1181–1188.

[RPV11] Ricci, A.; Piunti, M.; Viroli, M. “Environment programming in multi-agent systems: an artifact-based perspective”, *Autonomous Agents and Multi-Agent Systems*, vol. 23-2, Sep 2011, pp. 158–192.

[RVO07] Ricci, A.; Viroli, M.; Omicini, A. “Cartago: a framework for prototyping artifact-based environments in mas”. In: Proceedings of the International Conference on Environments for Multi-agent Systems, 2007, pp. 67–86.

[SD15] Singhal, V.; Dahiya, D. “Distributed task allocation in dynamic multi-agent system”. In: Proceedings of the International Conference on Computing, Communication Automation, 2015, pp. 643–648.

[SFOT05] Scerri, P.; Farinelli, A.; Okamoto, S.; Tambe, M. “Allocating tasks in extreme teams”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2005, pp. 727–734.

[SGFEB13] Such, J. M.; GarcíA-Fornes, A.; Espinosa, A.; Bellver, J. “Magentix2: a privacy-enhancing agent platform”, *Engineering Applications of Artificial Intelligence*, vol. 26-1, Jan 2013, pp. 96–109.

[SGSTS14] Segui-Gasco, P.; Shin, H. S.; Tsourdos, A.; Segui, V. J. “A combinatorial auction framework for decentralised task allocation”. In: Proceedings of the IEEE Globecom Workshops, 2014, pp. 1445–1450.

[SGSTS15] Segui-Gasco, P.; Shin, H.-S.; Tsourdos, A.; Segui, V. J. “Decentralised submodular multi-robot task allocation”. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2015, pp. 2829–2834.

[Sho93] Shoham, Y. “Agent-oriented programming”, *Artificial Intelligence*, vol. 60-1, Mar 1993, pp. 51–92.

[SKKR10] Stone, P.; Kaminka, G. A.; Kraus, S.; Rosenschein, J. S. “Ad hoc autonomous agent teams: collaboration without pre-coordination”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2010, pp. 1504–1509.

[SKV+12] Scerri, P.; Kannan, B.; Velagapudi, P.; Macarthur, K.; Stone, P.; Taylor, M.; Dolan, J.; Farinelli, A.; Chapman, A.; Dias, B.; Kantor, G. “Flood disaster mitigation: a real-world challenge problem for multi-agent unmanned surface vehicles”. In: Proceedings of the Autonomous Robots and Multirobot Systems Workshop, 2012, pp. 252–269.

[SP13] Settimi, A.; Pallottino, L. “A subgradient based algorithm for distributed task assignment for heterogeneous mobile robots”. In: Proceedings of the IEEE Conference on Decision and Control, 2013, pp. 3665–3670.

[SSFS12] Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N. “Conflict-based search for optimal multi-agent path finding”. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2012, pp. 563–569.

[SSPO15] Schneider, E.; Sklar, E. I.; Parsons, S.; Ozgelen, A. T. “Auction-based task allocation for multi-robot teams in dynamic environments”. In: Proceedings of the Conference on Towards Autonomous Robotic Systems, 2015, pp. 246–257.

[SUIM10] Stulp, F.; Utz, H.; Isik, M.; Mayer, G. “Implicit coordination with shared belief: a heterogeneous robot soccer team case study”, *Advanced Robotics*, vol. 24-7, Apr 2010, pp. 1017–1036.

[SWWA14] Smith, D.; Wetherall, J.; Woodhead, S.; Adekunle, A. “A cluster-based approach to consensus based distributed task allocation”. In: Proceedings of the Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2014, pp. 428–431.

[SZB16] Su, X.; Zhang, M.; Bai, Q. “Coordination for dynamic weighted task allocation in disaster environments with time, space and communication constraints”, *Journal of Parallel and Distributed Computing*, vol. 97-1, Nov 2016, pp. 47–56.

[TMS15] Turner, J.; Meng, Q.; Schaefer, G. “Increasing allocated tasks with a time minimization algorithm for a search and rescue scenario”. In: Proceedings of the IEEE International Conference on Robotics and Automation, 2015, pp. 3401–3407.

[TSM17] Talebpour, Z.; Savarè, S.; Martinoli, A. “Market-based coordination in dynamic environments based on the hoplites framework”. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2017, pp. 1105–1112.

[US13] Urakawa, K.; Sugawara, T. “Task allocation method combining reorganization of agent networks and resource estimation in unknown

environments”. In: Proceedings of the International Conference on Innovative Computing Technology, 2013, pp. 383–388.

[VDBSLM10] Van Den Berg, J.; Snoeyink, J.; Lin, M.; Manocha, D. “Centralized path planning for multiple robots: optimal decoupling into sequential plans”. In: Proceedings of the Conference on Robotics: science and systems, 2010, pp. 137–144.

[VSL+10] Vilenica, A.; Sudeikat, J.; Lamersdorf, W.; Renz, W.; Braubach, L.; Pokahr, A. “Coordination in multi-agent systems: a declarative approach using coordination spaces”. In: Proceedings of the European Meetings on Cybernetics and Systems Research, 2010, pp. 441–446.

[WdS08] Wang, Y.; de Silva, C. W. “A machine-learning approach to multi-robot coordination”, *Engineering Applications of Artificial Intelligence*, vol. 21-3, Apr 2008, pp. 470–484.

[WGL14] Wang, W.; Gao, X.; Liang, Z. “A dynamic strategy based on road-partitioning model in robocup rescue simulation”. In: Proceedings of the IEEE International Conference on Mechatronics and Automation, 2014, pp. 1789–1794.

[WHJ15] Wei, C.; Hindriks, K. V.; Jonker, C. M. “Auction-based dynamic task allocation for foraging with a cooperative robot team”. In: Proceedings of the European Conference on Multi-Agent Systems, 2015, pp. 159–174.

[WJ95] Wooldridge, M.; Jennings, N. R. “Intelligent agents: theory and practice”, *The Knowledge Engineering Review*, vol. 10-2, Jun 1995, pp. 115–152.

[WLL+16] Wang, Z.; Li, M.; Li, J.; Cao, J.; Wang, H. “A task allocation algorithm based on market mechanism for multiple robot systems”. In: Proceedings of the IEEE International Conference on Real-time Computing and Robotics, 2016, pp. 150–155.

[WMC15] Whitbrook, A.; Meng, Q.; Chung, P. W. H. “A novel distributed scheduling algorithm for time-critical multi-agent systems”. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2015, pp. 6451–6458.

[Woo09] Wooldridge, M. J. “An introduction to multiagent systems”. Wiley Publishing, 2009, 461p.

[Woo13] Wooldridge, M. “Intelligent agents”. MIT Press, 2013, chap. 1, pp. 3–50.

[XLL12] Xu, Y.; Li, X.; Liang, H. “Adjusting organization to improve coordination for large heterogeneous multi-agent teams”. In: Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, 2012, pp. 282–286.

[YJC13] Yan, Z.; Jouandeau, N.; Cherif, A. A. “A survey and analysis of multi-robot coordination”, *International Journal of Advanced Robotic Systems*, vol. 10-12, Dec 2013, pp. 18.

[YWN10] Yu, C.-H.; Werfel, J.; Nagpal, R. “Collective decision-making in multi-agent systems by implicit leadership”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1189–1196.

[YZF14] Yedidsion, H.; Zivan, R.; Farinelli, A. “Explorative max-sum for teams of mobile sensing agents”. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, 2014, pp. 549–556.

[ZCMZ13] Zhang, W.; Chen, X.; Ma, L.; Zhao, D. “Multi-agent pursuit with decision-making and formation control”. In: Proceedings of the Chinese Control Conference, 2013, pp. 7016–7022.

[ZL14] Zhang, K.; Li, X. “Human-robot team coordination that considers human fatigue”, *International Journal of Advanced Robotic Systems*, vol. 11-1, Jun 2014, pp. 9.

[Zlo06] Zlot, R. M. “An auction-based approach to complex task allocation for multirobot teams”, Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, 2006, 187p.

[ZLZ+16] Zhu, H.; Liu, D.; Zhang, S.; Zhu, Y.; Teng, L.; Teng, S. “Solving the many to many assignment problem by improving the kuhn-munkres algorithm with backtracking”, *Theoretical Computer Science*, vol. 618-C, Mar 2016, pp. 30–41.

[ZMC16] Zhao, W.; Meng, Q.; Chung, P. W. H. “A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario”, *IEEE Transactions on Cybernetics*, vol. 46-4, Apr 2016, pp. 902–915.

[ZML18] Zou, S.; Ma, Z.; Liu, X. “Auction-based mechanism for dynamic and efficient resource allocation”, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48-1, Jan 2018, pp. 34–49.

[ZMY+14] Zhou, J.; Mu, D.; Yang, F.; Dai, G.; Shell, D. A. “A distributed approach to load balance for multi-robot task allocation”. In: Proceedings of the IEEE International Conference on Mechatronics and Automation, 2014, pp. 612–617.


Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br