Received: 2 March 2017 | Revised: 6 May 2018 | Accepted: 30 August 2018

DOI: 10.1111/coin.12194

ORIGINAL ARTICLE

WILEY Computational Intelligence

Allocating structured tasks in heterogeneous agent teams

Tulio L. Basegio | Rafael H. Bordini

School of Technology, PUCRS, Porto Alegre, Brazil

Correspondence
Tulio L. Basegio and Rafael H. Bordini, School of Technology, PUCRS, 90619-900 Porto Alegre-RS, Brazil.
Emails: tulio.basegio@acad.pucrs.br; rafael.bordini@pucrs.br

Funding information
CNPq; CAPES; IFRS

Abstract

Task allocation is an important aspect of multiagent coordination. However, there are many challenges in developing appropriate strategies for multiagent teams so that they operate efficiently. Real-world scenarios such as flooding disasters usually require the use of heterogeneous robots and the execution of tasks with different structures and complexities. In this paper, we propose a decentralized 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 to evaluate the proposed mechanism. The results show that the proposed mechanism appears to scale well and provides near-optimal allocations.

KEYWORDS

multiagent systems, multirobot systems, task allocation

1 | INTRODUCTION

One of the challenges in developing multirobot systems is the design of coordination strategies in such a way that robots perform their operations efficiently.[1] Without such strategies, the use of multirobot systems in complex scenarios such as rescue operations after natural disaster becomes limited or even unfeasible. More generally, the same applies to multiagent systems.

Coordination is related to the social skills of agents, where agents communicate not only to share data, but also communicating their beliefs, goals, and plans to other agents.[2] With coordination, agents can achieve joint objectives and plans that otherwise might not be possible; it ensures that agents perform their tasks in coherent and efficient ways, synchronizing their actions and interactions with other agents.[3,4]

An important aspect considered in coordination problems is task allocation.[1,5-10] There are several features that should be considered by a mechanism for allocating tasks to multiple agents

in real-world scenarios such as considering their heterogeneity (different physical and computational capacities, eg, different sensors and types of mobility), the impact of individual variability to assign specific roles to individual robots, and the definition and allocation of different types of tasks.

Although many current real-world scenarios, such as disaster rescue, 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 generally focus on one type of constraint only, and the ones that deal with more features generally 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 decentralized 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 multirobot systems. Although in this paper we deal with static task allocation problems, in ongoing work, we address a dynamic version of such problems, which is important for our target application. The proposed mechanism was initially inspired from the work of Luo et al,[8] 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 in this paper first appeared in an earlier work of the authors.[11] In this paper, we provide further details about the process and algorithms, we provide new results from the comparison with the optimal solution and we introduced a comparison with other decentralized approaches. We also changed the way we determine the end of the allocation process; as in our earlier work,[11] we used the same approach as the work of Luo et al.[8]

In this paper, we use a flooding disaster scenario to exemplify our approach. Flooding disasters are typically very dynamic and have complex tasks to be executed.[12] In fact, it was the typical tasks in flooding rescue that inspired us to work on a task allocation mechanism that could address the various types of tasks we approach in this paper. During a rescue phase in a flooding disaster, teams are called into action to work in tasks such as locating and rescuing victims.[13] Such teams are normally organized by a hierarchy model,[14] with individuals playing different roles during a mission. The execution of tasks during the rescue stage poses a number of risks to the teams. Using robots in a coordinated way to help the team may minimize 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.

The remainder of this paper is organized as follows. Section 2 provides the background related to task allocation describing relevant concepts to our work as well as formalize our task allocation problem. Section 3 presents an overview about our task allocation process while Section 4 presents the proposed task allocation mechanism. In Section 5, we evaluate the proposed approach by comparing it with the optimal solution and other two decentralized approaches. Section 6 describes related works. Finally, in Section 7, we conclude this paper.

2 | MULTIAGENT TASK ALLOCATION

Task allocation among multiple robots (and more generally among multiple agents) consists of identifying which robots should perform which tasks to achieve cooperatively as many global goals as possible and in the best possible way. Previous to the definition of our multiagent 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.

2.1 | 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 indicate, eg, that some subset of the tasks must be carried out concurrently,[15] or by a single robot, etc.

2.2 | Utility

In this paper, 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 (eg, the quality with which a particular robot is likely to accomplish that task, how quickly that is likely to be done, and so forth).[16] 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. To calculate the utility value of a task, relevant aspects of the robot, the environment, and the actual task requirements should be taken into consideration.[16] In this work, we assume that only the robot itself is able to accurately determine its utility for a given task; there is no way to compute in a centralized way the utility functions for all the robots, hence the importance of a decentralized approach to task allocation.

2.3 | 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 paper, we use the following types of tasks, as defined in the work of Zlot[17] (see Figure 1).

Atomic task: A task is atomic if it cannot be decomposed into subtasks. It can only be carried out by a single robot. 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, ie, the decomposed parts must all be carried out by the same robot. We refer to this as a **DS task** throughout this paper.

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 affected by constraints and then we separate the compound task type into two types. 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). When there is no such constraint, 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**.

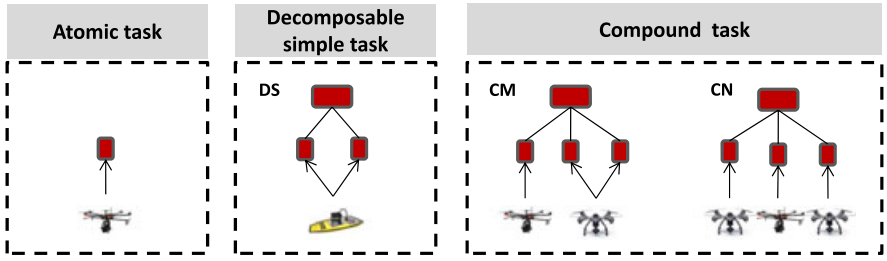

FIGURE 1 Types of tasks considered in our approach [Color figure can be viewed at wileyonlinelibrary.com]

Compound tasks have a list of subtasks that can be, in turn, atomic or compound; this way, it is possible to create a complex hierarchy of tasks, which increases the applicability of this approach to different scenarios. Similar structures are also possible for decomposable simple tasks.

Using a flooding scenario as follows, we exemplify how these types of tasks can be used.

Take as example a task where robots need to act as wireless repeater nodes in five specific locations in a row. It means the task needs to be allocated to exactly five different robots, as a robot cannot be in more than one place at a time. In this case, the multiagent organization will create a CN task with five subtasks to have exactly five robots allocated to that task.

For mapping an area, for instance, the organization may want that at most three robots are responsible for the mapping. In this case, the organization creates a CM task with three subtasks, thus allowing for the task to be allocated to one, two, or at most, three robots.

In a flooding disaster, areas that are not populated offer low risk to people's lives. In this case, the organization could determine that the collection of water samples for analysis in three different locations of that area should be performed by a single robot. Therefore, the organization could create a DS task with three subtasks, one for each collection location.

2.4 | Multirobot task allocation

The most basic task allocation problem addressed in the robotics literature can be stated as follows: *Given a set of robots R and a set of tasks T with each robot obtaining some utility value for the execution of each task, each task must be allocated to exactly one robot and each robot must be assigned to at most one task; the objective is to find an assignment of tasks to robots so that the overall utility of all the robots is maximized*.[8]

We here 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 to the amount of energy (fuel) available to a robot, thus limiting the number of tasks it can be assigned. In addition, 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. Moreover, because the approach can be used more generally than only in multirobot systems, we often use the term agent instead of robot.

2.5 | Problem statement

In this section, we formally state our MATA optimization 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 jth task has. Furthermore, we use the binary variable p_{jk} indicating whether st_k belongs to t_j to formalize the constraints aforementioned as follows:

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

$$\sum_{k=1}^{n_{st}} p_{jk} >= 1, \quad \forall j = 1, \ldots, n_t. \tag{2}$$

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 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

$$\sum_{i=1}^{n_r} f_{ik} \leqslant 1, \quad \forall k = 1, \ldots, n_{st}. \tag{3}$$

$$\sum_{k=1}^{n_{st}} f_{ik} \leqslant L_i, \quad \forall i = 1, \ldots, n_r. \tag{4}$$

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

$$\sum_{q=1}^{n_q} w_{jq} = 1, \quad \forall j = 1, \ldots, n_t. \tag{5}$$

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 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, ie, 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, ie, 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 (6) and (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

$$\sum_{k=1}^{n_{st}} f_{ik}.p_{jk} \geqslant \min_j, \quad \forall j \text{ s.t. } t_j \in T;\ i = 1, \ldots, n_r. \tag{6}$$

$$\sum_{k=1}^{n_{st}} f_{ik}.p_{jk} \leqslant \max_j, \quad \forall j \text{ s.t. } t_j \in T;\ i = 1, \ldots, n_r. \tag{7}$$

In addition, there are also constraints related to the roles the robots may play in the organization 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 to execute it. We use the following binary variables to formalize the constraints aforementioned: 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

$$\sum_{x=1}^{n_c} g_{yx} \leqslant n_c, \quad \forall y = 1, \ldots, n_e. \tag{8}$$

$$\sum_{y=1}^{n_e} h_{ky} \leqslant n_e, \quad \forall k = 1, \ldots, n_{st}. \tag{9}$$

$$\sum_{x=1}^{n_c} v_{ix} \leqslant n_c, \quad \forall i = 1, \ldots, n_r. \tag{10}$$

$$\sum_{y=1}^{n_e} z_{iy} \leqslant n_e, \quad \forall i = 1, \ldots, n_r. \tag{11}$$

$$\sum_{x=1}^{n_c} g_{yx}.v_{ix}.z_{iy} = \sum_{x=1}^{n_c} g_{yx}.z_{iy}, \quad \forall i = 1, \ldots, n_r;\ y = 1, \ldots, n_e. \tag{12}$$

$$\sum_{y=1}^{n_e} h_{ky}.z_{iy}.f_{ik} = \sum_{y=1}^{n_e} h_{ky}.f_{ik}, \quad \forall i = 1, \ldots, n_r;\ k = 1, \ldots, n_{st}. \tag{13}$$

Finally, the objective of our MATA problem is to find an allocation that maximizes the sum of utilities of the agents while satisfying all the above constraints. The idea is that the process of maximizing the sum of individual utilities simultaneously improves the global utility.[18] The objective function of our optimization problem can be stated as follows.

Objective:

$$\max_{\{f_{ik}\}} \sum_{i=1}^{n_r} \sum_{k=1}^{n_{st}} u_{ik}.f_{ik}. \tag{14}$$

3 | OVERVIEW OF THE ALLOCATION PROCESS

We propose a decentralized mechanism for 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. In this section, we first present a general view of the allocation process, focusing on the main elements of the proposed mechanism.

The main elements considered in the proposed mechanism are presented in Figure 2. Initially, we consider the existence of an organization 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 organization are published on a blackboard that can be viewed by all the agents available for the mission. Finally, the environment is the place where agents carry out the tasks. Blackboard is a widely known architecture (see the works

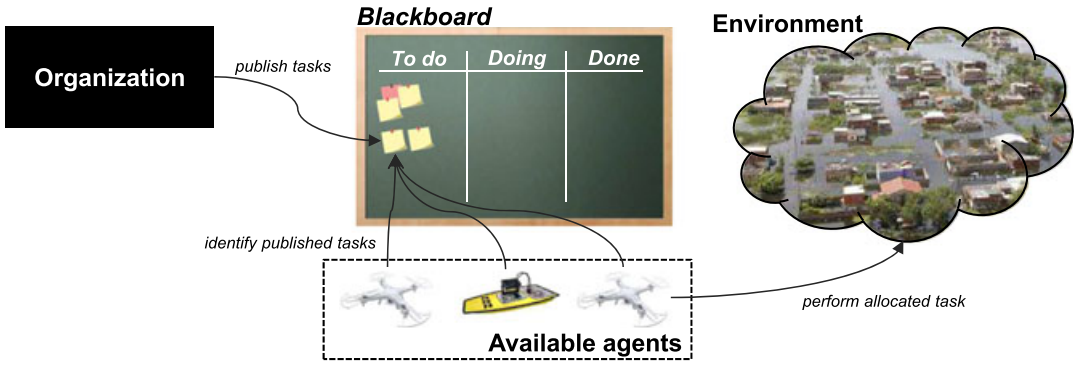

FIGURE 2 Overview of the task allocation problem [Color figure can be viewed at wileyonlinelibrary.com]

of Hayes-Roth[19] and Rudenko and Borisov[20]), which works as a global accessible space and can be used, eg, for sharing information among agents.[20]

Regarding the process itself, it is initially considered that an organization 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 needed, the organization publishes a set of tasks on the blackboard to which all agents have access. By identifying the new set of tasks available, the agents begin the allocation process based on the mechanism we describe in this paper. Information about the roles required by the organization is also published on the blackboard.

Simply put that each agent initially identifies the roles available in the organization, and based on its capabilities, it checks which roles it can possibly play in the organization. The agent then identifies on the blackboard tasks it can carry out based on the roles it can play. The agents then exchange bids for the tasks they want to be allocated to (tasks with the highest utilities for themselves). When an agent receives a bid that improves on its bid for a task it wanted allocated to itself, the agent withdraws that task from the list of its allocated tasks and checks which task it will bid for in order to replace the task 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 tasks start to carry them out.

Note that, at the end of the allocation process, there might be agents without any allocated task as well as tasks 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 organization announces more tasks than the total limit (capacity) of all the agents together, clearly some tasks will not be allocated. On the other hand, if the total limit of the agents together is higher than the number of tasks to be allocated, there may be agents without any task allocation. In addition, the available agents may not be able to play the roles required to carry out some tasks, so those tasks will not be allocated to any agent. If an agent is not able to play any of the roles required by the tasks, it will not be able to allocate any of them. Furthermore, other constraints related to the number of robots required for subtasks of certain types of tasks may also lead to incomplete allocations.

Next, we provide an example for our allocation process using a flooding scenario. According to the work of Murphy,[21] 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 this task, a robot needs to have flight capability and a camera to obtain the images. Note that it would be possible to have the same task for a robot with sailing capability to get images from a different perspective. Another task in flood disasters is the collection of water samples for analysis (ie, to check the level of water contamination[12]). To perform such a 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 two specific types of tasks. Note that, for the sake of simplicity, the information about tasks, roles, capabilities, and robots described are deliberately kept at rather high level, not even including subtasks.

Regarding the process itself, consider that the organization needs to work on a flooding disaster by performing the aforementioned tasks, ie, mapping areas and collecting water samples for analysis. The organization has three robots available to carry out a mission, ie, one unmanned surface vehicle (USV) and two unmanned aerial vehicles (UAVs), which we will call, respectively, USV1, UAV1, and UAV2. The robots start executing without having any assigned tasks. USV1 has water navigation capability* and resources to collect water samples, while UAV1 and UAV2 have flying capabilities and cameras to take pictures. The following predicates represent the information each robot has about itself:

$$\begin{aligned} &\text{USV1} : \text{capabilities}([\text{sail}, \text{sampler}]) \\ &\text{UAV1} : \text{capabilities}([\text{fly}, \text{camera}]) \\ &\text{UAV2} : \text{capabilities}([\text{fly}, \text{camera}]). \end{aligned}$$

In the organization, there are two possible roles to be played, ie, mapper and collector. In order to play the mapper role, a robot must have the capability to fly and must have a camera to take pictures. For the collector role, robots must have the capability to navigate on water and resources to collect water samples. The organization publishes information about the roles on the blackboard to which all robots have access. The following predicates represent this information:

$$\begin{aligned} &\text{role}(\text{mapper}, [\text{fly}, \text{camera}]) \\ &\text{role}(\text{collector}, [\text{sail}, \text{sampler}]). \end{aligned}$$

Considering the flooding scenario, the organization publishes the following tasks on the blackboard. The following predicates are composed of (and in this particular order): the task identifier, the task name, the region where the task is to be performed, and the role required for a robot to perform that task

$$\begin{aligned} &\text{task}(t_1, \text{collectWater}, \text{regionA}, \text{collector}) \\ &\text{task}(t_2, \text{takeImage}, \text{regionA}, \text{mapper}) \\ &\text{task}(t_3, \text{takeImage}, \text{regionB}, \text{mapper}). \end{aligned}$$

By perceiving the new set of tasks available, the robots begin the allocation process based on the proposed mechanism. First, each robot will identify which roles it can play in the organization. USV1 identifies it can play the collector role, whereas UAV1 and UAV2 identify they can only play the mapper role. Each robot is now able to identify the tasks it can bid for based on the roles it

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

can play. USV1 realizes it can bid only for task t_1, whereas UAV1 and UAV2 realize they can bid for tasks t_2 and t_3.

The robots start biding for the tasks they prefer until all the robots agree on the allocated tasks and the allocation process finishes after that. The robots then can start the execution of the allocated tasks, following their own plans and using the resources they have.

Note that, at any time, the organization may need to add new tasks or even new tasks might be discovered by the robots while executing the current tasks. In addition, during the execution of tasks in dynamic environments such as floodings, the robots may fail at any time. When the others robots realize that a robot failed, they start the reallocation of the tasks assigned to the failed robot. As we mentioned previously, here, we deal with static task allocation, but in ongoing work, we are extending our approach to deal with such dynamic allocation aspects.

4 | DECENTRALIZED TASK ALLOCATION

Our work aims at providing a decentralized 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 (eg, the possibility of sailing or flying) or even to the resources available to the agent (ie, the robot's payload such as cameras, sensors, etc). An agent may play one or more roles. The roles are defined by the organization 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 organization is also responsible for stating the tasks that are required in a given mission.

Remember that we are working with the type of tasks described in Section 2, ie, DS tasks, where all the subtaks need to be allocated to the same agent; CN tasks, where each subtask needs to be allocated to a different agent; and CM tasks, where the subtasks can be allocated to different agents (no constraints). The proposed mechanism allow us to work with those different type of tasks through the definition of the minimum and 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, eg, 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. Furthermore, our approach assumes reliable communication.

4.1 | Algorithms for the task allocation process

The proposed task allocation mechanism is based on algorithms that are executed by each agent in the organization, characterizing a decentralized solution. A general view of algorithms that constitute the core of the proposed mechanism is presented as follows.

Starting the allocation process (Algorithm 1): The initial algorithm is Algorithm 1, which receives as input two parameters, ie, 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 organization (note that both can change at runtime). In a new allocation process, the first step for an agent is to select only the tasks that are compatible with the roles that it can play (line 5 shows the call to the function in Algorithm 2, which takes care of that). Knowing the tasks that the agent can perform, it calls

Algorithm 3 (line 6), which updates for each task the minimum and maximum number of subtasks that a robot must take. Finally, it calls Algorithm 4 (line 7), which starts the bidding process for the tasks that can be allocated to that agent.

Algorithm 1
`startAllocation`(*blackboardTasks*, *organizationRoles*)

1: *allocatedSubtasks* = ∅;
2: *candidates* = ∅;
3: *taskList* ← *blackboardTasks*;
4: *roleList* ← *organizationRoles*;
5: *possibleTasks* ← *getPossibleTasks*(*taskList*, *roleList*);
6: *possibleTasks* ← *getMinMaxTaskType*(*possibleTasks*)
7: *taskAllocation*(*possibleTasks*, *candidates*, *allocatedSubtasks*);

Identifying the possible tasks (Algorithm 2): Algorithm 2 receives as input the list of all blackboard tasks that need to be carried out as well as the description of all the roles currently defined within the organization. Initially, the algorithm identifies the possible roles that the agent may play considering its capabilities and the capabilities required for each role (lines 1 to 12). Then, given the roles the agent can play, the algorithm identifies which tasks can be allocated to the agent (lines 13 to 17).

Algorithm 2
`getPossibleTasks`(*blackboardTaskList*, *rolesList*)

Let *agentCapabilities* be the list of agent's capabilities;
2: **for all** *role* r_k **in** *rolesList* **do**
validRole ← *true*
4: **for all** *capabilityRequired* **in** r_k **do**
if *capabilityRequired* **not in** *agentCapabilities* **then**
6: *validRole* ← *false*
end if
8: **end for**
if *validRole* = *true* **then**
10: *agentRoles.add*(r_k)
end if
12: **end for**
for all *task* t_j **in** *blackboardTaskList* **do**
14: **if** t_j.*role* **in** *agentRoles* **then**
possibleTasks.add(t_j)
16: **end if**
end for
18: **return** *possibleTasks*

Update min/max subtasks by task type (Algorithm 3): Algorithm 3 receives as input the list of all tasks the agent can execute. It goes through each task and updates the minimum and maximum number of subtasks that a robot must take from that task (lines 2 to 16). 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.

Algorithm 3
`getMinMaxTaskType(`*possibleTasks*`)`

```
    for all task t_j in possibleTasks do
3:      N_j ← t_j.subTasks.count
        minType ← taskTypes.getMin(t_j.type);
        maxType ← taskTypes.getMax(t_j.type);
6:      if minType = N then
            t_j.minSubTask ← N_j;
        else
9:          t_j.minSubTask ← minType;
        end if
        if maxType = N then
12:         t_j.maxSubTask ← N_j;
        else
            t_j.maxSubTask ← maxType;
15:     end if
    end for
    return possibleTasks
```

Performing the task allocation (Algorithm 4): Algorithm 4 receives as input the list of tasks that can be allocated to agent r_i (which is running the algorithm). Initially, the algorithm checks if the number of tasks the agent allocated to itself (na_i) so far is lower than its capacity (the task limit l_i), and if so, it begins the analysis of all possible tasks to identify the tasks that can still be allocated (lines 5 to 16). The analysis goes through each task as described as follows. We use l'_i to refer to the difference between l_i and na_i. For each task t_j in the list of tasks the agent is able to execute, the algorithm identifies the number of subtasks the agent can select as candidates for allocation (line 12). In order to get this value, the algorithm first checks if the agent has capacity to select the minimum number of subtasks required by task t_j (line 10) and if there are still subtasks of task t_j that are not in the list of allocated subtasks (line 11). Then, it selects as candidates the *nToAlloc* best subtasks from task t_j (line 13). The choice of candidates is carried out based on the utility of each subtask of that task. From the list of subtasks selected as candidate, the algorithm selects the best subtasks for allocation, considering the task limit for the agent (line 17). For the subtasks selected, the algorithm calculates the value for the bids to be sent to the other agents. This calculation was inspired by the bid calculation formula introduced in the work of Luo et al.[8] The bid for a subtask is its utility value minus the amount that would be lost if the next best subtask were taken instead. The subtasks in the *bestCandidate* list are not considered when identifying the next best subtask. Lines 19 to 22 refer to the calculation of the bids and their broadcasting. As noted earlier, our approach requires reliable communication.

Processing the received bids (Algorithm 5): Each agent processes the bids received from other agents by executing Algorithm 5. When an agent receives a bid from another agent, which is greater than the bid the agent itself provided when allocating the subtask to itself, the agent has to remove that subtask from its allocated subtasks, and then it tries to allocate others subtasks by executing Algorithm 4.

Algorithm 4
`taskAllocation`(*possibleTasks*, *candidates*, *allocatedSubtasks*)

```
    Let l be the agent's max. number of allowed concurrent tasks;
    na ← allocatedSubtasks.size;
    l' ← l − na;
 4: if l' > 0 then
        for all task t_j in possibleTasks do
            minT ← t_j.minSubTask;
            maxT ← t_j.maxSubTask;
 8:         naT ← allocatedSubtasks.countSubtasksFrom(t_j);
            notAllocT ← maxT − naT;
            if l' >= minT then
                if maxT > naT then
12:                 nToAlloc ← min[l', maxT, notAllocT];
                    candidates.add(nBestSubtasks(t_j.subTasks, nToAlloc));
                end if
            end if
16:     end for
        bestCandidates ← getBest(candidates, l');
        allocatedSubtasks.add(bestCandidates);
        for all subtask in bestCandidates do
20:         bid = subtask.utility − (subtaskNext.utility − subtaskNext.winBid) + 1;
        end for
        communicate new bid values to all other agents;
    end if
```

Algorithm 5
`processBids`(*receivedBids*)

```
    if receivedBids ≠ ∅ then
        for all bid in receivedBids do
            if bid.subtask in allocatedSubtasks then
 5:             if bid.value > allocatedSubtasks.subtask.bidValue then
                    allocatedSubtasks.remove(subtask)
                end if
            end if
        end for
10: end if
    Let la be the agent's maximum number of allowed concurrent tasks;
    na ← allocatedSubtasks.size;
    if (la − na) > 0 then
        taskAllocation(possibleTasks, candidates, allocatedSubtasks);
15: end if
```

Algorithms 4 and 5 are repeated until the agents agree on the allocation, ie, until the self-allocated subtasks do not undergo any further modifications.

TABLE 1 Example of candidate task list

Task	DS1			DS2		CN1		CM1	
Subtask (object)	st1	st2	st3	st4	st5	st6	st7	st8	st9
#subtasks (weight)	3			2		1	1	1	1
Utility (value)	12			8		5	1	4	3

4.2 | Selecting the best tasks from the candidate list using a knapsack algorithm

In Algorithm 4, we call the *getBest* function (line 17) for selecting the best tasks for allocation to an agent from its candidates list. Our *getBest* 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 maximize the total value of the items in the knapsack while respecting the maximum weight it can take. Analogous to the knapsack problem, the limit of tasks that an agent can be allocated corresponds to the weight limit of the knapsack; the number of positions that a task will occupy in the agent's task limit, ie, the number of subtasks that need to be taken together, corresponds to the weight of an item; and the utility value of a subtask corresponds to the value of an item (see Table 1).

In order to explain the use of the knapsack algorithm in this paper, let us consider an agent that has a limit to allocate up to five subtasks. Consider also the task samples available in Table 1. Recall that, for a decomposable simple task (such as DS1 and DS2 in Table 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 using the number of subtasks as the total of positions occupied by the task in the agent's limit, ie, its weight. Thus, task DS1 is considered as a task that has weight 3 (it will occupy three places in the agent's task limit) and has a total value of 12 (the sum of the utilities of the individual subtasks).

The subtasks from CN and CM tasks can be independently allocated. Thus, each subtask will occupy only one position in the agent's task limit and its utility value will also be considered individually.

At each iteration of Algorithm 4, the agent runs the knapsack algorithm (called on line 26) 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 maximized 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 4 ensure that only a typically small selection of tasks take part in this step of the overall allocation process, and agents typically have small task limits. In other words, even if a large number of tasks are available, the process will filter the tasks that will be sent to the *getBest* function (which calls our knapsack algorithm) and only a relatively small number of tasks will be considered for a small task limit.

4.3 | Determining the end of the allocation process

In distributed 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 the work of Luo et al,[8] 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 ie, 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, ie, 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 are 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. Choi et al[22] also used a similar approach.

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

- 0 – Set this value when agent a_j won all the tasks for which it has bid. It means that the agent does not need to send further bids for now;
- 1 – 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:

  a. when a_j does not win all of the tasks for which it has placed bids, ie, at least one of the tasks has already received a higher bid from another agent. For example, if agent a_i is processing a bid list with bids for three tasks and it realizes that the agent only won two of them, it means that a_j will need to send further bids.
  b. when a_j is outbid by another agent, ie, when it loses one of its allocated tasks. That can be checked when agent a_i is processing bids from other agents or when it allocates a task to itself that outbids a_j.

When an agent is outbid, it will always try to select another task to bid for. However, when that agent is not able to select another task, 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 0 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 task.

Regardless of the value set in the *AgentBidStatus* list, each task a bidding agent wins will be associated to it in *TaskWinner* list. In addition, when agent a_i allocates a task to itself, it will be associated to that task in the *TaskWinner* list.

In order to control the end of the whole allocation process, we do as follows. When there are no more bids to be processed, an agent will check 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 0, which means all agents sent their bids or *done* messages, and tasks were allocated in accordance with the bids.

In summary, at the beginning of the allocation process, each agent will send a bid list for all tasks that it wishes to allocate. Next, only the agents that were outbid will send bids for other tasks again, but only for the newly selected tasks. If the agent is not able to select another task, 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 task and the status of each agent related to the bids they sent. Furthermore, as most other auction-based approaches, our approach assumes reliable communication among agents.

4.4 | 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, ie, 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. First, we need to identify the tasks that are not completely allocated, ie, the tasks in which at least one of their subtasks was not allocated to any agent.

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. 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. Therefore, we run our allocation process again, but one task at a time, ie, we run the process to fully allocate one task, and 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 the 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.

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 global ordering is determined by the sum of points from all the voters.[23]

Simply put, in our use of Borda count, the agents submit their order of preference to all other agents. This preference lists can have different tasks and different lengths for each agent, because some tasks may not be possible to be executed by some agents due to the roles required by the task. All tasks not included in the list by an agent are assumed to be equally least preferred by that agent.

Based on the voting from all agents, each agent individually calculate the global ordering for the allocation process. The tasks will be allocated in that ordering. The agents know which task should be first processed and then start providing bids on the subtasks of that task. After receiving the bids, the winner for each subtask is known. 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 ignored. 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.

5 | EVALUATION

This section compares the performance of the proposed mechanism with the optimal solution. The GNU linear programming kit[24] was used to obtain (centralized) optimal solutions for comparison with our results. Our mechanism was implemented in belief-desire-intention agents[25,26] using a framework for multiagent systems development called JaCaMo.[27] We also compare the performance of our mechanism with two other decentralized solutions, ie, iterative consensus-based auction algorithm (ICBAA)[22] and sequential single-item auction algorithm (SSIA).[28] Finally, 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, ie, a task allocation algorithm (we call it TAA) used in the work of Ghamry et al[29] and a role-based task allocation (we call it RBTA) available in the work of Gunn and Anderson.[30]

By performance, we mean the overall utility obtained by all the agents to take on all the subtasks that they can. The coefficient of variation (standard deviation divided by the mean) was used as a measure of dispersion (ie, the amount of variability relative to the mean). The lower the coefficient of variation, the more homogeneous the data, ie, the dispersion in the data is smaller. The number of bid messages sent is also measured to assess the impact of the different solutions on the network traffic.

The simulations were run by varying a single parameter at each setting (Table 2). In all simulations, the subtasks of different types of task (CN, CM, and DS) were uniformly distributed. Moreover, for each agent, we randomly selected the utility values for each subtask from the utility

TABLE 2 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

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 100 iterations each. There were also simulations where we considered agents with capabilities to play any role, 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. The average results for these simulations were calculated from 20 iterations for each variation.

5.1 | 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 2. 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 ran simulations considering agents with capabilities to play any role, thus able to carry out any task. Figure 3A shows that the performance of the proposed solution improves and is closer to the optimal solution as we increase the number of agents. In addition, 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 with the results of the proposed solution (Figure 3B). However, this difference becomes smaller for larger agent teams. Regarding the number of bid messages, Figure 3C shows that the average number of bid messages placed by an agent remains stable for larger number of agents while the average number of bid messages by subtask increases since there are more agents bidding for the same subtasks.

Then, we ran simulations by randomly assigning from one to four 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 4 shows the results for these simulations with similar performance obtained in the previous simulations.

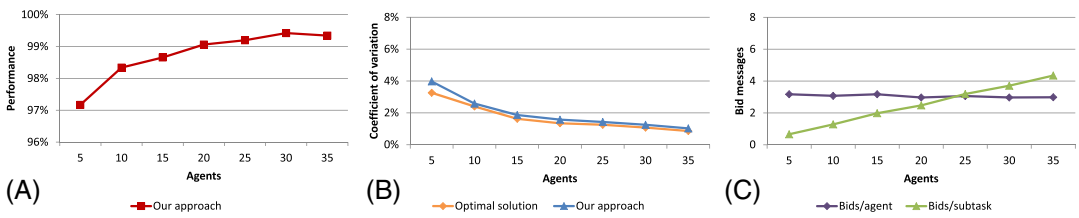

FIGURE 3 Performance results varying the number of agents [Color figure can be viewed at wileyonlinelibrary.com]

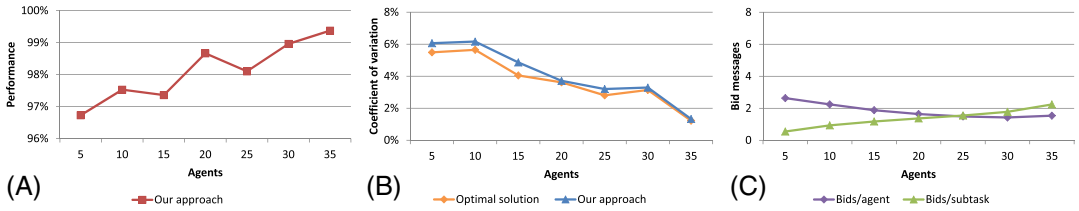

FIGURE 4 Performance results varying the number of agents and agents capabilities [Color figure can be viewed at wileyonlinelibrary.com]

5.2 | 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 2. 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 5A 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 5B). 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 5C).

Then, for the next simulations we randomly assign from 1 to 4 capabilities to each agent. Figure 6 shows the results with similar performance achieved in the first part of the simulations.

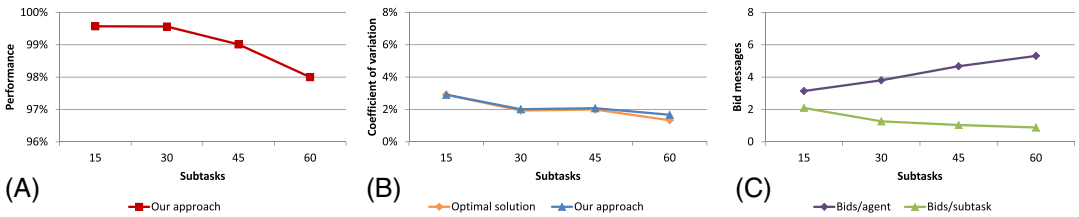

FIGURE 5 Performance results varying the number of subtasks [Color figure can be viewed at wileyonlinelibrary.com]

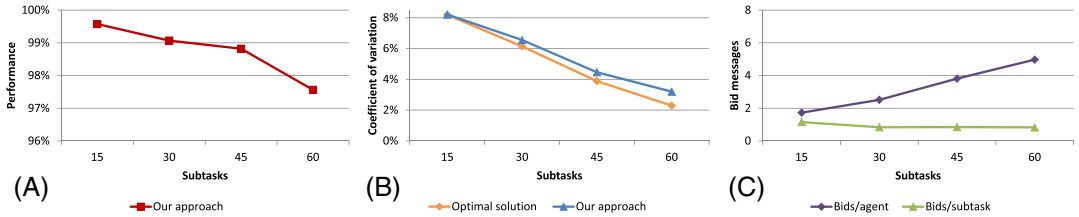

FIGURE 6 Performance results varying the number of subtasks and agents capabilities [Color figure can be viewed at wileyonlinelibrary.com]

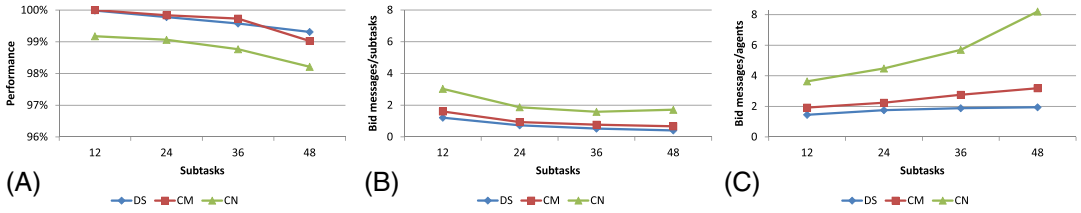

FIGURE 7 Performance results varying the number of subtasks for each type of task [Color figure can be viewed at wileyonlinelibrary.com]

5.2.1 | Separate simulations for each type of task

The aforementioned results show that the performance decreases somewhat with the increase in the number of subtasks. However, since CN, CM, and DS task types 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 separate simulations for each type of task.

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 100 simulations for each different number of subtasks.

Figure 7A 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 decrease 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 7B and 7C 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 7, 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.

5.3 | 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 2. 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 8A 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, ie, the limit of subtasks an agent can take does not impact the number of exchanged bid messages.

5.4 | 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.

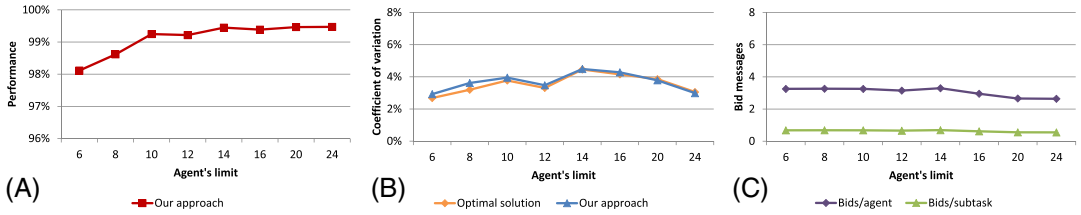

FIGURE 8 Performance results varying the limit of subtasks the agents can take [Color figure can be viewed at wileyonlinelibrary.com]

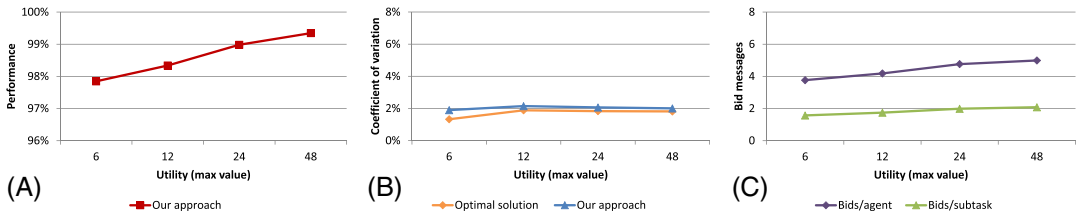

FIGURE 9 Performance results varying the utility range [Color figure can be viewed at wileyonlinelibrary.com]

Figure 9A 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, ie, the utility ranges have a small impact in the number of exchanged bid messages.

5.5 | 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 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 three and the number of subtasks to be allocated were 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 3 Simulations with the number of tasks greater than the total capacity of agents

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

Abbreviations: EA, entirely allocated; NA, completely unallocated; PA, partially allocated.

Table 3 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 (ie, all subtasks were allocated). The average results were calculated from 20 iterations for each variation.

5.6 | Comparison with SSIA and ICBAA

Here, we compare the performance of our mechanism with ICBAA[22] and SSIA.[28] 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 50 repetitions each.

5.6.1 | 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 4 shows the results of the simulations for the three algorithms, where the Utility represents the average utility of 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 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 less bid messages in all configurations.

In order to statistically analyze 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 analyzed the utility values obtained and also 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 4 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%

Abbreviations: ICBAA, iterative consensus-based auction algorithm; SSIA, single-item auction algorithm.

TABLE 5 Average number of bids and time to complete the allocation

	ICBAA		SSIA		Our approach	
Subtasks	Bid messages	Time, second	Bid messages	Time, second	Bid messages	Time, second
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

Abbreviations: ICBAA, iterative consensus-based auction algorithm; SSIA, single-item auction algorithm.

TABLE 6 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%

Abbreviations: SSIA, single-item auction algorithm.

TABLE 7 Average number of bids and time to complete allocation

	SSIA		Our approach	
Agents	Bid messages	Time, second	Bid messages	Time, second
5	73	9	36	6
10	90	10	53	8
25	126	14	100	17

Abbreviations: SSIA, single-item auction algorithm.

5.6.2 | 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), which could not be acceptable for real-world scenarios, here, we decide to focus only in the comparison with SSIA.

Table 6 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 7 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 less bid messages in all configurations.

We also statistically analyze 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.

5.7 | Comparison with TAA and RBTA

In this section, we compare the performance of our mechanism with the task assignment approaches used in two frameworks, ie, a task allocation algorithm (we call it TAA) used in the work of Gharmy et al[29] and a role-based task allocation (we call it RBTA) introduced in the work of Gunn and Anderson.[30] 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 the following simulation parameters were averaged over 50 repetitions each.

TABLE 8 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%

Abbreviations: TAA, task allocation algorithm.

TABLE 9 Average number of bids and time to complete allocation

	TAA		Our approach	
Agents/Tasks	Bid messages	Time, second	Bid messages	Time, second
5	15	6	8	2
10	55	7	19	2
15	120	8	30	3

Abbreviations: TAA, task allocation algorithm.

5.7.1 | 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, this simulations were run with 5, 10, and 15 agents and tasks. Table 8 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 some dispersion of the results, especially for the simulation with five agents/tasks.

Table 9 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 less bid messages in all configurations.

We performed paired t-tests to statistically analyze the results. We analyzed 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.

5.7.2 | Comparison with RBTA

In the RBTA approach, each agent can allocate more than one task (restricted to a certain limit). In addition, the agents may have different capabilities that limit the tasks they can allocate. The RBTA approach was defined consider the suitability expression used in the approach to exactly the same as the minimum task requirements. 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 10 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. Table 11 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 10 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%

Abbreviations: RBTA, role-based task allocation.

TABLE 11 Average number of bids and time to complete allocation

	RBTA		Our approach	
Tasks	Bid messages	Time, second	Bid messages	Time, second
40	284	11	60	5
50	409	12	94	6
60	531	14	158	12

Abbreviations: RBTA, role-based task allocation.

TABLE 12 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%

Abbreviations: RBTA, role-based task allocation.

TABLE 13 Average number of bids and time to complete allocation

	RBTA		Our approach	
Agents	Bid messages	Time, second	Bid messages	Time, second
10	531	14	158	10
12	537	13	126	10
15	542	14	110	9
18	608	14	113	11

Abbreviations: RBTA, role-based task allocation.

Then, we run the simulations by varying the number of agents (10, 12, 15, and 18 agents) to allocate 60 tasks. Table 12 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 13 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 analyze the results of our solution and the results from RBTA (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, except for the simulation with 18 agents (see Table 12), which had no statistical difference between the results.

5.8 | Discussion

We have empirically evaluated our approach by comparing its results with the optimal solution and the results from other two approaches. Our simulation results show that our approach provides near optimal solutions in all simulation variations (settings 1 to 4 in Table 2). In fact, the results are closer to the optimal solution when the task types were simulated individually (see Figure 7); 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 centralized optimal solution.

Regarding the comparison with SSIA and ICBAA algorithms, the results show that our approach performs better than the other two algorithms, although the coefficients of variation for all the three algorithms were similar. In addition, there is a large difference in the number of bid messages that were required, specially when compared with 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 large difference in the number of bid messages that were required by TAA and RBTA when compared with our approach.

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 precoordination, ie, agents are being developed by different organizations and should interact with other agents in the absence of prior knowledge, agreements, and possibly not sharing the same communication protocols and world models.[31,32]

In the work of Melo and Sardinha,[33] 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.

Albrecht et al[31] and Stone et al[32] provided more details about the problem of collaboration without precoordination. Albrecht et al[31] also referred to research related to multiagent interaction without prior coordination, such as the works of Hernandez-Leal et al[34] and Liemhetcharat and Veloso,[35] while the work of Stone et al[32] creates a challenge to the AI community to develop research related to ad hoc agent teams.

6 | RELATED WORK

There is a vast literature related to task allocation in multiagent or multirobot systems. Some such work aim 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 organization). 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 Macarthur et al[36] introduced 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 maximize the global utility. The algorithm considers that coalitions are not overlapping, ie, each agent is allocated to one coalition at a time. Constraints between tasks and subtasks are not considered by the algorithm.

In the work of Claes et al,[37] task allocation was addressed through distributed planning in each robot using Monte Carlo tree search. 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. In addition, they assume that the distribution of the orders is known, which allows them to model the probabilities of tasks appearing at each location.

Keshmiri and Payandeh[38] introduced 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 centralized 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.

Lerman et al[39] 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.

Chapman et al[40] proposed a decentralized 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 (ie, agents are homogeneous) and have the same costs to perform a given task. The solution does not consider subtasks, only tasks as atomic units.

Cao et al[41] proposed the allocation of new tasks to groups of robots. When new tasks arise, the allocation is initially performed by a centralized algorithm, which distributes the tasks to the groups of robots. Then, the tasks are allocated within each group in a decentralized manner using an auction-based algorithm. That approach considers that only one new task should be allocated to a group at time and that all new tasks need to be allocated.

Urakawa and Sugawara[42] considered the allocation of tasks that can be divided into subtasks. The agents are organized 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 they 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 of Gunn and Anderson,[30] a solution for allocating new tasks discovered by robots during their missions was put forward. The authors only considered 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 article 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.[30] Thus, it could be said of that approach that there is still a single point of failure within each team, so it is not exactly a decentralized solution like ours.

Turner et al[43] proposed a distributed solution for the reallocation of tasks in order to maximize 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 longer time to be executed. The basic idea is to "move" the tasks in the agents' schedules and also change tasks between the agents 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.

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.[44] In the case of distributed multiagent systems, decentralized POMDPs are used. However, decentralized POMDPs are known to be intractable in general settings.[45]

Most of the work aforementioned propose decentralized approaches, where each robot performs a single task at a time and each task requires only a single robot. In addition, most of them consider only the allocation of atomic tasks (with the exception of the work of Urakawa and Sugawara,[42] where the tasks are divided into subtasks). Most of the described research deals with heterogeneous entities (agents or robots) considering at least some aspect related to their physical capabilities, except other works,[38-40] which consider homogeneous agents. Regarding the use of roles in the system, Gunn and Anderson[30] defined the leader role within each group of robots, while Urakawa and Sugawara[42] used managers and workers in a hierarchical structure. With respect to the addition of new robots to the process; only Macarthur et al[36] considered that possibility, but the variation in resource availability is not taken into consideration. The approach presented in the work of Gunn and Anderson[30] makes considerations on that aspect. Only the approaches proposed in the works of the aforementioned authors[30] and Turner et al[43] consider that, at the end of the allocation process, tasks may not have been allocated, either by temporal constraints as described in the work of the aforementioned authors,[43] or because there is no robot capable of performing the task as in the work of Gunn and Anderson.[30]

6.2 | Allocating an initial set of tasks

In the work of Settimi and Pallottino,[46] a distributed solution for task allocation to a set of heterogeneous robots was presented in which robots' capabilities are considered. Unlike our approach, the solution presented in the work of the aforementioned authors[46] requires that only one task is allocated to each robot and that each task is allocated to one robot only.

In the work of Gernert et al,[47] the authors proposed a decentralized 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 decentralized solution for task allocation among multiple robots based on combinatorial auctions has been introduced by Segui-Gasco et al.[48] 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.

The solutions presented by other works[46-48] focus specifically on atomic tasks, unlike the approach put forward in this paper, which comprises other types of tasks as well.

Luo et al presented a similar set of algorithms that focus on different aspects of the task allocation process.[8,49,50] In the work of Luo et al,[49] the authors described 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 amount 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 Luo et al[50] proposed an extension where tasks with different durations are considered. Another work presented by the aforementioned authors[8] 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 any task. Unlike our work, that work does not consider the allocation of different types of tasks at the same time (which we have solved with the knapsack approach and the reconsideration of partially allocated tasks with the Borda count technique), aspects related to capabilities of robots, the use of roles associated with tasks is not considered either, and we have changed also 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 less messages.

Flushing et al[51] proposed a centralized 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 nonatomic tasks. Nonatomic 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.

Furthermore, Das et al[52] introduced a centralized 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, ie, the subtask being currently carried out and the next subtask to be carried out.

Most of the approaches aforementioned propose decentralized solutions (with the exception of the works of Flushing et al[51] and Das et al[52]), consider heterogeneous entities and some aspect related to their physical capabilities. The possibility of including new tasks in the course of execution was considered only by Gernert et al[47] and Das et al.[52] Das et al[52] assumed that new tasks are considered only in future allocation processes. In the work of Gernert et al,[47] when new tasks are perceived, a new allocation process begins among the robots within communication range. Only Das et al[52] mentioned the possibility of adding new robots to a running allocation process. None of the approaches aforementioned deals with the possibility of agents playing particular roles.

7 | CONCLUSION

A process to allocate tasks efficiently is crucial in many domains, especially if we consider that different tasks can be performed by robots with different capabilities. In this paper, we have proposed a decentralized mechanism for the allocation of different types of tasks to heterogeneous robot teams, considering that they can play different roles and carry out tasks according to their own capabilities.

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

The proposed mechanism considers decomposable simple tasks, for which all the subtasks need to be allocated to the same robot, compound tasks where each subtask needs to be allocated to a different robot, and compound tasks where subtasks can be allocated to any robot. 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 a robot can take from a task type.

Each robot in the proposed solution has a limit on the amount of tasks that it can allocate to itself and this limit may be different for individual robots. When the total capacity of the robots is greater than or equal to the total number of subtasks that need to be allocated, the base algorithms yield very good results. When the number of subtasks is greater than the total capacity of the robots, 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 and a simple auction mechanism to allocate entirely at least some of those outstanding tasks.

The results of the proposed mechanism were compared with the optimal solutions obtained with GNU linear programming kit. The results of various experiments we conducted show that the proposed mechanism has reasonable performance in regards to maximizing the total utility and also regarding the running time needed to reach a solution.

Future work aims to consider aspects such as task prioritizing and other task constraints. Allowing all subtasks to take up more than one unit from the robot task limit (ie, different task weights) would also be interesting. We also intend to evaluate the approach with real robots as mentioned earlier or on robot operating system–based simulations that are currently being developed.

ACKNOWLEDGMENT

We are grateful for the support given by CNPq and CAPES. Tulio Basegio thanks the support given by Federal Institute of Rio Grande do Sul (IFRS), Campus Feliz.

ORCID

Tulio L. Basegio https://orcid.org/0000-0002-0964-9354
Rafael H. Bordini https://orcid.org/0000-0001-8688-9901

REFERENCES

1. Yan Z, Jouandeau N, Cherif AA. A survey and analysis of multi-robot coordination. *Int J Adv Robot Syst*. 2013;10(12):399. http://www.intechopen.com/journals/international_journal_of_advanced_robotic_systems/a-survey-and-analysis-of-multi-robot-coordination. https://doi.org/10.5772/57313
2. Bordini RH, Hübner JF, Wooldridge M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Chichester, UK: John Wiley & Sons; 2007. *Wiley Series in Agent Technology*; vol 8.
3. Brazier FM, Mobach DG, Overeinder BJ, Wijngaards NJ. Supporting life cycle coordination in open agent systems. 2002.
4. Huhns MN, Stephens LM. Multiagent systems and societies of agents. In: Weiss Gerhard, ed. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press; 1999:79-120. *Intelligent Robotics and Autonomous Agents Series*.
5. Correa A. Distributed team formation in urban disaster environments. Paper presented at: 2014 IEEE Symposium on Intelligent Agents (IA); 2014; Orlando, FL. https://doi.org/10.1109/IA.2014.7009459
6. Fanti MP, Ukovich W, Mangini AM, Pedroncelli G. A quantized consensus algorithm for a multi-agent assignment problem. In: Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2013; Manchester, UK. https://doi.org/10.1109/SMC.2013.185
7. 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. Paper presented at: 2015 IEEE International Conference on Mechatronics and Automation (ICMA); 2015; Beijing, China. https://doi.org/10.1109/ICMA.2015.7237692
8. Luo L, Chakraborty N, Sycara K. Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks. *IEEE Trans Robot*. 2015;31(1):19-30. ISSN 1552–3098 https://doi.org/10.1109/TRO.2014.2370831
9. Segui-Gasco P, Shin H-S, Tsourdos A, Segui VJ. Decentralised submodular multi-robot task allocation. Paper presented at: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2015; Hamburg, Germany. https://doi.org/10.1109/IROS.2015.7353766
10. Smith D, Wetherall J, Woodhead S, Adekunle A. A cluster-based approach to consensus based distributed task allocation. Paper presented at: 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP); 2014; Torino, Italy. https://doi.org/10.1109/PDP.2014.87
11. Basegio TL, Bordini RH. An algorithm for allocating structured tasks in multi-robot scenarios. In: *Agent and Multi-Agent Systems: Technology and Applications: 11th KES International Conference, KES-AMSTA 2017 Vilamoura, Algarve, Portugal, June 2017 Proceedings*. Cham, Switzerland: Springer International Publishing AG; 2017:99-109. https://doi.org/10.1007/978-3-319-59394-4_10
12. Scerri P, Kannan B, Velagapudi P, et al. Flood disaster mitigation: a real-world challenge problem for multi-agent unmanned surface vehicles. In: *Advanced Agent Technology AAMAS 2011 Workshops, AMPLE, AOSE, ARMS, DOCM3AS, ITMAS, Taipei, Taiwan, May 2-6, 2011. Revised Selected Papers*. Berlin, Germany: Springer-Verlag GmbH Berlin Heidelberg; 2012:252-269. https://doi.org/10.1007/978-3-642-27216-5_16
13. Murphy RR, Tadokoro S, Nardi D, et al. Search and rescue robotics. In: *Springer Handbook of Robotics*. Berlin, Germany: Springer-Verlag Berlin Heidelberg; 2008:1151-1173. https://doi.org/10.1007/978-3-540-30301-5_51
14. Ramchurn SD, Huynh TD, Ikuno Y, et al. HAC-ER: a disaster response system based on human-agent collectives. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS); 2015; Istanbul, Turkey. http://dl.acm.org/citation.cfm?id=2772879.2772947

15. Korsah GA, Stentz A, Dias MB. A comprehensive taxonomy for multi-robot task allocation. *Int J Rob Res*. 2013;32(12):1495-1512. https://doi.org/10.1177/0278364913496484
16. Gerkey BP, Matarić MJ. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int J Robot Res*. 2004;23(9):939-954. https://www.scopus.com/inward/record.uri?eid=2-s2.0-4444338336&partnerID=40&md5=2ec33de9654d64c07a821075016b3dec
17. Zlot RM. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams* [PhD thesis]. Pittsburgh, PA: The Robotics Institute, Carnegie Mellon University; 2006.
18. Dias MB, Zlot R, Kalra N, Stentz A. Market-based multirobot coordination: a survey and analysis. *Proc IEEE*. 2006;94(7):1257-1270. https://doi.org/10.1109/JPROC.2006.876939
19. Hayes-Roth B. A blackboard architecture for control. *Artif Intell*. 1985;26(3):251-321. https://doi.org/10.1016/0004-3702(85)90063-3
20. Rudenko D, Borisov A. An overview of blackboard architecture application for real tasks. *Sci Proc Riga Tech Univ*. 2007;5:50-57.
21. Murphy RR. *Disaster Robotics*. Cambridge, MA: The MIT Press; 2014.
22. Choi H-L, Brunet L, How JP. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans Robot*. 2009;25(4):912-926. https://doi.org/10.1109/TRO.2009.2022423
23. Conitzer V. Comparing multiagent systems research in combinatorial auctions and voting. *Ann Math Artif Intell*. 2010;58(3-4):239-259. https://doi.org/10.1007/s10472-010-9205-y
24. Makhorin A. Gnu linear programming kit reference manual. 2012. http://www.gnu.org/software/glpk/glpk.html.
25. Bratman ME, Israel DJ, Pollack ME. Plans and resource-bounded practical reasoning. *Comput Intell*. 1988;4(3):349-355. https://doi.org/10.1111/j.1467-8640.1988.tb00284.x
26. Rao AS, Georgeff MP. BDI agents: from theory to practice. In: Proceedings of the First International Conference ON Multiagent Systems (ICMAS). 1995; San Francisco, CA.
27. Boissier O, Bordini RH, Hübner JF, Ricci A, Santi A. Multi-agent oriented programming with JaCaMo. *Sci Comput Program*. 2013;78(6):747-761. https://doi.org/10.1016/j.scico.2011.10.004
28. Koenig S, Keskinocak P, Tovey C. Progress on agent coordination with cooperative auctions. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI); 2010; Atlanta, GA. http://dl.acm.org/citation.cfm?id=2898607.2898883
29. Ghamry KA, Kamel MA, Zhang Y. Multiple UAVs in forest fire fighting mission using particle swarm optimization. Paper presented at: 2017 International Conference on Unmanned Aircraft Systems (ICUAS); 2017; Miami, FL. https://doi.org/10.1109/ICUAS.2017.7991527
30. Gunn T, Anderson J. Dynamic heterogeneous team formation for robotic urban search and rescue. *J Comput Syst Sci*. 2015;81(3):553-567. https://doi.org/10.1016/j.jcss.2014.11.009
31. Albrecht SV, Liemhetcharat S, Stone P. Special issue on multiagent interaction without prior coordination: guest editorial. *Auton Agent Multi-Agent Syst*. 2017;31(4):765-766. https://doi.org/10.1007/s10458-016-9358-0
32. Stone P, Kaminka GA, Kraus S, Rosenschein JS. Ad hoc autonomous agent teams: collaboration without pre-coordination. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI); 2010; Atlanta, GA. http://dl.acm.org/citation.cfm?id=2898607.2898847
33. Melo FS, Sardinha A. Ad hoc teamwork by learning teammates' task. *Auton Agent Multi-Agent Syst*. 2016;30(2):175-219. https://doi.org/10.1007/s10458-015-9280-x
34. Hernandez-Leal P, Zhan Y, Taylor ME, Sucar LE, Munoz de Cote E. Efficiently detecting switches against non-stationary opponents. *Auton Agent Multi-Agent Syst*. 2017;31(4):767-789. https://doi.org/10.1007/s10458-016-9352-6
35. Liemhetcharat S, Veloso M. Allocating training instances to learning agents for team formation. *Auton Agent Multi-Agent Syst*. 2017;31(4):905-940. https://doi.org/10.1007/s10458-016-9355-3
36. Macarthur KS, Stranders R, Ramchurn SD, Jennings NR. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI); 2011; San Francisco, CA. http://dl.acm.org/citation.cfm?id=2900423.2900535
37. Claes D, Oliehoek F, Baier H, Tuyls K. Decentralised online planning for multi-robot warehouse commissioning. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS); 2017; São Paulo, Brazil. http://dl.acm.org/citation.cfm?id=3091125.3091198
38. Keshmiri S, Payandeh S. Multi-robot, dynamic task allocation: a case study. *Intell Serv Robot*. 2013;6(3):137-154. https://doi.org/10.1007/s11370-013-0130-x

39. Lerman K, Jones CV, Galstyan A, Mataric MJ. Analysis of dynamic task allocation in multi-robot systems. CoRR. 2006. http://arxiv.org/abs/cs/0604111
40. Chapman AC, Micillo RA, Kota R, Jennings NR. Decentralised dynamic task allocation: a practical game: theoretic approach. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS); 2009; Budapest, Hungary. http://dl.acm.org/citation.cfm?id=1558109.1558139
41. Cao L, Shun Tan H, Peng H, Cong Pan M. Multiple UAVs hierarchical dynamic task allocation based on PSO-FSA and decentralized auction. Paper presented at: 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO); 2014; Bali, Indonesia. https://doi.org/10.1109/ROBIO.2014.7090692
42. Urakawa K, Sugawara T. Task allocation method combining reorganization of agent networks and resource estimation in unknown environments. Paper presented at: Third International Conference on Innovative Computing Technology (INTECH); 2013; London, UK. https://doi.org/10.1109/INTECH.2013.6653641
43. Turner J, Meng Q, Schaefer G. Increasing allocated tasks with a time minimization algorithm for a search and rescue scenario. Paper presented at: 2015 IEEE International Conference on Robotics and Automation (ICRA); 2015; Seattle, WA. https://doi.org/10.1109/ICRA.2015.7139669
44. 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 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02; 2013; Atlanta, GA. https://doi.org/10.1109/WI-IAT.2013.92
45. Bernstein DS, Zilberstein S, Immerman N. The complexity of decentralized control of Markov decision processes. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI); 2000; San Francisco, CA. http://dl.acm.org/citation.cfm?id=2073946.2073951
46. Settimi A, Pallottino L. A subgradient based algorithm for distributed task assignment for heterogeneous mobile robots. Paper presented at: 52nd IEEE Conference on Decision and Control; 2013; Florence, Italy. https://doi.org/10.1109/CDC.2013.6760447
47. Gernert B, Schildt S, Wolf L, et al. An interdisciplinary approach to autonomous team-based exploration in disaster scenarios. Paper presented at: 2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014); 2014; Hokkaido, Japan. https://doi.org/10.1109/SSRR.2014.7017655
48. Segui-Gasco P, Shin HS, Tsourdos A, Segui VJ. A combinatorial auction framework for decentralised task allocation. Paper presented at: 2014 IEEE Globecom Workshops (GC Wkshps); 2014; Austin, TX. https://doi.org/10.1109/GLOCOMW.2014.7063637
49. Luo L, Chakraborty N, Sycara K. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. Paper presented at: 2013 IEEE International Conference on Robotics and Automation; 2013; Karlsruhe, Germany. https://doi.org/10.1109/ICRA.2013.6630994
50. Luo L, Chakraborty N, Sycara K. Distributed algorithms for multirobot task assignment with task deadline constraints. *IEEE Trans Autom Sci Eng*. 2015;12(3):876-888. https://doi.org/10.1109/TASE.2015.2438032
51. Flushing E, Gambardella LM, Di Caro GA. A mathematical programming approach to collaborative missions with heterogeneous teams. Paper presented at: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems; 2014; Chicago, IL. https://doi.org/10.1109/IROS.2014.6942590
52. Das GP, McGinnity TM, Coleman SA. Simultaneous allocations of multiple tightly-coupled multi-robot tasks to coalitions of heterogeneous robots. Paper presented at: 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO); 2014; Bali, Indonesia. https://doi.org/10.1109/ROBIO.2014.7090496

How to cite this article: Basegio TL, Bordini RH. Allocating structured tasks in heterogeneous agent teams. *Computational Intelligence*. 2019;35:124–155. https://doi.org/10.1111/coin.12194