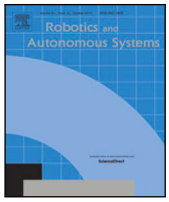
Robotics and Autonomous Systems 180 (2024) 104762

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



On a dynamic and decentralized energy-aware technique for multi-robot task allocation

Menaxi J. Bagchi *, Shivashankar B. Nair, Pradip K. Das

Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Guwahati, 781039, Assam, India

ARTICLE INFO

Keywords:
Decentralized dynamic task allocation
Energy-based task allocation
Precedence-constrained task allocation
Multi-robot systems
Multi-agent systems
Dynamic environment

ABSTRACT

In the real world, multi-robot systems need to deal with *on-the-fly* (runtime) arrivals of new sets of tasks. This entails repeated adjustments of their current task allocations to include the newer ones while also ensuring that the overall performance does not degrade. This paper proposes a decentralized and distributed dynamic task allocation algorithm to handle this issue in a multi-robot scenario. The proposed work provides a conflict-free allocation of a set of tasks constituting a job to robots and minimizes the *total execution time*. These jobs can comprise multiple independent and/or dependent tasks or a combination thereof, which are injected *on-the-fly* into a network of robots. The dependent tasks of a job are related by precedence constraints that specify the ordering or dependencies between pairs of tasks. The work also describes a decentralized adaptive energy threshold mechanism for determining whether or not a robot needs to visit a battery stockpile after the execution of a task. Conflicting task selections among the robots in this decentralized set-up are resolved using mobile agents during runtime. Apart from allocating tasks to the robots, these mobile agents exploit the benefits of centralized and decentralized systems and provide an advantage over auction-based task allocation algorithms. The *proposed* algorithm takes into consideration the energy requirements, both during the task allocation process and actual execution. The *proposed* algorithm also caters to strategies to deal with delays caused by obstacles and congestion during the actual execution of the tasks. Experiments conducted using *Webots*, an open-source robot simulator, and *Tartarus*, a multi-agent platform, authenticate the efficacy of the *proposed* algorithm compared to other prominent task allocation algorithms in terms of minimization of *average waiting time*, *total task allocation time*, *total job allocation time*, and *total execution time* of an experiment.

1. Introduction

The use of a Multi-Robot System (MRS) in a variety of applications has risen over the years due to advancements in processing capabilities and sensor technology. MRSs are employed in a number of applications such as industry automation, construction, warehouse management, healthcare, explorations, surveillance, search and rescue, and many more [1]. Multiple robots can efficiently and concurrently handle tasks that are distributed over a large area and can save time compared to a single robot. Additionally, if the robots are heterogeneous, different types of jobs can be effectively addressed. MRS algorithms can be implemented using two methods: Centralized or Decentralized [2]. In the former, a central entity is in charge of decision-making and coordination among the robots. Though easier to implement, this method faces many challenges due to its limited scalability, single-point failure, requirement of a consistent communication link between the robots and the central entity, etc. It is thus not very suitable for controlling complex situations, especially those involving many heterogeneous robots that tend to increase the computation and communication overheads at the central entity. Decentralized and distributed coordination form a better option when heterogeneous robots need to address the tasks that arrive *on-the-fly* into the network of robots. Decentralized and distributed MRS approaches do not make use of a central controlling entity [2]. In this approach, every robot is autonomous and can make independent decisions, making the system more robust to failures. Multi-Robot Task Allocation (MRTA), a notable area of research in an MRS, is defined as the problem of deciding which robot should execute a specific task given a chain of tasks while trying to achieve a set of objectives. MRTA problems are often classified using the taxonomy presented by Gerkey and Matarić [3]. They differentiate Single-Task (ST) and Multi-Task (MT) robots based on whether the robots can execute one task or multiple tasks at a time. Likewise, Single-Robot (SR) and Multi-Robot (MR) tasks are based on whether the tasks require one robot or multiple robots for their execution.

* Corresponding author.
E-mail address: menaxi@iitg.ac.in (M.J. Bagchi).

https://doi.org/10.1016/j.robot.2024.104762
Received 1 May 2024; Received in revised form 17 July 2024; Accepted 21 July 2024
Available online 23 July 2024

They also distinguish between Instantaneous Assignment (IA) vs. Time-extended Assignment (TA). In IA problems, each robot is assigned only one task at a time, whereas in the latter problem, each robot is assigned several tasks that must be executed in a specific order. The iTax taxonomy proposed by Korsah et al. [4] augments the taxonomy presented in [3] by taking into account the issues of interrelated utilities and constraints. They have distinguished between *No Dependencies*, *In-schedule Dependencies*, *Cross-schedule Dependencies* and *Complex Dependencies*. *No Dependencies* deal with simple or compound tasks with independent robot-task utilities. *In-schedule Dependencies* handle simple or compound tasks for which robot-task utilities have intra-schedule dependencies, while *Cross-schedule Dependencies* deal with simple or compound tasks for which robot-task utilities have both inter-schedule and intra-schedule dependencies. *Complex Dependencies* handle the task allocation for complex tasks for which robot-task utilities have inter-schedule dependencies in addition to *In-schedule Dependencies* and *Cross-schedule Dependencies* for simple and compound tasks. Nunes et al. [5] have augmented the Time-extended Assignment (TA) part of the taxonomy in [3] by temporal constraints in the form of time windows (TA:TW) and synchronization and precedence constraints (TA:SP). Numerous methods of allocating tasks in an MRS have been presented in [6]. MRTA algorithms can be divided into three approaches: Market-based [7], Behavior-based [8], and Optimization-based [9].

Whitbrook et al. [10] have proposed an algorithm for dynamic task allocation where reassignment of the already allocated tasks takes place whenever a dynamic event occurs, such as the arrival of a *new task*, addition of new vehicles, deletion of existing vehicles, etc. Yang et al. [11] have proposed a *partial reassignment* algorithm to tackle dynamic events. Both these works report the use of the Performance Impact (PI) algorithm [12] as their baseline and have used reassignment of the previously allocated tasks to deal with the dynamic events. Hence, for highly dynamic environments where the reassignment process occurs frequently, the reassignment of old tasks increases the time taken to reach a conflict-free allocation and leads to wastage of computation. Choi et al. [13] have proposed a distributed Consensus-based Bundle Algorithm (CBBA) that uses an auction phase followed by a consensus for task allocation. Jha and Nair [14] have described a distributed task allocation algorithm called *TANSA* that uses mobile agents to allocate multiple instances of tasks in an MRS. In their work, all the decision-making is performed by mobile agents. The robots, on the contrary, act as passive agents. Some works [15,16] consider the energies of the robots during the execution phase rather than in the allocation phase. They do not address the task allocation problem but focus on how to efficiently recharge batteries once their energy is depleted. A resource-based task allocation algorithm where tasks are executed on a first-cum-first-serve basis has been proposed in [17,18]. The author in [19] has proposed a resource-based task allocation algorithm for a multi-robot system. A resource in [17–19] refers to any expendable supply that the robot consumes during task execution, such as energy in the battery. The works described in [10–19] have not considered dependencies among the tasks. In many real-life scenarios such as search and rescue, warehouse automation, pickup and delivery, inspection jobs, etc., the outcome of one task is necessary for commencing the execution of another [5]. Hence, it is important to consider dependencies among the tasks. Liu et al. [20] have solved precedence-constrained MRTA problems. The authors in [20] have considered a static task allocation problem where all the tasks are known *a priori*. The works reported in [10–14,20] have not considered energy requirements of the tasks at the time of the task allocation and also during actual task execution. In the real world, the batteries on board a mobile robot must have sufficient energy to ensure the execution of the tasks allocated to it. Thus, it is essential that this energy too, is taken into account during the task allocation process. Auction-based MRTA algorithms use market-based approaches where the auctioneer can either be a central server or one of the bidders [13]. All robots are considered to be stationary while the negotiations are going on, an assumption which is hard to impose in the real world [14]. Most of the MRTA problems focus on static task allocation problems. In contrast, in the real world, the environment could be dynamic, where new tasks could appear during the execution of the previous set of tasks. The *DfMA* algorithm [21] makes use of concepts taken from both dataflow computing and mobile agent technology to ensure efficient execution of multiple jobs simultaneously in an MRS. New jobs, which comprise a set of tasks, are injected *on-the-fly* into a network of *Processing Systems* (*PS*s) by their respective mobile agents, and are executed with minimal interference to those that are currently being executed, thus, preventing the stalling of the entire system of robots due to the addition and execution of new jobs. These mobile agents facilitate *on-the-fly* programming of entities in a network by bringing in the code for the entity within the network.

In this paper, we propose a decentralized and distributed dynamic multi-robot task allocation algorithm. The algorithm handles a dynamic influx of n jobs without any prior information about their numbers or times of arrival into a network of heterogeneous mobile robots. In the present implementation, each job could consist of a set of independent tasks or a combination of independent and dependent tasks. The proposed work describes the use of mobile agents to allocate all the tasks of these jobs. The mobile agents also resolve the conflicting selection of tasks among the robots, if any, which arises due to the decentralized selection of tasks by the robots. The objective of the proposed MRTA algorithm is to provide a conflict-free allocation of the tasks within these jobs to the robots in a way so as to reduce the *total execution time* of all the n jobs taken together. The proposed work does not optimize the energy consumption of the robots. According to a recent paper by Datsko et al. [22], "*energy-aware*" means taking into account energy consumption. The *proposed* algorithm is energy-aware because it considers the energy requirements of the tasks and the energy available in the robots at the time of the task allocation process and during their actual executions. It also uses an adaptive rather than a fixed energy threshold for recharging the robots. Strategies for managing unexpected delays in task execution during runtime that might crop up due to obstacles, congestion due to multiple robots crowding at a particular place at the same time, etc., are also addressed in this work. The main contributions of the paper are as follows:

1. A decentralized and distributed precedence-constrained task allocation algorithm for a multi-robot system that deals with *on-the-fly* arrival of jobs into a network of heterogeneous robots is proposed. The *proposed* algorithm uses mobile agents for resolving the conflicting selections of tasks among the robots and for allocating the tasks to the suitable robots.
2. The *proposed* algorithm is energy-aware. An adaptive energy threshold that replaces the conventional fixed energy threshold for recharging the robots is used. The *proposed* algorithm takes into account the energy requirements of the tasks and the energy available in the robots while estimating the costs and the bid values of the tasks during the task allocation process and also during actual execution.
3. Strategies for handling unexpected delays during actual execution are proposed.

The rest of the paper is organized as follows: Section 2 provides a brief description of mobile agents as they have been used in the work described herein. Section 3 describes the problem formulation and the objective of the work. Section 4 illustrates the methodology, followed by Section 5, which describes the experiments conducted and results obtained. Conclusions and the possible future work are included in Section 6.

2. Mobile agents

Like conventional agents, mobile agents are autonomous entities that can learn, adapt, carry programs and data as payloads, clone,

and even terminate themselves [23]. An agent can be programmed to communicate with other agents and overcome failures [24]. They differ from their static counterparts because they are mobile and can flit through computing nodes in a network. The mobile agents are perfect candidates for carrying out asynchronous, distributed, and parallel executions [14]. These agents can perform local executions by bringing the computations close to the edge devices. Mobile agents have been used in a wide range of applications ranging from wireless sensor networks [25], security [26], e-learning [27], robot control [28], e-commerce [29] and many more. These agents are used in an MRS for various distributed applications. Pasodas et al. [30] have shown that these agents can provide better performance than the traditional communication methods for controlling mobile robots. Jha and Nair [31] have illustrated the use of mobile agents for learning and sharing information in a distributed environment comprising a group of networked robots. Nestinger et al. [32] have used mobile agents to coordinate the activities of robots in a factory environment and to enable dynamic deployment of control algorithms and tasks in automation systems without the need to halt the system for task modification. Jung and Yeom [24] have proposed the use of mobile agents to transfer a message to many receivers and have shown the advantage of mobile agents for message transfer as compared to a unicast protocol for message transfer. Bandyopadhyay and Paul [33] have shown the efficiency of mobile agents in message delivery in a highly dynamic ad hoc network where it is difficult to design routing schemes for efficient communication between any source and destination.

A mobile agent platform is an execution environment that provides means to create, maintain, and execute agents within a network. It also aids in the migration of mobile agents from one platform to another. Many mobile agent platforms have been developed over the years. *Jade* [34], *Concordia* [35] and *Aglets* [36] are Java-based mobile agent platforms while *Mobile-C* [37] is based on C/C++. *Typhon* [38] and *Tartarus* [39] are some examples of Prolog-based mobile agent platforms.

3. Formal description of the proposed MRTA algorithm

This section formally describes the proposed MRTA algorithm.

We have used an MRS scenario consisting of a group of networked heterogeneous mobile robots, each having a processor on board. A robot can, in addition to the execution of tasks, host an agent platform within. The robots form a fully connected network via a WiFi router. Let $R = \{R_1, R_2, \ldots, R_r\}$, where $r \in \mathbb{N}$, be a set of networked mobile robots. Each robot $R_i \in R$ is capable of executing either a single type of task, τ^j or a set of different types of tasks, $\{\tau^1, \tau^2, \tau^3, \ldots, \tau^j\}$, where $j \in \mathbb{N}$. A task is said to be compatible with a robot if the latter can execute it.

Each robot, R_i, maintains the following lists:

- **Temporary List**: Whenever a *new job* is injected into the network of mobile robots, the *temporary list* of a robot R_i, $tmp\_lst_{R_i}$, stores all information regarding the tasks of this *new job*. The contents of this list and the order of the jobs present inside the list are the same for all the robots in the network.
- **Task List**: This list within a robot R_i, represented as $tsk\_lst_{R_i}$, stores all information regarding the tasks that are currently allocated to the robot.
- **Allocation List**: The *allocation list* of a robot R_i, $alloc\_lst_{R_i}$, stores information regarding the assignment of the tasks of all the jobs to the robots in the network. Thus, the contents of this list will be the same for all the robots in the network.
- **Bid List**: The *bid list* of a robot R_i, $bid\_lst_{R_i}$, stores the estimated bid values and the estimated costs of performing all the tasks that are currently assigned to the robot. $\mu_{R_i^n}$ is the estimated bid value of the n^{th} task in $tsk\_lst_{R_i}$. The estimated cost of performing the n^{th} task in $tsk\_lst_{R_i}$ is represented as $\lambda_{R_i^n}$.

These lists are elucidated in detail in Section 4. Apart from the information regarding the lists, the robot also keeps track of its location, $loc\left(R_i\right)$. $loc\left(.\right)$ represents location as 2D coordinates.

We define $J = \{J_1, J_2, J_3, \ldots, J_n\}$, where $n \in \mathbb{N}$, as a set of n jobs which are injected *on-the-fly* (during runtime) into the network of robots. The number of jobs, their respective arrival times, and the number of constituent tasks of each job are not known *a priori* by the *proposed* algorithm. Each job $J_n \in J$ comprises a set of h tasks of the form, $J_n = \left\{\tau^1_{(n,1)}, \tau^2_{(n,2)}, \tau^1_{(n,3)}, \tau^5_{(n,4)}, \ldots, \tau^j_{(n,h)}\right\}$ where n represents the *job_id*, τ^j is one of the constituent tasks of J_n and, h represents the order in which the task $\tau^j_{(n,h)}$ appears in the set of tasks of job J_n, where $n, h, j \in \mathbb{N}$. Here, h need not necessarily indicate the order of execution of the concerned tasks. The value of h could be different for different jobs, i.e., different jobs could contain varying numbers of constituent tasks. As can be seen, tasks within J_n could be of the same type, distinct, or a combination of both. The information regarding each task $\tau^j_{(n,h)}$ of a job J_n, is expressed as a tuple of the form $\left\langle loc\left(\tau^j_{(n,h)}\right), pred\left(\tau^j_{(n,h)}\right), succ\left(\tau^j_{(n,h)}\right)\right\rangle$. $loc\left(\tau^j_{(n,h)}\right)$ refers to the location where the task is to be executed or the execution location of that task, $pred\left(\tau^j_{(n,h)}\right)$ is the predecessor set of $\tau^j_{(n,h)}$ which contains the predecessor tasks of $\tau^j_{(n,h)}$. The predecessor tasks must be executed before the commencement of $\tau^j_{(n,h)}$. $pred\left(\tau^j_{(n,h)}\right)$ is a subset of J_n, $pred\left(\tau^j_{(n,h)}\right) \subset J_n$. $succ\left(\tau^j_{(n,h)}\right)$ is the successor set of $\tau^j_{(n,h)}$ which contains the successor tasks of $\tau^j_{(n,h)}$. The successor tasks must be executed after the execution of $\tau^j_{(n,h)}$. $succ\left(\tau^j_{(n,h)}\right)$ is also a subset of J_n, $succ\left(\tau^j_{(n,h)}\right) \subset J_n$.

A job could consist of a set of independent tasks or a set of independent and dependent tasks. A job consisting of a set of independent and dependent tasks can be represented in the form of a precedence graph where the nodes of the graph represent tasks that constitute the job while the edges depict the precedence constraints between a pair of tasks (nodes). An edge directed from $\tau^j_{(n,h)}$ to $\tau^k_{(n,q)}$ signifies that $\tau^j_{(n,h)}$ is a predecessor of $\tau^k_{(n,q)}$ and the execution of $\tau^j_{(n,h)}$ should be completed before starting the execution of $\tau^k_{(n,q)}$. For a job consisting of a set of independent tasks, the predecessor and successor sets of the tasks of such a job would be empty. The predecessor and the successor sets model the dependencies between the tasks of a job (the nodes of the precedence graph) consisting of independent and dependent tasks.

The allocation of a job refers to the conflict-free allotment of its constituent tasks to the appropriate robots. It is assumed that for every task within any job, there exists at least one robot that can execute it. Therefore, given a task, since there exists a robot that can execute it, the proposed MRTA will ensure the allocation of all tasks amongst the robots. As some of the robots in the set R can execute different types of tasks, the *proposed* algorithm can handle different types of tasks even in situations where the number of robots is less than the types of tasks comprising the jobs. It is also assumed that if a task can be executed in time t by a robot, other robots capable of executing the same task also take the same amount of time t to execute it.

Each job J_n is associated with a mobile agent, termed as the *Initialization Agent*, IA_n. This mobile agent is responsible for informing all the robots in the network about the job. The mobile agent also resolves the conflicting selections of the tasks of J_n, if any, among the robots and allocates the tasks of the job to the appropriate robots.

Fig. 1 shows a fully connected heterogeneous network of four mobile robots R_1, R_2, R_3, and R_4 each hosting an agent platform, where all the four robots are connected via a WiFi router. The WiFi router has been used only to form a connected network and merely acts as a medium for communication amongst the entities, viz., the robots. The mobile agent IA_1 is shown carrying a job J_1, consisting of independent and dependent tasks. J_1 is represented in the form of a precedence graph. The mobile agent IA_2 carries a job J_2 consisting of three independent tasks as its payload. J_2 cannot be represented as a

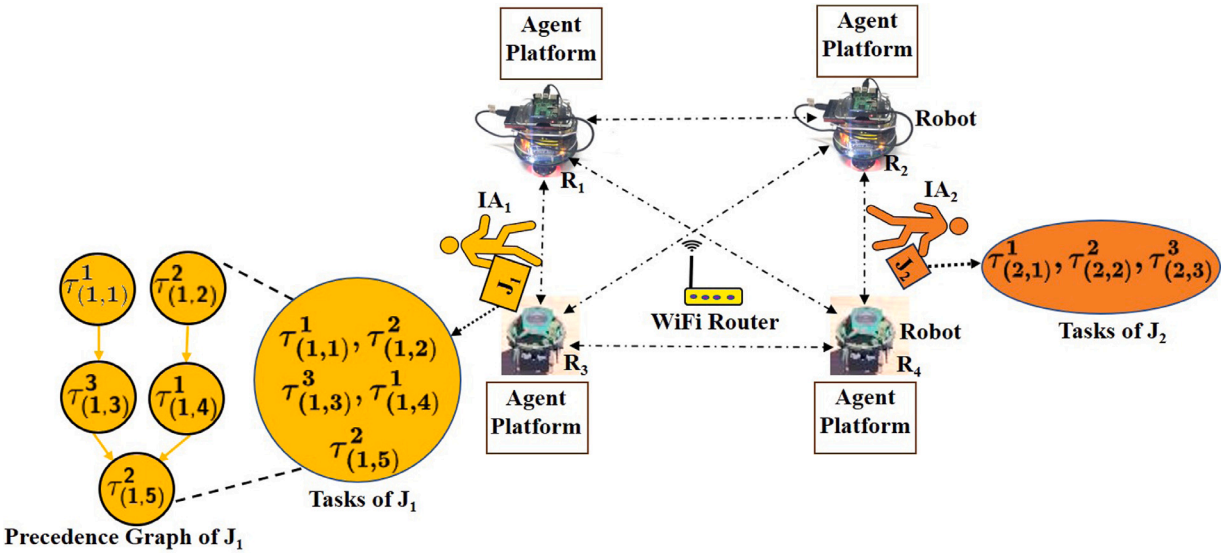


Fig. 1. Architecture of the proposed MRS.

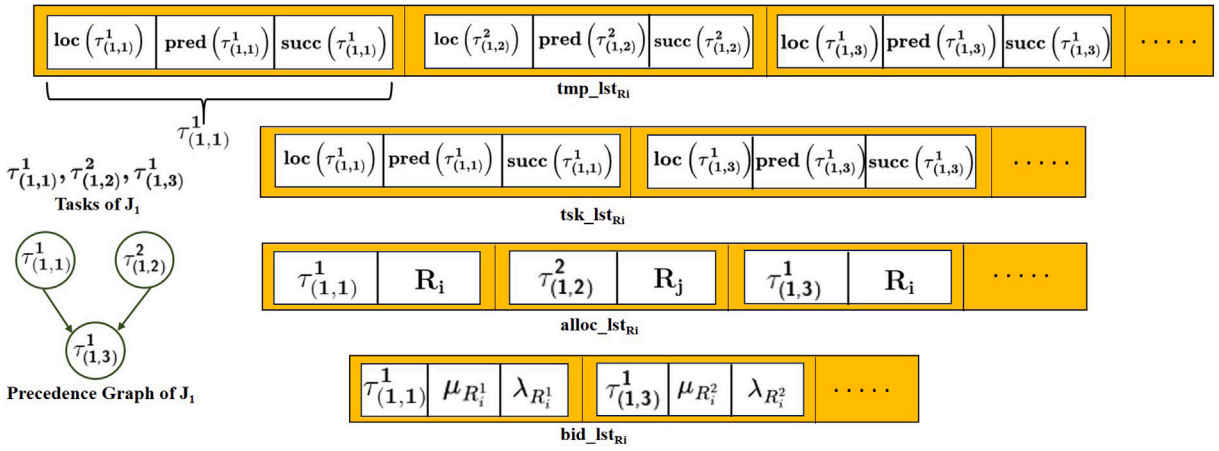


Fig. 2. Lists within a Robot, R_i.

precedence graph because all its constituent tasks are independent of each other. The bi-directional arrows indicate the directions in which the mobile agents can move. The mobile agents can move between robots in one hop as they form a fully connected network. Multiple mobile agents (carrying their respective jobs) could also inhabit the network of mobile robots at a given instant of time.

Fig. 2 shows the contents of the lists which a robot maintains. Though the lists could contain the information of multiple jobs, to make things simple, only a single job J_1, along with its associated precedence graph, is shown. The $tmp\_lst_{R_i}$ and the $alloc\_lst_{R_i}$ contain the information and the allocation details of all the tasks of J_1. The $tsk\_lst_{R_i}$ and the $bid\_lst_{R_i}$ contain the information and the estimated costs and estimated bid values only for the tasks allocated to R_i.

The objective of the proposed work is to provide a conflict-free allocation of the tasks of the n jobs to the robots to minimize the *total execution time* of all these jobs by minimizing the waiting times of their constituent tasks.

The *total execution time* of n jobs that needs to be minimized can be expressed as:

$$t_{tot\_exe} = C_{J_n} - \mathcal{A}_{J_1} \tag{1}$$

t_{tot_exe} in (1) denotes the *total execution time* of n jobs. C_{J_n} denotes the instant of time at which the last job, J_n completed its execution. $\mathcal{A}_{J_1}$ indicates the time of arrival of the first job J_1 into the network of robots.

Whenever a *new job* is injected into the network of robots, the corresponding tasks of that job are allocated to those robots that can start the execution of the tasks at the earliest. This minimizes their waiting times. The increase in the time taken to reach the locations of the tasks that have been shifted because of the insertion of the tasks of this *new job* into the *task lists* of the robots is taken care of during the task allocation process. A robot is allocated a task only if it has the capability of executing that task.

The objective of the proposed task allocation algorithm is subjected to the following three constraints:

Constraint #1: A task cannot be allocated to more than one robot i.e.

$$tsk\_lst_{R_y} \cap tsk\_lst_{R_z} = \emptyset,\ \forall R_y, R_z \in R\ ;\ \forall R_y \neq R_z \tag{2}$$

where $tsk\_lst_{R_y}$ and $tsk\_lst_{R_z}$ are the *task lists* of robots R_y and R_z, respectively.

Constraint #2: This governs the precedence constraints between the tasks $\tau^j_{(n,h)}$ and $\tau^z_{(n,k)}$ of a job J_n. It is formally stated below-

$$C_{\tau^j_{(n,h)}} < \mathcal{S}_{\tau^z_{(n,k)}},\ \forall \tau^j_{(n,h)} \in pred\left(\tau^z_{(n,k)}\right)\ ;\ \forall \tau^z_{(n,k)} \in succ\left(\tau^j_{(n,h)}\right) \tag{3}$$

where $S_{\tau^z_{(n,k)}}$ and $C_{\tau^j_{(n,h)}}$ represent the time instant at which a task $\tau^z_{(n,k)}$ started its execution and the time instant at which a task $\tau^j_{(n,h)}$ which is a predecessor task of $\tau^z_{(n,k)}$ completed its execution, respectively. $pred\left(\tau^z_{(n,k)}\right)$ is the predecessor set of task $\tau^z_{(n,k)}$. $succ\left(\tau^j_{(n,h)}\right)$ is the successor set of task $\tau^j_{(n,h)}$. *Constraint #2* implies that a task cannot start its execution before the completion of execution of its predecessor task(s). The tasks of different jobs are, however, not related by any precedence constraint.

Constraint #3: The robots consume energy while performing a task. Therefore, the energy requirement of a task, as well as the energy possessed by a robot, should be considered while estimating the cost and the bid value for the task at the time of the task allocation process. This is because the allocation of a task to a robot depends on the estimated bid value offered by the robot for that task. Defining a fixed global energy threshold for all robots, as in [15,40], forces the robots to visit the battery stockpile when this threshold is reached. This could happen even though the robots have enough energy to execute some of the succeeding tasks in their *task lists* before visiting the battery stockpile. Therefore, using such a fixed energy threshold can cause unnecessary visits to the battery stockpile, thereby increasing the waiting times for the tasks and consequently the *total execution time*. In the *proposed* algorithm, *Constraint #3* forces the robots to possess dynamic/adaptive energy thresholds. Such a threshold provides flexibility and reduces unessential visits to the battery stockpile, thereby saving on the waiting times of the tasks and the *total execution time*. The dynamic threshold depends on the current energy available in the robot, the estimated energy required to execute the next task in its *task list*, and the estimated energy needed to visit the battery stockpile from the execution location of that next task. The choice of when to visit the battery stockpile is taken independently by each robot for its own *task list* without assistance from any centralized entity. Dynamic thresholds could thus be different for different robots. *Constraint #3* pertains to the relationship between the energy requirement of a task and the energy available in a robot and is expressed as

$$\varepsilon_{R_i^p} > e_{R_i^{p+1}} + e_{R_i}\left(loc\left(R_i^{p+1}\right) \rightarrow loc\left(\mathcal{BS}\right)\right) \quad , \; p = \left\{1, 2, \ldots, \left|tsk\_lst_{R_i}\right|\right\}; \; \forall R_i \in R \tag{4}$$

where, $\varepsilon_{R_i^p}$ is the energy remaining in R_i after execution of the p^{th} task in its *task list*. In the case of the first task in the *task list*, it will indicate the initial energy endowed to the robot. $e_{R_i^{p+1}}$ is the energy required by R_i to execute the $(p+1)^{th}$ task in its *task list*, $tsk\_lst_{R_i}$. $e_{R_i^{p+1}}$ represents the energy required by R_i to travel from its current location to the execution location of the $(p+1)^{th}$ task in its *task list* plus the energy required to accomplish that task. In the case of tasks other than the first task in its *task list*, the current location would be the location of the p^{th} task (previous task) in its *task list*. $e_{R_i}\left(loc\left(R_i^{p+1}\right) \rightarrow loc\left(\mathcal{BS}\right)\right)$ is the energy required by R_i to move from the execution location of the $(p+1)^{th}$ task in its *task list*, $loc\left(R_i^{p+1}\right)$, to the location of the battery stockpile, $loc\left(\mathcal{BS}\right)$. $\rightarrow$ indicates the movement of a robot from one location to another. $\left|tsk\_lst_{R_i}\right|$ represents the *number* of elements in $tsk\_lst_{R_i}$. This constraint implies that the minimum energy available in R_i before executing a task from its *task list* should be greater than the energy required to execute that task plus the energy required to travel from the execution location of that task to the battery stockpile. *Constraint #3* is checked for each task in the *task list* of a robot to ensure that adequate energy is available not only to execute the task but also to visit the battery stockpile, if required, after executing that task. *Constraint #3* must hold for each task in the *task lists* of all robots. If it does not hold for any one of the tasks, then the robot needs to refill its energy before executing that task.

The proposed MRTA is energy-aware, in the sense that if the energy within the batteries powering the robot is depleted during the execution of the tasks allocated to it, the robot is forced to visit a battery stockpile to replace its onboard batteries with fully charged ones. In the proposed MRTA algorithm described in Section 4, each robot can perform only one task at a time. Further, though each task requires only one robot to perform it, each robot could be assigned multiple tasks. The proposed task allocation algorithm, thus, falls under the category of *ST-SR-TA = Single-Task robots, Single-Robot tasks, Time-extended Assignment* as defined in [3]. As each task requires only one robot for its execution, the proposed task allocation algorithm provides a conflict-free allocation of the tasks of the jobs to the robots in the network where a particular task is allocated to a single robot. *ST-SR-TA* problems are NP-hard [41] as they are complex, combinatorial decision problems [10]. The work proposed in this paper also considers precedence constraints among the tasks of a job. This imposes partial order relationships between pairs of tasks of a job allocated to the same or different robots. Hence, it can also be categorized as *ST-SR-TA:SP* [5]. *In-schedule Dependencies* [4] commonly occur in Time-extended Assignment (TA) problems and are also enforced by precedence constraints between a pair of tasks of a job that are allocated to the same robot. *Cross-schedule Dependencies* [4] are imposed by precedence constraints when two or more tasks of the same job related by precedence constraints are allocated to different robots.

The proposed MRTA algorithm can be applied in various scenarios such as cleaning of factory sites, search and rescue operations where tasks may need to be executed in a particular order (as in a disaster scenario where roads must be cleared to gain access to the victims), in a warehouse (where a series of tasks must be executed in some predefined order - e.g. picking, packing and transporting items), in a pickup and delivery scenario where an item must be picked up before delivery and for inspection jobs where some objects with a higher priority need to be inspected before the lower priority ones.

4. Methodology

In this section, we describe the proposed task allocation algorithm for a multi-robot system.

4.1. Proposed task allocation algorithm

The proposed task allocation algorithm consists of two stages - a *task selection* stage performed by the robots and a *contention elimination* stage performed by the *Initialization Agents*. These two stages are executed consecutively for each job before starting the task allocation process for the subsequent job.

The proposed work uses a similar setting as in [21] where jobs are injected *on-the-fly* into a fully connected network of mobile robots by their respective *Initialization Agents*. The IA_n carries the job J_n as its payload and provides the information of the constituent tasks of the job to each robot by migrating from one robot to the other in the network. This information includes the locations where the tasks are to be executed and predecessor and successor sets of the constituent tasks within the job. The information about the job is appended to the *temporary lists* of the robots by the IA_n. The IA_n knows the number of robots present in the fully connected network.

After sharing the job information with the robots, the IA_n resides inside the agent platform in any one of the robots and waits for the *task selection* stage of the task allocation algorithm for that robot to get completed so that it can commence the *contention elimination* stage of the algorithm. Since jobs could arrive asynchronously, multiple *Initialization Agents* could be present inside the network of robots at a

given instant of time. In the proposed work, the new jobs together with the older ones, are executed concurrently, similar to those in [21].

4.1.1. Task selection stage

The robots select the tasks independently and concurrently in a decentralized and distributed manner. During the *task selection* stage of the task allocation algorithm, each robot R_i selects the compatible tasks of the jobs from its *temporary list*, $tmp\_lst_{R_i}$ and inserts them at appropriate positions inside its $tsk\_lst_{R_i}$.

Algorithm 1: Task Selection Stage of Task Allocation Algorithm performed by R_i

Input: $tmp\_lst_{R_i}$ of R_i
Output: Updated $tsk\_lst_{R_i}$ and $bid\_lst_{R_i}$

```
1: while tmp_lst_Ri is not empty do
2:    start_index, end_index = find_tasks_of_job(J_n)
3:    for a ← start_index to end_index do
4:       compatibility = check_compatibility(tmp_lst_Ri [a])
5:       if compatibility==1 then
6:          d_l = check_dependency(tmp_lst_Ri [a])
7:       end if
8:       for p ∈ d_l do
9:          Estimate cost, λ_{R_i^p} and bid value, μ_{R_i^p} using Eqs. (5a) and (6)
10:      end for
11:      Determine min_{p∈d_l}(μ_{R_i^p})
12:      update_task_list()
13:      update_bid_list()
14:   end for
15:   estimate_enreq() for each task in tsk_lst_Ri using Eq. (7)
16:   Re-estimate costs and bid values of the tasks of J_n in tsk_lst_Ri which are
      predicted to be executed after visiting battery stockpile using Eqs. (5b)
      and (6)
17:   update_bid_list()
18:   Delete all tasks of job J_n from tmp_lst_Ri
19:   flag = job_id
20:   while flag != 0 do
21:      Before proceeding with the selection of the tasks for the next job,
         wait till conflict resolution and the task allocation of the job J_n is
         complete.
22:   end while
23: end while
```

The *task selection* stage of the task allocation algorithm running on a robot R_i is depicted by Algorithm 1. The function *find_tasks_of_job* in line number 2 finds the indices where the first and the last tasks of a job, J_n, are present inside $tmp\_lst_{R_i}$. All the other tasks of J_n are present at consecutive positions inside $tmp\_lst_{R_i}$ in between these two indices. The *check_compatibility* function in line number 4 of Algorithm 1 determines whether R_i is capable of executing a particular task. If that task is found to be compatible, the *check_dependency* function in line number 6 of Algorithm 1 finds the feasible positions where the task could be inserted into $tsk\_lst_{R_i}$. While inserting each task into $tsk\_lst_{R_i}$, *In-schedule Dependencies* are taken care of.

Based on the objective of the proposed work, the estimated cost of performing a task (estimated execution cost) $\tau^j_{(n,i)}$ (p^{th} task in $tsk\_lst_{R_i}$) is defined as the amount of time taken by R_i to start its execution. It is calculated using Eqs. (5a) and (5b).

$$\lambda_{R_i^p} = \begin{cases} t\left(loc\left(R_i\right) \rightarrow loc\left(R_i^p\right)\right) & \text{if } p = 1 \\ t\left(loc\left(R_i\right) \rightarrow loc\left(R_i^1\right)\right) + & \\ \sum_{k=2}^{p} t\left(loc\left(R_i^{k-1}\right) \rightarrow loc\left(R_i^k\right)\right) & \\ + \sum_{k=1}^{p-1} t_{R_i^k} & \text{if } p > 1 \end{cases} \tag{5a}$$

$$\lambda_{R_i^p} = \begin{cases} \alpha * \left\{ t\left(loc\left(R_i\right) \rightarrow loc\left(R_i^p\right)\right)\right\} + & \\ (1-\alpha) * \left\{ t\left(loc\left(R_i\right) \rightarrow loc\left(\mathcal{BS}\right)\right) + \right. & \\ \left. t\left(loc\left(\mathcal{BS}\right) \rightarrow loc\left(R_i^p\right)\right)\right\} & \text{if } p = 1 \\ \beta * \left\{ t\left(loc\left(R_i\right) \rightarrow loc\left(R_i^1\right)\right)\right\} + & \\ (1-\beta) * \left\{ t\left(loc\left(R_i\right) \rightarrow loc\left(\mathcal{BS}\right)\right) + \right. & \\ \left. t\left(loc\left(\mathcal{BS}\right) \rightarrow loc\left(R_i^1\right)\right)\right\} + & \\ \sum_{k=2}^{p} \left[\delta * \left\{ t\left(loc\left(R_i^{k-1}\right) \rightarrow loc\left(R_i^k\right)\right)\right\} + \right. & \\ (1-\delta) * \left\{ t\left(loc\left(R_i^{k-1}\right) \rightarrow loc\left(\mathcal{BS}\right)\right) + \right. & \\ \left. \left. t\left(loc\left(\mathcal{BS}\right) \rightarrow loc\left(R_i^k\right)\right)\right\}\right] & \\ + \sum_{k=1}^{p-1} t_{R_i^k} & \text{if } p > 1 \end{cases} \tag{5b}$$

Eq. (5a) represents the estimated cost of performing a task by a robot before taking into account the energy requirements of the tasks in the *task list* of the robot as shown in line number 9 of Algorithm 1. Eq. (5b) represents the estimated cost of performing a task by a robot after taking into account the energy requirements of the tasks in the *task list* of the robot as shown in line number 16 of Algorithm 1. The first case, *p=1*, depicts a scenario when the *task list* of R_i contains only one task, whereas the second case, $p > 1$, depicts a scenario when there is more than one task in the *task list* of R_i.

In Eq. (5b), $\alpha, \beta, \delta \in \{0, 1\}$. For example, in the first case, *p=1*, when α takes the value of *1*, it indicates that the p^{th} task of R_i satisfies *Constraint #3*, and hence there is no need to visit the battery stockpile before executing it. $\rightarrow$ represents the movement of a robot from one location to another.

In Eq. (5b), for the case *p=1*, $t\left(loc\left(R_i\right) \rightarrow loc\left(\mathcal{BS}\right)\right) + t\left(loc\left(\mathcal{BS}\right) \rightarrow loc\left(R_i^p\right)\right)$ represents the time taken by R_i to visit the location of the battery stockpile, $loc\left(\mathcal{BS}\right)$, from its current location, $loc\left(R_i\right)$, and from the battery stockpile to the location of the p^{th} task in its *task list*, $loc\left(R_i^p\right)$.

In Eqs. (5a) and (5b), for the case $p > 1$, $t\left(loc\left(R_i\right) \rightarrow loc\left(R_i^1\right)\right)$, represents the time taken by R_i to reach the execution location of the first task in its *task list*, $loc\left(R_i^1\right)$, from its current location, $loc\left(R_i\right)$, at the time of estimating the cost of performing the first task.

Further, in Eq. (5b), the first summation term for the case $p > 1$, $\sum_{k=2}^{p}\left[\delta * \left\{t\left(loc\left(R_i^{k-1}\right) \rightarrow loc\left(R_i^k\right)\right)\right\} + (1-\delta) * \left\{t\left(loc\left(R_i^{k-1}\right) \rightarrow loc\left(\mathcal{BS}\right)\right) + t\left(loc\left(\mathcal{BS}\right) \rightarrow loc\left(R_i^k\right)\right)\right\}\right]$, represents the sum total of the times taken by the robot, R_i, to travel from the execution location of the first task in $tsk\_lst_{R_i}$ to the execution location of $\tau^j_{(n,i)}$, (p^{th} task) via all the tasks in between them in $tsk\_lst_{R_i}$. This sum also includes the time taken to visit the battery stockpile, wherever applicable. It may be noted that the value of δ may not be the same for all values of k.

In Eqs. (5a) and (5b), for the case $p > 1$, the second summation term represents the sum of the times taken by R_i to execute all the tasks that are present before $\tau^j_{(n,i)}$ in the *task list* of R_i. Here, $t_{R_i^k}$ represents the time taken by R_i to accomplish the k^{th} task in its *task list*. The time taken to execute the p^{th} task in $tsk\_lst_{R_i}$ is used to estimate the cost of performing the $(p+1)^{th}$ task in $tsk\_lst_{R_i}$. This is because the p^{th} task in $tsk\_lst_{R_i}$ contributes toward the time taken to start the execution of the $(p+1)^{th}$ task in $tsk\_lst_{R_i}$. Therefore, the upper limit of the second summation term is *(p-1)*.

The estimated bid value of a task is determined by estimating the cost of performing that task and also considering the impact of its insertion in the *task list* of a robot which results in an increase in the estimation of costs of performing the tasks that have been shifted because of it as in [12]. Eq. (6) below gives the estimated bid value offered by R_i for the p^{th} task, $\tau^j_{(n,i)}$, in its *task list*, $\mu_{R_i^p}$.

$$\mu_{R_i^p} = \lambda_{R_i^p} + \left(\sum_{k=p+1}^{|tsk\_lst_{R_i}|} \lambda_{R_i^k} - \sum_{k=p}^{|tsk\_lst_{R_i}|} \lambda_{R_i^k} \right) \tag{6}$$

Here $\lambda_{R_i^p}$ gives the estimation of the cost of performing $\tau^j_{(n,i)}$ and is calculated using Eqs. (5a) and (5b). $\sum_{k=p+1}^{|tsk\_lst_{R_i}|} \lambda_{R_i^k}$ is the summation of the estimations of the costs incurred in performing the tasks that would follow $\tau^j_{(n,i)}$ (the p^{th} task) in $tsk\_lst_{R_i}$, had $\tau^j_{(n,i)}$ been the p^{th} task. The tasks would be shifted because of the insertion of $\tau^j_{(n,i)}$ in $tsk\_lst_{R_i}$, and hence the lower limit of this summation is *(p+1)*. $\sum_{k=p}^{|tsk\_lst_{R_i}|} \lambda_{R_i^k}$ represents the summation of the estimations of the costs of performing these shifted tasks before the insertion of $\tau^j_{(n,i)}$ in $tsk\_lst_{R_i}$, and hence the lower limit of this summation is *p*. The absence of $\tau^j_{(n,i)}$ in the *task list* would move the latter tasks one position forward in $tsk\_lst_{R_i}$. The expression inside the parenthesis gives the impact of including $\tau^j_{(n,i)}$ in $tsk\_lst_{R_i}$ in terms of increasing the estimated costs (starting times) of the subsequent tasks in $tsk\_lst_{R_i}$. Using the Eqs. (5a) and (6), the estimated cost and the estimated bid value of $\tau^j_{(n,i)}$ are calculated for all its feasible insertion positions in $tsk\_lst_{R_i}$. Among all the feasible positions in $tsk\_lst_{R_i}$, the one that gives the minimum estimated bid value for a task is determined by the robot as shown in line number 11 of Algorithm 1. The *task list* is updated by inserting that task at that position. The costs and the bid values of all the tasks of J_n in $tsk\_lst_{R_i}$ are then re-estimated and the values are updated in the $bid\_lst_{R_i}$. This is done to take into account the changes introduced due to the insertions of the tasks into the *task list*.

The robot consumes energy during movement and sensing. The estimated energy $e_{R_i^p}$, required by R_i to execute a task, $\tau^j_{(n,i)}$, which is the p^{th} task in its *task list* is given by Eq. (7) below.

$$e_{R_i^p} = \sum_{k=1}^{s} e_{R_i,sensor_k} + \sum_{q=1}^{m} e_{R_i,motor_q} \tag{7}$$

where

$$\begin{aligned} e_{R_i,sensor_k} &= P_{R_i,sensor_k} * t_{R_i,sensor_k} \\ e_{R_i,motor_q} &= \int_{t=0}^{t=t_{R_i,motor_q}} P_{R_i,motor_q} dt \end{aligned} \tag{8}$$

$e_{R_i^p}$ depends mainly on the energy consumed by the sensors as represented by the first term and the energy consumed by the motors due to the movement of the robot as represented by the second term in Eq. (7). Here, s and m denote the number of sensors and the number of motors, respectively. The energy consumed by the sensors is assumed to be almost stable [42].

Eq. (8) gives the estimated energy consumed by the k^{th} sensor and the q^{th} motor of R_i for $\tau^j_{n,i}$. Here, $P_{R_i,sensor_k}$, and $P_{R_i,motor_q}$ represent the power consumed by the k^{th} sensor and the q^{th} motor of R_i for $\tau^j_{(n,i)}$, respectively. $t_{R_i,sensor_k}$ and $t_{R_i,motor_q}$ denote the periods for which the k^{th} sensor and the q^{th} motor of R_i were active during that task. This period includes the travel time of the robot from its current location to the execution location of $\tau^j_{(n,i)}$ plus the time required to execute that task. The current location would be the location of the task preceding $\tau^j_{(n,i)}$ in the *task list* of the robot in case of tasks other than the first task in its *task list*. While calculating the estimated energy, the period during which the sensors and the motors were active during that task execution is estimated by assuming that all robots move with a constant velocity. A constant velocity is used to estimate the energy consumption by the robots for executing the tasks from their *task lists* and also while estimating the energy required to visit the battery stockpile (if needed) during the task allocation process (before actual execution) to predict the costs and bid values offered by the robot for each task. Chaudhari et al. [43] mention that the energy consumption increases with an increase in linear velocity. Decreasing the linear velocity increases the time taken to reach the execution location of the task. Since the energy consumption by sensors is proportional to their operational time, the total energy consumption increases. According to them, maintaining a constant linear velocity ensures minimum energy consumption. Chaudhari et al. [43], therefore, suggest that a constant linear velocity minimizes energy consumption in a robot.

The values of the power consumption of the sensors used in the experiments were taken from the datasheets of these sensors. Power consumed by a motor was estimated based on [40]. It is given by Eq. (9) below-

$$P_{R_i,motor} = L + m_{R_i} a_{R_i} v_{R_i} + m_{R_i} v_{R_i} g f_r \tag{9}$$

where L is the transformation loss, m_{R_i}, a_{R_i}, and v_{R_i} are the mass, acceleration and linear velocity of the robot, R_i, respectively. f_r is ground friction constant and g is gravity constant.

After estimating the energy requirement of each task as shown in line number 15 of Algorithm 1, a check is done to ensure that each task satisfies *Constraint #3*. This helps to determine whether there is a requirement to visit the battery stockpile to refill the energy before the task is to be executed. If this *Constraint #3* is not satisfied for a task, the robot predicts the need to visit the battery stockpile to refill its energy before proceeding to move toward the execution location of that task. It is assumed that the robot swaps its depleted battery with a fully charged one, mimicking the battery swapping strategy as given in [44]. For every task, the estimated value of its required energy is subtracted from the total energy available in the battery of the robot to determine the energy that will be available after executing that task. The costs and the bid values of the task that does not satisfy *Constraint #3* and its subsequent tasks are re-estimated using Eq. (5b) and Eq. (6) and updated in $bid\_lst_{R_i}$ as shown in line numbers 16 and 17 of Algorithm 1. This is done to take into account the extra time that would be incurred in visiting the battery stockpile and swapping the battery before moving toward the location where the task is to be executed. The tasks comprising the job, J_n, are then removed from $tmp\_lst_{R_i}$ as shown in line number 18 of Algorithm 1.

Each robot estimates the costs, the bid values, and the energy required to execute the tasks in its *task list* independently using only its local information. A robot does not have access to the corresponding values of the other robots.

After selecting all the appropriate tasks of a job J_n, each robot sets the value of their *flag* to *job_id* (n in this case) to indicate the completion of the *task selection* stage of the task allocation algorithm for J_n as shown in line number 19 of Algorithm 1. Then the IA_n (which is responsible for bringing in the job J_n) residing inside the agent platform in one of the robots commences the *contention elimination* stage of the task allocation algorithm for J_n. The completion of the *contention elimination* stage is found by checking the value of the *flag* as shown by the second *while* loop in line number 20 of Algorithm 1. A non-zero value of the *flag* indicates that the *contention elimination* stage is in progress for the job with the *job_id* as an identifier. The *task selection* stage for the next job in $tmp\_lst_{R_i}$ starts only after the task allocation of the current job J_n is complete as indicated in line number 21 in Algorithm 1. After all the tasks of a job J_n are allocated by resolving any conflicts in the selections of its tasks among the robots, the task allocation process for the tasks of the next job commences.

4.1.2. Contention elimination stage

Since the selection of the tasks of a job is performed in a decentralized manner, there is a possibility that multiple robots could select the same task for execution during the *task selection* stage. In order to mitigate such a scenario, the elimination of the conflicting selections

of the tasks of a job is performed by the corresponding *Initialization Agent* that carries the job into the network of mobile robots. This agent operates autonomously without the need for any centralized supervision, thereby facilitating decentralized conflict resolution. This mobile agent moves into the network of robots and does the conflict resolution with minimal interference to the robots in the network. This facilitates the execution of the previously allocated tasks and the conflict resolution process of the *new job* to proceed concurrently. A mobile *Initialization Agent* is unaware of the presence of other mobile *Initialization Agents* in the network of robots.

Algorithm 2: Contention Elimination Stage of Task Allocation Algorithm for job J_n performed by IA_n

Input: *bid list* of each robot
Output: *task list* containing conflict-free allocation of tasks of job J_n

1: **for** each task $\in J_n$ **do**
2: p_t = *find_predecessor*(task)
3: **if** $p_t == 0$ **then**
4: *add_set*(*task*, ξ^n)
5: **else**
6: *add_set*(*task*, ξ'^n)
7: **end if**
8: **end for**
9: **while** $\xi^n \neq \emptyset$ **do**
10: **for** each task $x \in \xi^n$ **do**
11: Migrates from one robot to another and compares the estimated bid values offered by each robot for x to resolve conflicting selections among the robots, if any
12: Migrates to the most suitable robot that offered the least estimated bid value for x and allocates x to it
13: *update_alloc_list*() on each robot
// Line numbers 14 to 18 are called for the robots for which
// the task is deleted. IA_n migrates to the robots and performs
// these actions.
14: Removes x from the *task lists* of the robots
15: Removes the estimated bid value of x from *bid lists* of the robots
16: *estimate_enreq*() for each task in the *task lists* of the robots using Eq. (7)
17: Re-estimates costs and bid values of the tasks of job J_n in the *task lists* of the robots using Eqs. (5b) and (6)
18: *update_bid_list*()
19: **end for**
20: **for** each task $y \in \xi'^n$ **do**
21: r_t = *ready_task*(y)
22: **if** $r_t == 1$ **then**
23: *add_set*(y, ξ^n)
24: $\xi'^n \Leftarrow \xi'^n \setminus \xi^n$
25: **end if**
26: **end for**
27: **end while**
28: **for** each $R_i \in R$ **do**
29: *flag* = 0
30: **end for**

Even though a fully centralized system can produce an optimal solution in theory for task allocation, it suffers from communication and computation overheads, single-point failures, etc. On the other hand, fully distributed systems are robust to failures and reduce computation overheads, but they produce sub-optimal solutions [45]. Contention elimination and task allocation in consensus-based distributed algorithms become tricky when multiple robots have the lowest bid value for the same task [46]. Distributed systems require multiple interactions to achieve consensus among the participating entities. This results in an increase in time needed to reach a consistent task allocation and also generates a considerable amount of network traffic by transmitting large amounts of data [13]. Mobile agents reduce network traffic by reducing the number of interactions required to achieve consensus by enabling local interactions [47]. Distributed systems mostly use broadcasting for communication, which leads to poor scalability [2]. Mobile agents can be used for communication among distributed systems without the need to broadcast. Mobile agents combine the advantages of centralized and decentralized systems. They act as a good tool for task allocation as they can migrate, collect, and access all the bid values of the robots in the network. Since each *Initialization Agent* handles only its assigned job, its failure would mean that only its associated job is lost. Allocation of tasks of other jobs by other *Initialization Agents* could continue undisturbed. Mobile agents also provide some advantages over a distributed auction-based task allocation algorithm. Auction-based algorithms often require that the bid values from each agent be transmitted to the auctioneer, which limits the network topologies that can be used. To overcome this, in auction-based task allocation algorithms, only the direct neighbors within the single-hop communication range can compete in the task allocation process. Hence, such algorithms suffer from mis-assignment problems if the participating robots are heterogeneous or none of them are capable of performing the task [13,17]. Mobile agents, on the contrary, overcome this issue by engaging all the robots in the network to participate in the task allocation process so that the most suitable robot can be allocated the task. In distributed auction-based algorithms, the auctioneer waits until it has received the bid values from all the robots (or for a predefined period) [48]. This can affect the performance of the algorithm. Mobile agents, however, overcome this issue by visiting all the robots individually to collect and compare the estimated bid values provided by the robots and eventually allocate the tasks to appropriate robots.

Algorithm 2 depicts how the mobile agent, IA_n, performs the contention elimination process for a job, J_n, which it has already provided to the robots. For every job J_n, there is an IA_n which brings that job into the network of mobile robots and also allocates its constituent tasks to the robots by resolving conflicting selections of its constituent tasks, if any. These *IA*s take care of *Cross-schedule Dependencies* between the tasks of the same job while allocating tasks to different robots. Hence, they allocate the tasks of the job in such a way that *Constraint #2* is satisfied.

A task $\tau^j_{(n,h)}$ of a job J_n is said to be *ready* if all its predecessor tasks are already executed or its predecessor set, $pred\left(\tau^j_{(n,h)}\right)$, is empty indicating that the task has no predecessor. A set of all such tasks is termed here as a *ready_set* of job J_n and is denoted as ξ^n. The other tasks of J_n whose predecessor tasks have not finished their execution fall under the *non_ready_set* denoted as ξ'^n.

Line numbers 1 to 8 of Algorithm 2 are executed only once at the beginning to generate the initial *ready_set*. IA_n finds the predecessor task(s) of each task from its predecessor set using the *find_predecessor* function in line number 2 of Algorithm 2. The tasks are added to the *ready_set* and *non_ready_set* using the *add_set* function. A task with no predecessor is an independent task. Therefore, initially, all independent tasks of a job are allocated to the robots, followed by those whose predecessors are already allocated to take care of the dependencies imposed by precedence constraints, and hence, satisfy *Constraint #2*. IA_n migrates into the network of mobile robots to resolve the conflicting selections of the tasks of J_n. It moves from one robot to another, comparing the estimated bid values for each task of J_n, after making sure that the robot has finished selecting the compatible tasks of J_n. When it lands on the very first robot, it collects the estimated bid value offered by the robot for a task and migrates to the next robot, where it compares the estimated bid value provided by this robot with that of the previous one. It discards the higher bid value and thereby always carries the minimum bid value for the task as its payload. It migrates from one robot to another in the network and performs this action, effectively performing a distributed consensus algorithm. In a consensus-based algorithm, the bid values are exchanged between the neighboring robots, and their comparisons are performed by each of the robots to resolve conflicting task selections and to reach a consensus. In the *proposed* algorithm, too, the comparison of the estimated bid values is performed locally within each of the robots by the *Initialization Agent*. The bid value carried by the *Initialization Agent* after visiting all the

other robots, is compared with the bid value provided by the robot on which it lands at the end to decide the robot that will be allocated the task. The *Initialization Agent* can land up in any one of the robots at the end. Each *Initialization Agent* follows its own path while migrating from one robot to another. Therefore, the path followed by each of them during the distribution of jobs and the *contention elimination* stage need not necessarily be the same. It is possible that these agents eventually land up in different robots at the end. Also, there is no central node or fixed robot that resolves the conflicts and allocates the tasks of all the jobs.

The following rules that aid in resolving the conflicting selections of the tasks of J_n among multiple robots for independent tasks of a job are stated below:

1. If the estimated bid values of a task provided by all the robots are different, the robot that provides the least estimated bid value among all is allocated the task.
2. If the estimated bid values of a task provided by all the robots are not different and the same lowest estimated bid value is offered by more than one robot, then the robot quoting the least estimated cost (given in terms of time taken to reach the execution location of the task and start its execution) is allotted that task.
3. If both the estimated bid values and the estimated costs offered by these robots are the same, then the mobile agent allocates this task to that robot that it encountered first while resolving the conflict for this task.

In case of a dependent task $\tau^j_{(n,h)}$, the associated IA_n first checks whether the estimated time taken by a robot (which is competing for this task) to reach the location of $\tau^j_{(n,h)}$ is greater than the estimated time(s) taken by a robot(s) that is allocated the predecessor task(s) of $\tau^j_{(n,h)}$ to reach the location(s) of its predecessor task(s). This takes care of the dependencies due to precedence constraints among the tasks of the same job. This also prevents an idle waiting period for the robot that is allocated the dependent task, $\tau^j_{(n,h)}$, whose predecessor task is not yet over, and hence conserves energy. If this condition is satisfied, then the same rules as in the case of independent tasks of a job apply to the dependent tasks as well. In case none of the robots satisfy this condition, $\tau^j_{(n,h)}$ is given to that robot, which gives the least estimated bid value in case the estimated bid values are different. In case the estimated bid values are similar, whereas the estimated costs are different, the task is allotted to the robot that takes the maximum time among all the robots vying for that task. This minimizes the idle waiting period for the robot and hence saves energy. In case the estimated costs and the estimated bid values are the same, then a similar approach as given in point number *3* given above is employed.

For each task of the job, J_n in the *ready_set*, steps 11 to 18 of Algorithm 2, are performed by IA_n, where step 13 is performed on every robot in the network. After allocating a task to the most suitable robot, the *allocation lists* of all the robots are updated with the mapping information as shown in line number 13 of Algorithm 2.

After resolving the conflicting selections of a task and allocating it to a robot, IA_n removes that task and its associated estimated cost and bid value from the *task lists* and the *bid lists* of the robots that quoted a higher estimated bid value for the task by migrating to the robots. After that, the energy requirements of the tasks in the *task lists* of these robots are again estimated, and the tasks that do not satisfy *Constraint #3* are noted to predict the requirement of visiting the battery stockpile before proceeding with the executions of such tasks. The costs and the bid values of all the tasks of J_n in the *task lists* of these robots are then re-estimated using Eq. (5b) and Eq. (6), and the values are updated in their *bid lists*. This is done to take into account the changes introduced due to the removal of the task from the *task lists* of these robots and the additional time that would be consumed by the robots for visiting the battery stockpile before moving toward the execution locations of the tasks in their *task lists* that did not satisfy *Constraint #3*. After all the tasks of the current *ready_set* are processed, the next set of tasks that are eligible to be included in the *ready_set* is determined by executing the *ready_task* function in line number 21 of Algorithm 2. These eligible tasks are added to the *ready_set* and are removed from the *non_ready_set*. The steps in line numbers 9 to 27 of the Algorithm 2 are executed until all the tasks of the job J_n are allocated to the robots. The value of the *flag* is reset to 0 to allow the commencement of the task allocation process of the next job present in the *temporary lists* of the robots. The mobile agent, IA_n, being autonomous, self-destructs after resolving all the conflicting selections of the tasks of J_n.

Algorithm 3: Handling Unexpected Delays during Actual Execution by robot R_i

Input: $tsk\_lst_{R_i}$ of R_i
Output: Ready task from $tsk\_lst_{R_i}$

```
1: while tsk_lst_{R_i} is not empty do
2:   r_t = ready_task(tsk_lst_{R_i^1})
3:   if r_t == 0 then
4:     if length_Of(tsk_lst_{R_i}) ≥ 2 then
5:       for j ← 2 to n do
6:         c_k = check_task(tsk_lst_{R_i^j})
7:         if c_k == 1 then
8:           distance = calc_dist(tsk_lst_{R_i^j})
9:           if distance ≤ η then
10:            for t ← 1 to t_1 do
11:              r_t = ready_task(tsk_lst_{R_i^1})
12:              if r_t == 1 then
13:                return tsk_lst_{R_i^1}
14:              else
15:                wait for 1 second
16:              end if
17:            end for
18:          else
19:            for t ← 1 to t_2 do
20:              r_t = ready_task(tsk_lst_{R_i^1})
21:              if r_t == 1 then
22:                return tsk_lst_{R_i^1}
23:              else
24:                wait for 1 second
25:              end if
26:            end for
27:          end if
28:          return tsk_lst_{R_i^j}
29:        end if
30:      end for
31:    end if
32:  end if
33: end while
```

Auction-based MRTA algorithms use market-based approaches and are decentralized and distributed in nature [41,48]. In an auction-based algorithm, any robot that discovers a task becomes an auctioneer and announces it to all the other robots. The other robots then place bids, and the auctioneer determines the winner after examining these bids [40,48–50]. The works reported in [48–50] have not considered the energy requirements of the tasks during the bid generation process, while the works described in [40,49], ignore tasks with precedence constraints.

In the *proposed* algorithm, a mobile *Initialization Agent* is responsible for distributing the job it is carrying to all the robots in the network. It carries the code for contention elimination. Therefore, this code is not resident in any of the robots or any central node. Since all the robots are in a fully connected network, it can migrate from one robot to another in the network in any order to collect and compare the estimated bid values for a task, to resolve conflicts, if any, and then

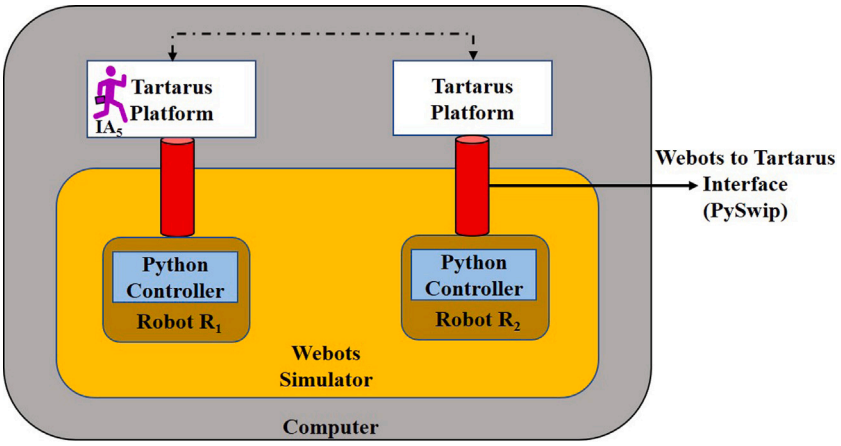


Fig. 3. Communication between the robots inside a *Webots* simulator and the associated *Tartarus* platforms.

allocate the task to a robot that provided the minimum bid value in a manner similar to a distributed auction algorithm for task allocation described in [40,49,50]. The *Initialization Agents* also tend to perform multi-robot task allocation in a decentralized manner as in [48]. This mobile *Initialization Agent* acts like a robot in [40,49,50] wherein the robot finds and facilitates the auction of tasks in a network of robots in a distributed manner. Just like the auctioneering robot in [40,49,50] differ depending on which of them discovers the task to be auctioned, in the *proposed* algorithm too, there are different mobile *Initialization Agents* for different jobs.

4.2. Handling unexpected delays during actual execution

The robots encounter unanticipated delays during actual execution due to obstacles, congestion caused by multiple robots moving in the same area at the same time, etc. They execute the *obstacle avoidance* algorithm to avoid the obstacles and the congestion. Consider a scenario where a robot, R_i might reach the location of a task $\tau^j_{(n,h)}$, $loc\left(\tau^j_{(n,h)}\right)$, whose predecessor tasks (allocated to different robots) might not have been executed due to obstacles and congestion or because of the reason mentioned in the previous subsection (*contention elimination* stage). As a result, R_i cannot perform $\tau^j_{(n,h)}$ and is forced to wait, which leads to idling. This situation not only affects other tasks that come after $\tau^j_{(n,h)}$ in $tsk\_lst_{R_i}$, but also the ones that are dependent on $\tau^j_{(n,h)}$ and allocated to different robots. The proposed task allocation algorithm incorporates a strategy to determine the course of action of a robot under such unexpected scenarios during actual execution, and hence prevents delays and reduces energy consumption during the idling period. This is outlined in Algorithm 3. This algorithm runs individually on all robots.

Each robot R_i takes the first task from its *task list* denoted as $tsk\_lst^1_{R_i}$ and checks whether it is *ready* for execution. The *ready_task* function takes the first task from $tsk\_lst_{R_i}$ as an input and returns 1 if that task is *ready*, and 0, otherwise as illustrated by line number 2 in Algorithm 3. If the task is *ready*, the robot checks its energy level. It re-estimates the energy required for that *ready* task, and also that is required to visit the battery stockpile (if needed) from the execution location of this *ready* task. It uses a constant velocity to estimate these energy requirements using the Eqs. (7) to (9). If the robot has sufficient energy to satisfy *Constraint #3* for the *ready* task, it proceeds with its execution. Otherwise, it visits the battery stockpile to refill its energy by swapping its battery with a fully charged one before proceeding with that task. When the first task in $tsk\_lst_{R_i}$ is not *ready*, the concerned R_i searches for a suitable *candidate task* in $tsk\_lst_{R_i}$ to handle this situation. A *candidate task* is the one that the robot executes when the first task in its *task list* is not *ready*. Every *candidate task* must be *ready* and satisfy *Constraint #3*. The *length_Of* function in line number 4 of Algorithm 3 checks the number of tasks in $tsk\_lst_{R_i}$. If the length of $tsk\_lst_{R_i}$ is less than 2, the robot waits for the first task in $tsk\_lst_{R_i}$ to become *ready*. Else, a suitable *candidate task* among the tasks in $tsk\_lst_{R_i}$ is identified using the *check_task* function in line number 6 of Algorithm 3. The loop in line number 5 of Algorithm 3 runs from 2 to n because the robot searches all the tasks starting from the second task up to the end of its *task list*. In case R_i does not find a suitable *candidate task*, the robot waits for the first task to become *ready*.

However, if R_i finds a *candidate task*, it waits for the first task in $tsk\_lst_{R_i}$ (which was not *ready*) to become *ready*. The time duration for which the robot waits depends on the distance between the execution location of the *candidate task* and the current location of the robot, which is the execution location of the *non-ready task*. The *calc_dist* function in line number 8 of Algorithm 3 calculates this distance. If the distance is less than or equal to a threshold (η), i.e., if the *candidate task* is near the current location of R_i, it decides to wait for a short period, t_1 seconds as shown in line number 10 of Algorithm 3. Otherwise, it waits for a longer period, t_2 seconds, as shown in line number 19 of Algorithm 3. During this waiting period, R_i continuously checks the status of the first task in its *task list*. If the first task becomes *ready* during the waiting period, R_i proceeds with its execution. On the contrary, if the first task is still not *ready* after the waiting period is over, R_i creates a mobile agent termed the *Helper Agent* and adds this *non-ready* first task from its *task list* as a payload to the *Helper Agent* to reallocate that task to a suitable robot, if any. Then, R_i proceeds with the execution of the *candidate task*. The *Helper Agent* migrates through the network of robots and searches for a compatible robot that is either idle (does not contain any task in its *task list*) or which contains not more than two tasks in its *task list*. Once it finds such a robot, it generates both the estimated cost and the estimated bid value for the task on behalf of that robot by finding a suitable place in the *task list* of that robot where the task could be inserted. It considers the energy requirements of the tasks and the energy available in the robot while generating the estimated costs and the estimated bid values. The *Helper Agent* carries this estimated cost and estimated bid value as a payload and moves into the network of robots to search for other suitable robots. In this way, it moves from one robot to the other in the network, compares the estimated bid values to eventually find the most suitable robot for the task, and allocates the task to it by inserting the task into the appropriate position in the *task list* of that robot. The *Helper Agent* also inserts the corresponding estimated cost and the estimated bid value into the *bid list* of that robot. The *Helper Agent* updates the *allocation lists* of all the robots in the network. It also removes this task and its corresponding estimated cost and estimated bid value from $tsk\_lst_{R_i}$ and the $bid\_lst_{R_i}$ of R_i (robot to which the task was previously allocated), respectively. If the *Helper Agent* is not able to find any suitable, compatible robot, the task is not reallocated to any other robot.

Fig. 4. Arena in the *Webots* simulator and Precedence Graph of J_n.

Every time a robot completes a task execution, it calculates the energy required to execute that task and subtracts this value from its available energy. It also deletes that task from its *task list* and starts inspecting the first task from its *task list* to check whether it is *ready* for execution. If, after the execution of a task, a robot finds the successor set of the task to be non-empty, it intimates this completion to other robots that have been allocated the successor tasks via the network. The *allocation list* of robot R_i, $alloc\_lst_{R_i}$, aids it in finding such robots.

5. Experiments and results

This section describes the experimental setup and results of the experiments conducted to validate the *proposed* algorithm. The experiments were conducted using *Webots* [51], an open-source robot simulator. We have also used *Tartarus* [39], a multi-agent platform written in *SWI-Prolog* [52], to aid in the creation and movement of mobile agents amongst the platforms. The controllers of the robots within *Webots* were programmed using *Python*. Two robots R_1 and R_2 within a *Webots* simulator, each having a *Python* controller and an associated *Tartarus* platform is portrayed by Fig. 3. The bi-directional arrow indicates the directions in which the *Initialization Agent*, IA_5, can move. To facilitate communication between the *Python-based Webots* controllers and the mobile agents residing in the *SWI-Prolog-based Tartarus* platforms, the *Webots to Tartarus Interface* described in [53] was used. This *interface* forms a bridge between the robot controllers and the mobile agents hosted by the *Tartarus* platforms. The *interface* uses a function called *PySwip* [54] to facilitate querying the *SWI-Prolog-based Tartarus* platform from the *Python-based robot controllers* within *Webots*. The *interface* converts all *Python* queries into corresponding *Tartarus* commands to aid in programming and control of the agents from within *Webots*. It allows the programmers to include the agent programming paradigm into *Python-based controller codes* in *Webots*.

5.1. Experimental setup

In our implementation, we have used five e-puck robots within one *Webots* instantiation running on a personal computer. In addition to a dedicated controller, each e-puck robot was also associated with dedicated *Tartarus* platforms running on the same computing system. These five platforms thus provided for the movement of the mobile agents from one e-puck robot to another, thereby aiding in the transfer of information, both onto the associated robot and also amongst the robots, via the *Webots to Tartarus Interface* [53]. Since *Tartarus* platforms are known to run on real robots [21], the simulated scenario used herein can be realized in the real world too.

Fig. 4 shows the experimental setup used. As can be observed, it consists of a rectangular arena of size 1.5 m x 1.5 m containing five e-puck robots and thirty objects of different shapes, sizes, and colors placed in ten randomly located clusters in the arena (Clusters have been marked using an oval shape). The objects are spatially distributed inside the arena. The light brown-colored cylindrical object at the top right corner of the arena represents the battery stockpile where the robots that need to recharge visit and swap their depleted batteries with readily available fully charged ones. Each robot was equipped with 8 IR sensors, a GPS sensor, a compass sensor, and a camera. The 8 IR sensors were used to avoid obstacles. The camera was used to detect the color of the objects, while the GPS sensor provided the location of the robot within the arena. The compass sensor was used to determine the direction (heading) of the robot.

We chose a scenario that depicts cluster inspection jobs wherein a single cluster inspection job refers to the inspection of individual objects in a particular order inside the cluster. Here, inspection refers to the retrieval of the properties of the objects, which include their color, height, radius, etc., and examining them. In the experiments conducted, we have considered three types of tasks, viz., (i) inspection of conical-shaped objects, (ii) inspection of square-shaped objects, and (iii) inspection of rectangular-shaped objects. The inspection of the conical, square, and rectangular objects present inside the arena was carried out by five autonomous e-puck robots equipped with the required sensors. During the task execution process, a robot navigates to the vicinity of the location of an object (up to a distance of 0.7 cm from the object) in a cluster, detects its color, and collects data regarding its properties, such as height and radius (radius data is collected in case of a conical object). If the height and the radius of the object do not match a given specification based on its color, the color of the object is changed to black, indicating it to be faulty. Otherwise, the color of the object remains unchanged. After all the objects inside the cluster are inspected, the corresponding cluster inspection job is

Table 1
Description of the heterogeneity of the five e-puck robots used in the experiments.

Robot(s)	Capability of robot
R1, R5	Can cater to conical-shaped objects
R2, R4	Can cater to square-shaped objects
R3, R5	Can cater to rectangular-shaped objects

complete. The camera is disabled when not in use, i.e., while the robot is navigating. After it reaches the location where the task is to be executed, the camera is enabled to detect the color of the object. This helps to minimize the energy consumption of the robot.

Several cluster inspection jobs comprising combinations of these tasks were then provided to test the efficacy of the proposed task allocation algorithm. In Fig. 4, 10 such cluster inspection jobs indicated with ten brown-colored ovals can be seen. The above-mentioned setup consisting of 10 homogeneous inspection jobs is randomly chosen. The *proposed* algorithm will show the same performance for heterogeneous jobs as well. There are dependencies between the tasks of a job. Fig. 4 shows the precedence graph with six tasks for a cluster inspection job J_n, where the cluster is depicted by the brown-colored oval at the upper right-hand side of the arena. In the precedence graph, $\tau^1_{(n,1)}$ corresponds to the inspection of the conical-shaped object marked as 1, as shown in the arena in Fig. 4. $\tau^1_{(n,2)}$, $\tau^2_{(n,3)}$, $\tau^2_{(n,4)}$, $\tau^3_{(n,5)}$, and $\tau^3_{(n,6)}$ correspond to the inspection of the objects marked as 2, 3, 4, 5, and 6, respectively. The arrows in the graph give the precedence relationship between a pair of tasks. Hence, $\tau^1_{(n,1)}$ has to be executed before $\tau^1_{(n,2)}$ and $\tau^2_{(n,3)}$, $\tau^1_{(n,2)}$ has to be executed before $\tau^2_{(n,4)}$ and $\tau^2_{(n,3)}$ has to be executed before $\tau^3_{(n,5)}$ and $\tau^3_{(n,6)}$. However, $\tau^2_{(n,3)}$ and $\tau^2_{(n,4)}$ are not connected by an arrow, hence they can be executed concurrently or in any order. Similarly, $\tau^1_{(n,2)}$, $\tau^3_{(n,5)}$, and $\tau^3_{(n,6)}$ can be performed concurrently or in any order. The cluster depicted by the brown-colored oval at the lower left-hand side of the arena represents another cluster inspection job with independent tasks, i.e., all three tasks could be executed either concurrently or in any order. This job J_n is carried as a payload by the corresponding *Initialization Agent*, IA_n, into the network of these five e-puck robots. IA_n migrates into this network of robots and provides all the information regarding J_n to these five robots. After each robot selects the compatible tasks from within J_n, IA_n migrates from one e-puck robot to the other via their corresponding *Tartarus* platforms. While doing so, it compares the estimated bid values offered by these robots for each task within J_n and then allocates the tasks to the robots that provide the corresponding minimum bid values. It also removes the unallocated tasks of this job from the *task lists* of the robots. After it has resolved the conflicts and allocated all the tasks of J_n, it kills itself. This self-destruction ensures that there are no useless agents occupying space in the network of robots.

To ensure that the *proposed* algorithm can accommodate both homogeneous and heterogeneous robot types, we have categorized the e-puck robots as homogeneous and heterogeneous by programming their behavior. Homogeneous robots are the ones that have the same capabilities and are behaviorally the same, whereas heterogeneous robots have different capabilities and have different behavioral characteristics.

Experiments were conducted with both homogeneous and heterogeneous robots. The objects in the arena are different from each other in terms of their geometrical shapes. Accordingly, the robots were also designed to have heterogeneous capabilities. We have defined robot heterogeneity in terms of its capability of catering to objects of different geometrical shapes, as shown in Table 1. In case all the robots are homogeneous, then all of them can cater to all the objects irrespective of their geometrical shapes.

The experiments conducted were used to study decentralized and distributed dynamic task allocation scenarios with jobs arriving *on-the-fly* into the network of mobile robots, where a group of robots participates in the task allocation problem. The precedence graphs of these jobs consist of either 2, 3, 5, or 6 nodes (tasks). There is no dependency between the tasks of different jobs.

To compare the performance of the *proposed* algorithm, we implemented two other distributed task allocation algorithms, viz., *reassignment* algorithm [10] and *partial reassignment* algorithm [11]. Unlike the *proposed* algorithm, these two algorithms do not consider tasks with dependencies and deal with individual tasks rather than jobs consisting of a set of tasks. In addition, they do not take into account the energy requirement of a task and the energy available in a robot while generating estimated costs and estimated bid values at the time of the task allocation process and also during actual execution in contrast to the *proposed* algorithm. In *reassignment* algorithm, whenever a *new task* arrives, all tasks barring the *protected* tasks in the *task list* of each of the robots become eligible for reallocation. *Protected* tasks are those that have already been executed or are currently being executed. In the case of *partial reassignment* algorithm, the arrival of a *new task* causes the robots to release only a part of their *task lists* for reallocation, excluding the *protected* tasks. The number of tasks to be released by each robot herein was determined based on the equation given below [11].

$$\sigma_{R_i} = \left|tsk\_lst_{R_i}\right| - 0.8\left(\frac{\sum_{j=1}^{|n_r|}\left|tsk\_lst_{R_j}\right|}{|n_r|}\right) \tag{10}$$

where σ_{R_i} in (10) refers to the number of tasks released by robot R_i and n_r represents number of participating robots in the reallocation process. It indicates that each robot releases tasks according to its workload and the average workload of all the participating robots.

We conducted the following experiments to validate the performance of the *proposed* algorithm and compared the results of the *proposed* algorithm with that of the *reassignment* and *partial reassignment* algorithms. We have used some symbols below to represent the three cases where $\not\mathbb{D}$ denotes independent tasks (without dependency), $\mathbb{D}$ represents dependent tasks, $\not\mathbb{E}$ represents a case where energy is not taken into consideration, and $\mathbb{E}$ represents a case where energy is taken into consideration.

- **Case #1 $\not\mathbb{D}\not\mathbb{E}$:** In this set of experiments, the *proposed* algorithm dealt with individual tasks with no dependencies among them instead of jobs and ignored energy consumption similar to the other two algorithms. Therefore, in this set of experiments, all three algorithms (the *proposed*, the *reassignment*, and the *partial reassignment* algorithms) handled independent tasks and disregarded energy consumption. New tasks were injected into the network of mobile robots every 20 s. For this set of experiments, a task corresponds to an inspection of an object in the arena as described before and tasks are not grouped together as clusters here.
- **Case #2 $\not\mathbb{D}\mathbb{D}\not\mathbb{E}$:** In this set of experiments, the *reassignment* and the *partial reassignment* algorithms were augmented with jobs. A job consisted of either all independent tasks or a combination of independent and dependent tasks similar to the *proposed* algorithm. The energy requirements of the tasks and the energy available in the robots were not considered in the case of all three algorithms as in *Case #1*. Therefore, in this set of experiments, all three algorithms dealt with jobs and ignored energy consumption. New jobs were injected into the network of robots every 10 s.
- **Case #3 $\not\mathbb{D}\mathbb{D}\mathbb{E}$:** In this set of experiments too jobs were considered for all three algorithms as in *Case #2*. The available energy in a robot and the energy needed for the tasks were considered while generating the estimated costs and the estimated bid values of the tasks at the time of the task allocation process and also during actual execution for the *proposed* algorithm only. The other two algorithms did not consider the energy requirements while generating the estimated costs and the estimated bid values during the task allocation process but took into account the

Table 2
Comparison of *Average waiting time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #1* with κ kept fixed at 1:4.

Type of robot	No. of robots	No. of tasks	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Average waiting time (s)	Average waiting time (s)	Average waiting time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	8	83.75	80.87	57.00	31.94	29.51
	3	12	108.33	92.75	61.25	43.45	33.96
	4	16	106.56	85.62	57.58	45.96	32.74
	5	20	111.70	83.00	52.10	53.35	37.22

Table 3
Comparison of *Total task allocation time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #1* with κ kept fixed at 1:4.

Type of robot	No. of robots	No. of tasks	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total task allocation time (s)	Total task allocation time (s)	Total task allocation time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	8	153.66	148.66	146.00	4.98	1.79
	3	12	244.00	238.00	227.00	6.96	4.62
	4	16	330.00	323.33	308.33	6.56	4.63
	5	20	412.33	408.33	389.66	5.49	4.57

Table 4
Comparison of *Total execution time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #1* with κ kept fixed at 1:4.

Type of robot	No. of robots	No. of tasks	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total execution time (s)	Total execution time (s)	Total execution time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	8	243.20	232.66	208.20	14.39	10.51
	3	12	340.20	323.00	313.00	7.99	3.09
	4	16	395.66	376.00	368.80	6.78	1.91
	5	20	519.00	439.33	410.20	20.96	6.63

energy requirements of the tasks and the energy available in the robots during actual execution for this set of experiments. New jobs in this set of experiments too were injected into the network of robots every 10 s.

For the experiments where energy consumption was considered, each robot was initially endowed with 100 units of energy. After executing a task, the amount of energy that was required to execute it was calculated by a robot using Eqs. (7) to (9). This calculated value was then subtracted from the available energy in the robot. However, it may be noted that in the real world, where robots are equipped with energy sensors, the remaining energy in a robot could directly be obtained from such devices without the use of Eqs. (7) to (9). In all our experiments, the robots were made to move with a constant linear velocity to enable minimum energy consumption as suggested in [43]. Minimizing rapid acceleration and deceleration can reduce energy consumption [55]. However, it is important to note that the velocities of the robots could vary during actual execution on encountering obstacles while moving. Also, the velocity of the robot could vary depending on the charge present in its battery. This is because when the charge present in the battery of the robot is low, it is not able to deliver the requisite amount of current to maintain the same velocity. The values of the hyperparameters η, t_1, and t_2 used in Algorithm 3 were taken as 0.7, 30 s, and 60 s, respectively.

We have defined ϑ as the ratio of n_r, the number of robots to n_j, the number of jobs. κ is defined as the ratio of n_r and n_τ where n_τ represents number of tasks. *72* experiments were conducted to validate the effectiveness of the proposed MRTA algorithm. Each of the experiments is repeated *5* times and the average of the results is reported.

We define a few terms used in portraying the results of the experiments conducted:

- **Average Waiting Time**: The *average waiting time* of an experiment is the average of the waiting times of the tasks in that experiment. The *waiting time* of a task is the time difference between the arrival time of the task into the network of mobile robots and the time when a robot starts the execution of that task.
- **Total Task Allocation Time**: The *total task allocation time* of an experiment is the time difference between the time of arrival of the first task into the network of robots and the time instant at which the allocation of the last task of that experiment is completed. The task allocation time is the sum of the times taken by the robots to select that task and the time taken by the corresponding *Initialization Agent* to resolve all conflicting selections of that task amongst the associated robots and allocate that task.
- **Total Job Allocation Time**: The *total job allocation time* of an experiment is the time difference between the time of arrival of the first job into the network of robots and the time instant at which the allocation of the last job of that experiment is completed. The job allocation time is the sum of the times taken by the robots to select the appropriate tasks of that job and the time taken by the corresponding *Initialization Agent* to resolve all conflicting selections of these tasks amongst the associated robots and allocate the tasks of that job.
- **Total Execution Time**: The *total execution time* of *n* jobs in an experiment is the time difference between the time of arrival of the first job into the network of robots and the instant of time at which all the jobs of that experiment were executed. In the current context, the arrival of a job means the instant when the *Initialization Agent* enters the network of robots.
- **Energy Refill Time**: The *energy refill time* of an experiment is the sum of the times taken by all the robots to travel from their current locations after executions of their respective tasks to the battery stockpile, swap their batteries with fully charged ones, and then move on to reach the locations of the next task in their *task lists*. Multiple robots could go toward the battery stockpile

Table 5
Comparison of *Average waiting time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #2* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Average waiting time (s)	Average waiting time (s)	Average waiting time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	223.43	107.40	95.50	57.25	11.08
	3	6	247.75	121.73	99.13	59.98	18.56
	4	8	224.68	168.90	116.30	48.23	31.14
	5	10	506.96	200.06	119.27	76.47	40.38
Heterogeneous Robots	2	4	269.72	169.66	156.32	42.04	7.86
	3	6	279.80	183.81	163.20	41.67	11.21
	4	8	339.68	202.78	175.36	48.37	13.52
	5	10	459.33	242.74	193.84	57.79	20.14

Table 6
Comparison of *Total job allocation time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #2* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total job allocation time (s)	Total job allocation time (s)	Total job allocation time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	38.00	36.33	33.66	11.42	7.34
	3	6	65.66	61.66	57.00	13.18	7.55
	4	8	100.33	91.00	84.00	16.27	7.69
	5	10	217.00	143.00	113.66	47.62	20.51
Heterogeneous Robots	2	4	37.33	35.00	31.33	16.07	10.48
	3	6	66.35	63.33	55.66	16.11	12.11
	4	8	110.33	95.66	80.66	26.89	15.68
	5	10	182.66	138.66	110.00	39.77	20.66

Table 7
Comparison of *Total execution time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #2* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total execution time (s)	Total execution time (s)	Total execution time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	372.33	218.66	177.00	52.46	19.05
	3	6	519.00	257.00	245.66	52.66	4.41
	4	8	523.66	438.33	308.33	41.12	29.65
	5	10	1099.00	572.33	356.00	67.60	37.79
Heterogeneous Robots	2	4	561.66	318.66	298.00	46.94	6.48
	3	6	693.00	405.33	313.33	54.78	22.69
	4	8	776.21	524.33	412.33	46.87	21.36
	5	10	933.75	640.89	474.65	49.16	25.93

simultaneously to refill their energies. In such a case, since there is an overlap in the travel times, the difference between the minimum of the start times and the maximum of the finish times of the travels is considered while calculating the *energy refill time* of that experiment.

- **Number of Battery Swaps**: This indicates the number of times the robots visit the battery stockpile to swap their batteries with fully charged ones during an experiment.

5.2. Results

This subsection presents a comparison of the results obtained by running the *proposed* algorithm with that of the *reassignment* [10] and the *partial reassignment* [11] algorithms. These comparisons include *average waiting time*, *total task allocation time*, *total job allocation time*, *total execution time*, *energy refill time*, and *number of battery swaps*. Different test conditions were generated by varying the number of robots, the number of jobs, and the tasks within each job. The order in which the jobs were released into the network of mobile robots was kept identical across the columns for each of the rows in all the tables to avoid any possible bias in the results.

The analysis and the results of the experiments conducted in different cases, follow.

Case #1: $\mathbb{DE}$

Table 2 portrays the *average waiting times*, Table 3 portrays the *total task allocation times*, and Table 4 portrays the *total execution times* of the proposed, the *reassignment* and the *partial reassignment* algorithms. The *proposed* algorithm outperformed the other two algorithms with respect to *average waiting time*, *total task allocation time*, and *total execution time* in all the experiments as portrayed in Tables 2, 3, and 4, respectively. The last columns in each of these tables represent the % reduction in these times achieved by the *proposed* algorithm when compared to the other two algorithms (the % reduction in time is calculated because the *proposed* algorithm aims to minimize these times). This may be attributed to the fact that in the *proposed* algorithm, whenever a *new task* was injected into the network of mobile robots, only this task was allocated while the previous allocations remained undisturbed. On the contrary, a part or all, except for the *protected* tasks of the previous allocations, became eligible for reallocation upon the arrival of a *new task* in case of the other two algorithms and were reallocated along with the *new task*. This increased the number of tasks to be allocated because new tasks could arrive before the previous ones are allocated and executed if the time interval between the arrival of tasks

Table 8
Comparison of *Average waiting time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Average waiting time (s)	Average waiting time (s)	Average waiting time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	288.33	109.16	90.70	68.54	16.91
	3	6	365.07	114.80	93.09	74.50	18.91
	4	8	573.18	163.20	112.04	80.45	31.34
	5	10	742.90	230.36	154.53	79.19	32.91
Heterogeneous Robots	2	4	216.10	172.83	150.60	30.31	12.86
	3	6	255.57	206.79	154.48	39.55	25.29
	4	8	365.31	211.76	181.10	50.42	14.47
	5	10	505.83	284.53	265.77	47.45	6.59

Table 9
Comparison of *Total job allocation time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total job allocation time (s)	Total job allocation time (s)	Total job allocation time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	40.00	36.66	34.00	15.00	7.25
	3	6	63.00	61.00	53.00	15.87	13.11
	4	8	110.00	103.33	86.00	21.81	16.77
	5	10	185.33	153.00	114.33	38.31	25.27
Heterogeneous Robots	2	4	42.66	35.66	32.00	24.98	10.26
	3	6	64.33	59.33	55.33	13.99	6.74
	4	8	114.00	93.66	81.00	28.94	13.51
	5	10	180.00	146.00	114.00	36.66	21.91

into the network of robots is short. This results in an increase in the time taken by each robot to select the appropriate tasks and the time taken to reach a conflict-free allocation, consequently increasing the *total task allocation times* of the experiments that used the other two algorithms. It also resulted in delayed executions of both the tasks that had arrived earlier and also the new ones, thus increasing the waiting times culminating in increased *average waiting times* and *total execution times* when the *reassignment* and the *partial reassignment* algorithms were used. This is portrayed by the results in Tables 2, 3 and 4.

From Table 2, it may be observed that the *average waiting time* does not always increase with an increase in the number of tasks, as observed in the case when the number of tasks increases from 12 to 20 in the case of the proposed and the *partial reassignment* algorithms and from 12 to 16 in the case of the *reassignment* algorithm. This is mainly because the *average waiting time* of an experiment is dependent on the individual waiting times of the tasks in that experiment. The waiting time of a task depends on the execution location of the task, the location of the robot that has been allocated the task, and the number of tasks the concerned robot has to execute before reaching the execution location of that task. In an experiment, a robot encountering an obstacle on the way while navigating toward the location where a task is to be executed or congestion due to multiple robots moving at the same time in the same area also adds to the waiting times of the task and also all its subsequent tasks in the *task list* of the robot. This increases the *average waiting time* of that experiment. It is thus possible that, in an experiment with a lesser number of tasks, the *average waiting time* is higher as compared to an experiment containing a greater number of tasks than the former.

It can be observed from Table 3 that the *total task allocation time* increases with an increase in the number of tasks. The *total execution time* too increases as the number of tasks increases, as can be observed from Table 4. This is because the *total task allocation time* and the *total execution time* of an experiment depend on the number of tasks that need to be allocated and executed. The more the number of tasks in an experiment, more is the time taken to allocate the tasks to the robots and to complete the executions of all the tasks. As each *new task* is injected into the network of robots after a particular time interval, the difference in the arrival times of the tasks in an experiment also contributes to the *total task allocation time* and the *total execution time* of that experiment.

Case #2: $\mathbb{DDE}$

Table 5 portrays the *average waiting times*, Table 6 portrays the *total job allocation times*, and Table 7 portrays the *total execution times* of the *proposed*, the *partial reassignment* and the *reassignment* algorithms, where the last columns in each table represent the % reduction in these times achieved by the *proposed* algorithm when compared to the other two algorithms.

The *proposed* algorithm outperformed the other two algorithms with respect to *average waiting time*, *total job allocation time* and *total execution time* in all the experiments as portrayed in Tables 5, 6, and 7, respectively, because of similar reasons as delineated in *Case #1*. In addition to these reasons, for the other two algorithms, upon the arrival of a *new job* into the network of robots, tasks of prior jobs whose predecessors need to be reallocated have to wait before starting their executions till the allocation and subsequent execution of their predecessors are complete. This increases the waiting times of tasks of the old jobs in an experiment, subsequently increasing the *average waiting time* and the *total execution time* of that experiment. On the contrary, the *proposed* algorithm avoids reallocations on *new job* arrivals, hence preventing such additional waiting periods.

Table 5 indicates that the *average waiting time* does not always exhibit an increase with a rise in the number of jobs as observed in the case of the *reassignment* algorithm when the robots are homogeneous, and the number of jobs increases from 6 to 8 because of similar reasons as outlined in *Case #1*. Besides, a delay in the execution of a task in an experiment due to an obstacle or congestion in an area not only affects the subsequent tasks of that task in the *task list* of the robot (as mentioned in *Case #1*) but also its successor tasks that are allocated to other robots. This results in an increase in the *average waiting time* of that experiment. Therefore, the *average waiting time* of an experiment may exceed that of another despite having fewer jobs compared to the latter. It may also be noted that the *total job allocation time* and the *total execution time* increase with a rise in the number of jobs, as can be

Table 10
Comparison of *Total execution time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total execution time (s)	Total execution time (s)	Total execution time (s)	Proposed vs. [10]	Proposed vs. [11]
Homogeneous Robots	2	4	730.00	197.33	178.66	75.52	9.46
	3	6	852.00	253.66	222.66	73.87	12.22
	4	8	1271.00	574.66	294.00	76.86	48.83
	5	10	1575.00	935.00	709.66	54.94	24.10
Heterogeneous Robots	2	4	414.66	318.33	290.00	30.06	8.89
	3	6	833.20	654.33	308.33	62.99	52.87
	4	8	991.45	697.66	530.00	46.54	24.03
	5	10	1221.00	862.00	793.66	34.99	7.92

Table 11
Comparison of *Energy refill time* (in s) and *Number of battery swaps* among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* with ϑ kept fixed at 1:2.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]		Partial reassignment [11]		Proposed method	
			Energy refill time (s)	Number of battery swaps	Energy refill time (s)	Number of battery swaps	Energy refill time (s)	Number of battery swaps
Homogeneous Robots	2	4	403.00	1	0	0	0	0
	3	6	305.00	1	0	0	0	0
	4	8	660.00	2	314.33	1	0	0
	5	10	712.00	2	444.00	2	315.33	1
Heterogeneous Robots	2	4	0	0	0	0	0	0
	3	6	333.66	1	321.00	1	0	0
	4	8	300.00	1	268.66	1	254.00	1
	5	10	434.00	3	386.00	2	351.66	1

Table 12
Comparison of *Average waiting time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* where ϑ takes different values - 1:1, 1:2, 1:3, 1:4 and 1:5.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Average waiting time (s)	Average waiting time (s)	Average waiting time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	2	151.73	96.20	55.06	63.71	42.76
		4	216.10	172.83	150.60	30.31	12.86
		6	361.33	222.30	195.40	45.92	12.10
		8	484.94	295.42	234.13	51.71	20.74
		10	748.73	577.22	461.92	38.30	19.97

Table 13
Comparison of *Total job allocation time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* where ϑ takes different values - 1:1, 1:2, 1:3, 1:4 and 1:5.

Type of robot	No. of robots	No. of Jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total job allocation time (s)	Total job allocation time (s)	Total job allocation time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	2	30.66	28.00	17.33	43.47	38.10
		4	42.66	35.66	32.00	24.98	10.26
		6	63.31	59.66	57.02	9.93	4.42
		8	92.00	84.66	78.00	15.21	7.86
		10	148.30	135.00	94.55	36.24	29.96

observed from Tables 6 and 7, respectively. This is because when the number of jobs in one experiment surpasses that of another, the time taken to allocate its constituent tasks and execute them also exceeds that of the latter experiment. In addition, the time difference between the arrival of two consecutive jobs also contributes to an increase in the *total job allocation time* and the *total execution time* of an experiment, as discussed in *Case #1*.

Case #3: $\not\mathbb{D}\mathbb{D}\mathbb{E}$

Tables 8, 9 and 10 provide the various times consumed when the jobs had both independent and dependent tasks, and the energy consumed during execution was also taken into account for all three algorithms (energy consumption was considered during the task allocation process only for the *proposed* algorithm). Table 11 portrays the *energy refill times* and *number of battery swaps* made for each of the three algorithms.

We have also conducted experiments to test the performance of the *proposed* algorithm and to compare against those of the other two algorithms by varying ϑ, whose results are portrayed in Tables 12 to 15. Table 12 depicts the *average waiting times*, Table 13 portrays the *total job allocation times*, and Table 14 portrays the *total execution times* for

Table 14
Comparison of *Total execution time* (in s) among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* where ϑ takes different values - 1:1, 1:2, 1:3, 1:4 and 1:5.

Type of robot	No. of robots	No. of jobs	Reassignment [10]	Partial reassignment [11]	Proposed method	Reduction in time (%)	
			Total execution time (s)	Total execution time (s)	Total execution time (s)	Proposed vs. [10]	Proposed vs. [11]
Heterogeneous Robots	2	2	239.34	148.66	70.00	70.75	52.91
		4	414.66	318.33	290.00	30.06	8.89
		6	873.63	724.66	618.66	29.18	14.62
		8	960.00	747.66	624.33	34.96	16.49
		10	1662.00	1241.25	993.00	40.25	20.00

Table 15
Comparison of *Energy refill time* (in s) and *Number of battery swaps* among the *Proposed*, *Partial reassignment* and *Reassignment* Algorithms for *Case #3* where ϑ takes different values - 1:1, 1:2, 1:3, 1:4 and 1:5.

Type of robot	No. of robots	No. of jobs	Reassignment [10]		Partial reassignment [11]		Proposed method	
			Energy refill time (s)	Number of battery swaps	Energy refill time (s)	Number of battery swaps	Energy refill time (s)	Number of battery swaps
Heterogeneous Robots	2	2	0	0	0	0	0	0
		4	0	0	0	0	0	0
		6	495.66	2	467.63	2	313.66	1
		8	390.00	2	385.33	2	336.33	2
		10	1013.00	4	634.33	4	585.00	3

the three algorithms. Table 15 depicts the *energy refill times* and *number of battery swaps* made for the three algorithms.

The *proposed* algorithm outperformed the other two algorithms in terms of *average waiting time*, *total job allocation time* and *total execution time* in all the experiments as portrayed in Tables 8, 9, 10, 12, 13 and 14 where the last column in each table represents the % reduction in these times achieved by the *proposed* algorithm when compared to the other two algorithms. This reduction in time is due to the reasons cited in *Case #1* and *Case #2*. In addition, as the other two algorithms do not consider the energy requirement of a task at the time of the task allocation process, the generated estimated costs and estimated bid values do not include the extra time required to visit the battery stockpile as and when such a need arises. This may not lead to an efficient allocation of tasks to robots.

It can also be noted from Tables 11 and 15 that the *number of battery swaps* in the *proposed* algorithm is generally lower than those in the other two algorithms because the other two algorithms did not take into account the energy need of the tasks during the task allocation process. The waiting times for the tasks that are to be executed after visiting the battery stockpile and also their successor tasks increase (as the successor tasks cannot start their executions unless their predecessor tasks have completed their executions), thereby contributing to the increase in the *average waiting time* and the *total execution time* of an experiment in case of the other two algorithms. All these contribute to the higher *average waiting times* and *total execution times* in the case of the other two algorithms as compared to the *proposed* algorithm. This highlights the significance of considering the energy requirements of the tasks and also the energy available in a robot *a priori* while estimating the execution costs and bid values during the task allocation process.

The *energy refill time* of an experiment depends on the distance between the execution locations of the tasks and the battery stockpile and not on the number of jobs and the *number of battery swaps*.

As stated in *Case #2*, here too the *total job allocation time* and the *total execution time* increase with an increase in the number of jobs as can be observed from Tables 9, 10, 13 and 14.

5.3. Discussions

The *proposed* algorithm can be made to cater to deadlines that put restrictions on the start times of tasks as in [10,11]. During the *task selection* stage, when a robot finds the feasible positions for task insertion into its *task list*, the positions for which the estimated costs (in terms of time taken to reach the execution location of the task) come out to be less than the deadline for the task, can become candidate positions in the *task list*. When a task is inserted into the *task list* of a robot, the tasks that are present after the newly inserted task get delayed. Hence, their starting times may exceed their deadlines. Therefore, among the candidate positions, positions for which the deadline constraints of both the new and old tasks are satisfied could be found. The one for which the estimated bid value is the lowest could be selected as the suitable position for the task. If such a position is not found, then the robot could ignore that task during the *task selection* stage. If more than one robot selects this task, then the conflicting selections could be resolved by the corresponding *Initialization Agent*. Before actual task execution, the deadline constraint of the tasks, however, may need to be re-checked. In case the robot is not able to reach the location of the task before its deadline, that task could be abandoned as in [10,11]. Any successor task of that task also needs to be abandoned.

The ability of mobile agents to react dynamically to critical situations makes it easier to build robust and fault-tolerant systems [47]. A robot may fail during task execution in the real world. The *proposed* algorithm can be made to handle such a situation using mobile agents. Mobile agents provide the advantage over other methods of fault detection where the participating robots are directly involved in the process [56]. In such scenarios, it puts an extra burden on the robots and compels them to stop their current work to take care of the situation. On the contrary, mobile agents can autonomously move into the network of robots without human intervention and take care of this situation with minimal interference to the robots in the network and without halting the executions being performed by other robots in the network. A mobile *Fault Detector Agent* could be made to periodically patrol the network of robots to detect a faulty robot. On encountering such a robot, it can be made to inform all the other robots in the network about the same. Unfinished tasks of this robot, together with the successor tasks of these unfinished tasks, could be retrieved and reallocated by the *Fault Detector Agent* using the *proposed* algorithm.

When a new robot is added to the network of robots, the newly added robot can create a mobile agent and subsequently inform the *Initialization Agents* about its existence, thereby making them aware of its presence during the task allocation process.

The use of mobile agents can allow this *proposed* algorithm to cater to dynamic changes during execution. Such dynamic changes could

include the abandoning of an already allocated task or a change in the location of a task. Under such conditions, one of the robots receiving this information could spawn a mobile agent which in turn could migrate across the network and ensure that the corresponding deletion of relevant tasks or changes in locations are made at the concerned robots.

Thus, apart from their use in the task allocation process, mobile agents can aid in realizing other functionalities such as fault detection, *on-the-fly* addition and deletion of robots in the network, etc. Since these agents can be embedded with behaviors, they can also be made to learn and adapt from the network they inhabit.

6. Conclusions and future work

This paper proposes a decentralized and distributed dynamic task allocation algorithm that falls under the *ST-SR-TA:SP* category. Unlike static task allocation problems, in dynamic type, jobs arrive *on-the-fly* into the network of robots making conflict-free allocations a big challenge, especially in decentralized systems. Jobs could consist of a set of either all independent tasks or a mixture of independent and dependent tasks. The proposed work provided an algorithm for conflict-free allocation of the tasks of all the incoming *n* jobs, injected *on-the-fly*, to a set of networked robots to minimize their *total execution time*. The *proposed* algorithm is energy-aware. It uses an adaptive energy threshold instead of a fixed one for making decisions on whether or not to visit a battery stockpile. In addition, the need to visit the battery stockpile is decided individually by each robot for its own tasks without any centralized agency. The proposed task allocation algorithm also considers the energy requirements of the tasks and the energy available within the robots both during task allocation (while generating the estimated costs and estimated bid values) and during actual execution. The selection of the compatible tasks of the jobs is carried out by the robots independently. The elimination of the conflicting selections of tasks by the robots and allocation of tasks to the robots are done by the respective *Initialization Agents* autonomously in a decentralized manner. These agents aid in exploiting the best of both centralized and decentralized systems and also, the benefits of auction-based task allocation algorithms. *In-schedule Dependencies* and *Cross-schedule Dependencies* have also been addressed during the task allocation process. The use of mobile agents can aid in augmenting this algorithm with strategies to cope with unforeseen situations such as delays during the execution of tasks caused due to obstacles, congestion due to the movement of multiple robots in the same place simultaneously, etc. Experiments are carried out using *Webots* and *Tartarus* and the *proposed* algorithm is compared with the *reassignment* and the *partial reassignment* algorithms. Experimental results portray the efficiency of the *proposed* algorithm in minimizing the *average waiting time*, *total task allocation time*, *total job allocation time*, and *total execution time* of an experiment as compared to the other two algorithms. The results also show the importance of considering energy requirements at the time of the task allocation process.

We plan to extend the present implementation of the *proposed* algorithm to other taxonomies of MRTA such as *ST-MR-TA= Single-Task robots, Multi-Robot tasks, Time-extended Assignment.*

On encountering an obstacle during actual execution, a robot may need to decelerate and then avoid it. This can consume extra energy. After avoiding the obstacle, the robot can sense its remaining energy (using on-board energy sensors), based on which the speed could be varied. Such cases may cause the robot to have an energy different from that estimated by it and, if required, eventually force it to visit the location of the battery stockpile. Hence, a suitable value of speed is determined with which it can either directly visit the battery stockpile or execute the next task and visit the battery stockpile if a need arises, whichever is feasible. Further, in case of extreme, yet infrequent scenarios, such as when a robot lands up with insufficient energy to even visit a battery stockpile, a request for help could be raised by spawning a mobile agent, which in turn would find a robot that could transport a battery and aid in a swap. We plan to consider the same in our future works involving real robots.

CRediT authorship contribution statement

Menaxi J. Bagchi: Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Shivashankar B. Nair:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Pradip K. Das:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgments

The first author would like to acknowledge the Ministry of Human Resource Development (MHRD), Government of India, for the financial support provided during the course of this work.

References

[1] G. Bekey, J. Yuh, The status of robotics, IEEE Robot. Autom. Mag. 15 (1) (2008) 80–86.
[2] J.K. Verma, V. Ranga, Multi-robot coordination analysis, taxonomy, challenges and future scope, J. Intell. Robot. Syst. 102 (2021) 1–36.
[3] B.P. Gerkey, M.J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, Int. J. Robot. Res. 23 (9) (2004) 939–954.
[4] G.A. Korsah, A. Stentz, M.B. Dias, A comprehensive taxonomy for multi-robot task allocation, Int. J. Robot. Res. 32 (12) (2013) 1495–1512.
[5] E. Nunes, M. Manner, H. Mitiche, M. Gini, A taxonomy for task allocation problems with temporal and ordering constraints, Robot. Auton. Syst. 90 (2017) 55–70.
[6] A. Khamis, A. Hussein, A. Elmogy, Multi-robot task allocation: A review of the state-of-the-art, in: Cooperative Robots and Sensor Networks 2015, Springer, 2015, pp. 31–51.
[7] F. Quinton, C. Grand, C. Lesire, Market approaches to the multi-robot task allocation problem: a survey, J. Intell. Robot. Syst. 107 (2) (2023) 29.
[8] L.E. Parker, F. Tang, Building multirobot coalitions through automated task solution synthesis, Proc. IEEE 94 (7) (2006) 1289–1305.
[9] H. Chakraa, F. Guérin, E. Leclercq, D. Lefebvre, Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art, Robot. Auton. Syst. (2023) 104492.
[10] A. Whitbrook, Q. Meng, P.W. Chung, Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 732–747.
[11] M. Yang, W. Bi, A. Zhang, F. Gao, A distributed task reassignment method in dynamic environment for multi-UAV system, Appl. Intell. 52 (2) (2022) 1582–1601.
[12] A. Whitbrook, Q. Meng, P.W. Chung, A novel distributed scheduling algorithm for time-critical multi-agent systems, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2015, pp. 6451–6458.
[13] H.-L. Choi, L. Brunet, J.P. How, Consensus-based decentralized auctions for robust task allocation, IEEE Trans. Robot. 25 (4) (2009) 912–926.
[14] S.S. Jha, S.B. Nair, TANSA: task allocation using nomadic soft agents for multirobot systems, IEEE Trans. Emerg. Top. Comput. Intell. 2 (4) (2017) 308–318.
[15] F. Sempé, A. Munoz, A. Drogoul, Autonomous robots sharing a charging station with no communication: a case study, in: Distributed Autonomous Robotic Systems 5, Springer, 2002, pp. 91–100.
[16] F. Michaud, E. Robichaud, Sharing charging stations for long-term activity of autonomous robots, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol. 3, IEEE, 2002, pp. 2746–2751.
[17] D.-H. Lee, S.A. Zaheer, J.-H. Kim, Ad hoc network-based task allocation with resource-aware cost generation for multirobot systems, IEEE Trans. Ind. Electron. 61 (12) (2014) 6871–6881.
[18] D.-H. Lee, S.A. Zaheer, J.-H. Kim, A resource-oriented, decentralized auction algorithm for multirobot task allocation, IEEE Trans. Autom. Sci. Eng. 12 (4) (2014) 1469–1481.

[19] D.-H. Lee, Resource-based task allocation for multi-robot systems, Robot. Auton. Syst. 103 (2018) 151–161.
[20] X.-F. Liu, B.-C. Lin, Z.-H. Zhan, S.-W. Jeon, J. Zhang, An efficient ant colony system for multi-robot task allocation with large-scale cooperative tasks and precedence constraints, in: 2021 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2021, pp. 1–8.
[21] M.J. Bagchi, D.D. Kulkarni, S.B. Nair, P.K. Das, On embedding a dataflow architecture in a multi-robot system, in: 2022 Sixth IEEE International Conference on Robotic Computing, IRC, IEEE, 2022, pp. 271–276.
[22] D. Datsko, F. Nekovar, R. Penicka, M. Saska, Energy-aware multi-UAV coverage mission planning with optimal speed of flight, IEEE Robot. Autom. Lett. (2024).
[23] V.A. Pham, A. Karmouch, Mobile software agents: An overview, IEEE Commun. Mag. 36 (7) (1998) 26–37.
[24] I.Y. Jung, H.Y. Yeom, An efficient and reliable message passing by mobile agent, in: Proceedings 15th International Conference on Information Networking, IEEE, 2001, pp. 900–905.
[25] M. Chen, S. Gonzalez, V.C. Leung, Applications and design issues for mobile agents in wireless sensor networks, IEEE Wirel. Commun. 14 (6) (2007) 20–26.
[26] A. Boukerche, R.B. Machado, K.R. Jucá, J.B.M. Sobral, M.S. Notare, An agent based and biological inspired real-time intrusion detection and security model for computer network operations, Comput. Commun. 30 (13) (2007) 2649–2660.
[27] O.R. Zaíane, Building a recommender agent for e-learning systems, in: International Conference on Computers in Education, 2002. Proceedings, IEEE, 2002, pp. 55–59.
[28] W.W. Godfrey, S.B. Nair, An immune system based multi-robot mobile agent network, in: International Conference on Artificial Immune Systems, Springer, 2008, pp. 424–433.
[29] P. Maes, R.H. Guttman, A.G. Moukas, Agents that buy and sell, Commun. ACM 42 (3) (1999) 81–ff.
[30] J.L. Posadas, J.L. Poza, J.E. Simó, G. Benet, F. Blanes, Agent-based distributed architecture for mobile robot control, Eng. Appl. Artif. Intell. 21 (6) (2008) 805–823.
[31] S.S. Jha, S.B. Nair, On a multi-agent distributed asynchronous intelligence-sharing and learning framework, in: Transactions on Computational Collective Intelligence XVIII, Springer, 2015, pp. 166–200.
[32] S.S. Nestinger, B. Chen, H.H. Cheng, A mobile agent-based framework for flexible automation systems, IEEE/Asme Trans. Mechatron. 15 (6) (2009) 942–951.
[33] S. Bandyopadhyay, K. Paul, Evaluating the performance of mobile agent-based message communication among mobile hosts in large ad hoc wireless network, in: Proceedings of the 2nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 1999, pp. 69–73.
[34] F. Bellifemine, A. Poggi, G. Rimassa, JADE: a FIPA2000 compliant agent development environment, in: Proceedings of the Fifth International Conference on Autonomous Agents, 2001, pp. 216–217.
[35] T. Walsh, Concordia: An infrastructure for collaborating mobile agents, in: Proc. 1st Int. Workshop on Mobile Agents, MA'97, 1997.
[36] H. Tai, K. Kosaka, The aglets project, Commun. ACM 42 (3) (1999) 100–101.
[37] B. Chen, H.H. Cheng, J. Palen, Mobile-C: a mobile agent platform for mobile C/C++ agents, Softw. - Pract. Exp. 36 (15) (2006) 1711–1733.
[38] J. Matani, S.B. Nair, Typhon-a mobile agents framework for real world emulation in prolog, in: Multi-Disciplinary Trends in Artificial Intelligence: 5th International Workshop, MIWAI 2011, Hyderabad, India, December 7–9, 2011. Proceedings 5, Springer, 2011, pp. 261–273.
[39] T. Semwal, M. Bode, V. Singh, S.S. Jha, S.B. Nair, Tartarus: a multi-agent platform for integrating cyber-physical systems and robots, in: Proceedings of the 2015 Conference on Advances in Robotics, 2015, pp. 1–6.
[40] B. Kaleci, O. Parlaktuna, Performance analysis of bid calculation methods in multirobot market-based task allocation, Turk. J. Electr. Eng. Comput. Sci. 21 (2) (2013) 565–585.
[41] X. Chen, P. Zhang, G. Du, F. Li, A distributed method for dynamic multi-robot task allocation problems with critical time constraints, Robot. Auton. Syst. 118 (2019) 31–46.
[42] L. Hou, L. Zhang, J. Kim, Energy modeling and power measurement for mobile robots, Energies 12 (1) (2018) 27.
[43] M. Chaudhari, L. Vachhani, R. Banerjee, Towards optimal computation of energy optimal trajectory for mobile robots, IFAC Proc. Vol. 47 (1) (2014) 82–87.
[44] B. Zou, X. Xu, R. De Koster, et al., Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system, European J. Oper. Res. 267 (2) (2018) 733–753.
[45] N. Kalra, R. Zlot, M.B. Dias, A. Stentz, Market-based multirobot coordination: A comprehensive survey and analysis, Def. Tech. Inf. Cent. (2005) 1257–1270.
[46] P. Mahato, S. Saha, C. Sarkar, M. Shaghil, Consensus-based fast and energy-efficient multi-robot task allocation, Robot. Auton. Syst. 159 (2023) 104270.
[47] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Commun. ACM 42 (3) (1999) 88–89.
[48] E. Nunes, M. McIntire, M. Gini, Decentralized multi-robot allocation of tasks with temporal and precedence constraints, Adv. Robot. 31 (22) (2017) 1193–1207.
[49] M. Hoeing, P. Dasgupta, P. Petrov, S. O'Hara, Auction-based multi-robot task allocation in comstar, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 1–8.
[50] B.P. Gerkey, M.J. Mataric, Sold!: Auction methods for multirobot coordination, IEEE Trans. Robot. Autom. 18 (5) (2002) 758–768.
[51] O. Michel, Cyberbotics ltd. webots™: professional mobile robot simulation, Int. J. Adv. Robot. Syst. 1 (1) (2004) 5.
[52] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, Swi-prolog, Theory Pract. Log. Program. 12 (1–2) (2012) 67–96.
[53] J.S. Nair, D.D. Kulkarni, A. Joshi, S. Suresh, On decentralizing federated reinforcement learning in multi-robot scenarios, in: 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference, SEEDA-CECNSM, IEEE, 2022, pp. 1–8.
[54] Y. Tekol, contributors, PySwip v0.2.10, 2020, URL https://github.com/yuce/pyswip.
[55] M. Soori, B. Arezoo, R. Dastres, Optimization of energy consumption in industrial robots, a review, Cogn. Robotics 3 (2023) 142–157.
[56] V.C. Kalempa, L. Piardi, M. Limeira, A.S. de Oliveira, Multi-robot preemptive task scheduling with fault recovery: A novel approach to automatic logistics of smart factories, Sensors 21 (19) (2021) 6536.

Menaxi J. Bagchi received a B.Tech. degree from the SRM University, Chennai, India in 2016, and an M.Tech. degree from the International Institute of Information Technology (IIIT) Bhubaneswar, India in 2018 in Computer Science and Engineering. She is currently pursuing her Ph.D. Degree in Computer Science and Engineering from the Indian Institute of Technology (IIT) Guwahati, India. Her research interests include multi-robot systems and multi-agent systems.

Shivashankar B. Nair received his Ph.D., M.E., and M.Sc. degrees from Amravati University, India. He is a Professor at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati, India. He has co-authored the book entitled Artificial Intelligence (with E. Rich and K. Knight) published by McGraw-Hill. His research interests are Networked Robotics, Mobile Agents, and Bioinspired systems. He is the Editor of the IETE Technical Review (Taylor & Francis) and a Fellow of the Institution of Engineers, India, and the Robotics Society of India.

Pradip K. Das (Member, IEEE) received a B.Sc. degree (Hons.) in statistics from the Arya Vidyapeeth College, Gauhati University, India, in 1989, the M.Sc. degree in mathematical statistics from the Ramjas College, University of Delhi, India, in 1991, and the Ph.D. degree in computer science from the University of Delhi, in 1999. He is currently a Professor at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, India.