Task allocation for a swarm of robots

Submitted as Research Report for SIT723

18/06/2021

T1-2021

Idhant Bhambri

217613916

Bachelor of Software Engineering Honours (S464)

Supervised by: Dr. Jan Carlo Barca, Dr. Kevin Lee

Abstract

Bushfires in Australia are a widespread and regular occurrence that have contributed significantly to shaping the nature of the continent over millions of years. Eastern Australia is one of the most fire-prone regions of the world, and its predominant eucalyptus forests have evolved to thrive on the phenomenon of bushfire.

The main goal of the project is to investigate how swarms of drones can be allocated tasks in a firefighting scenario where the drones are disconnected from the cloud, and from human operators. The algorithms which will be developed as a part of this project should allow the swarm to autonomously carry out multiple tasks such as extinguishing fires and retrieving people that are trapped behind the fire front in parallel.

In the literature review the definition for the ideal algorithm was given along with three suitable algorithms that matched the ideal algorithm. The literature review is concluded with a gap analysis of current research, using which the research questions were defined with their importance and relevance to the project.

A research design and methodology was created to enable the answering of the research questions. The experiment design was presented along with the data collection rules and how the data can be used to evaluate the experiment quantitatively. Possible errors and risks were also mentioned and the strategy on how to mitigate them.

Using the research design, a minimal viable artefact was developed. The development tools were touched on and the Centralised version of the Bees Algorithm was thoroughly discussed alongside the code logic and functions. How the algorithm allocated the task was broken down with the relevant mathematical functions and models.

The created artefact was then evaluated quantitatively and qualitatively. The evaluation was divided in the respective subsections with the qualitative evaluation aiming to answer whether the experiment accomplished its goals and the quantitative evaluation aiming to answer how well the experiment accomplishes the goals.

Two sub-experiments were presented and discussed in context of the research questions. Experiment-1 showed that when the amount of tasks were increased the algorithm displayed a linear global and local allocation time trend. Similarly Experiment showed that when the location of robots and tasks in the system was randomised the algorithm achieved the allocation in a similar liner fashion. It was noted that due to the small subset, the true task increase vs performance trend was not discussed rather its initial tangent when the number of tasks in the system were fairly low.

Finally the results of the experiment design, development and evaluation are discussed in relation to the research questions and how the artefact was able to answer the questions partially.

Contents

List of Figures

List of Tables

1 Introduction

Bushfires in Australia are a widespread and regular occurrence that have contributed significantly to shaping the nature of the continent over millions of years. Eastern Australia is one of the most fire-prone regions of the world, and its predominant eucalyptus forests have evolved to thrive on the phenomenon of bushfire. However, the fires can cause significant property damage and loss of both human and animal life. Bushfires have killed approximately 800 people in Australia since 1851 (human-life factor), and billions of animals (ecological factor). In 2012, the total accumulated cost was estimated at $1.6 billion (economical factor).

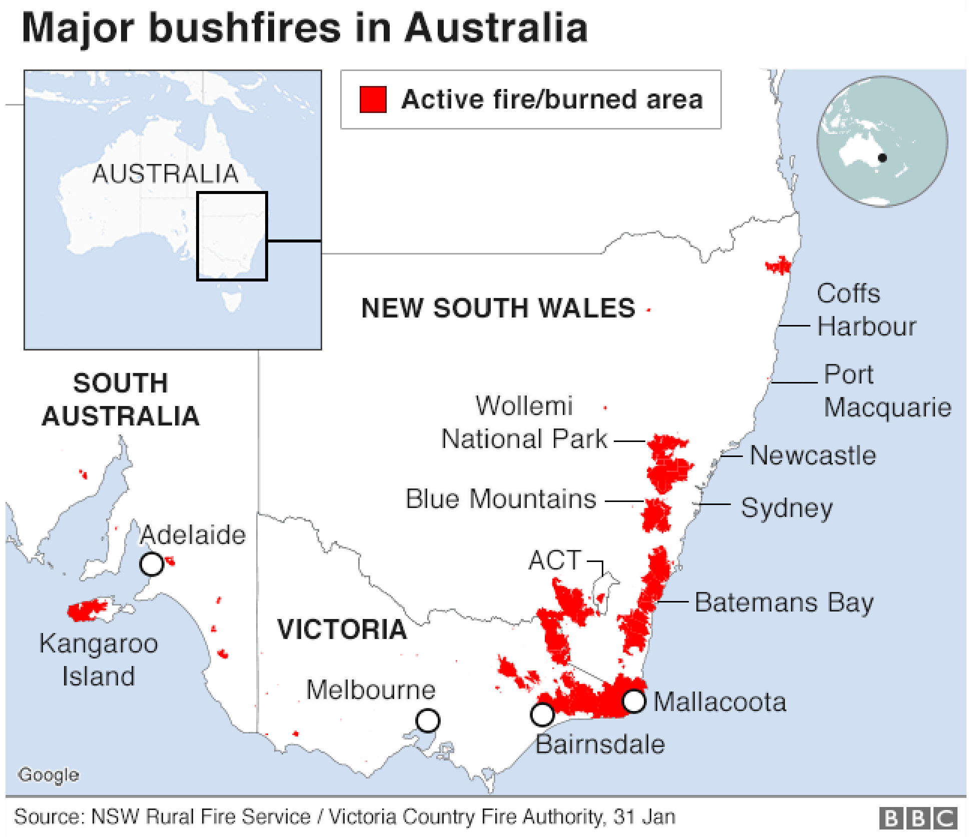

Figure 1: Major Bushfires in Australia

Bushfires are especially problematic in remote areas where it is difficult to stop the spread of fires due to the terrain and logistical problems. In the bushfires of 2019-2020, kangaroo island suffered several fires which burnt more than 2,100 square kilometres (520,000 acres), approximately 52% of the Island. The main question is, what can we do to prevent this in the future?

With the exponential rise of technology in the past decade, it has become possible

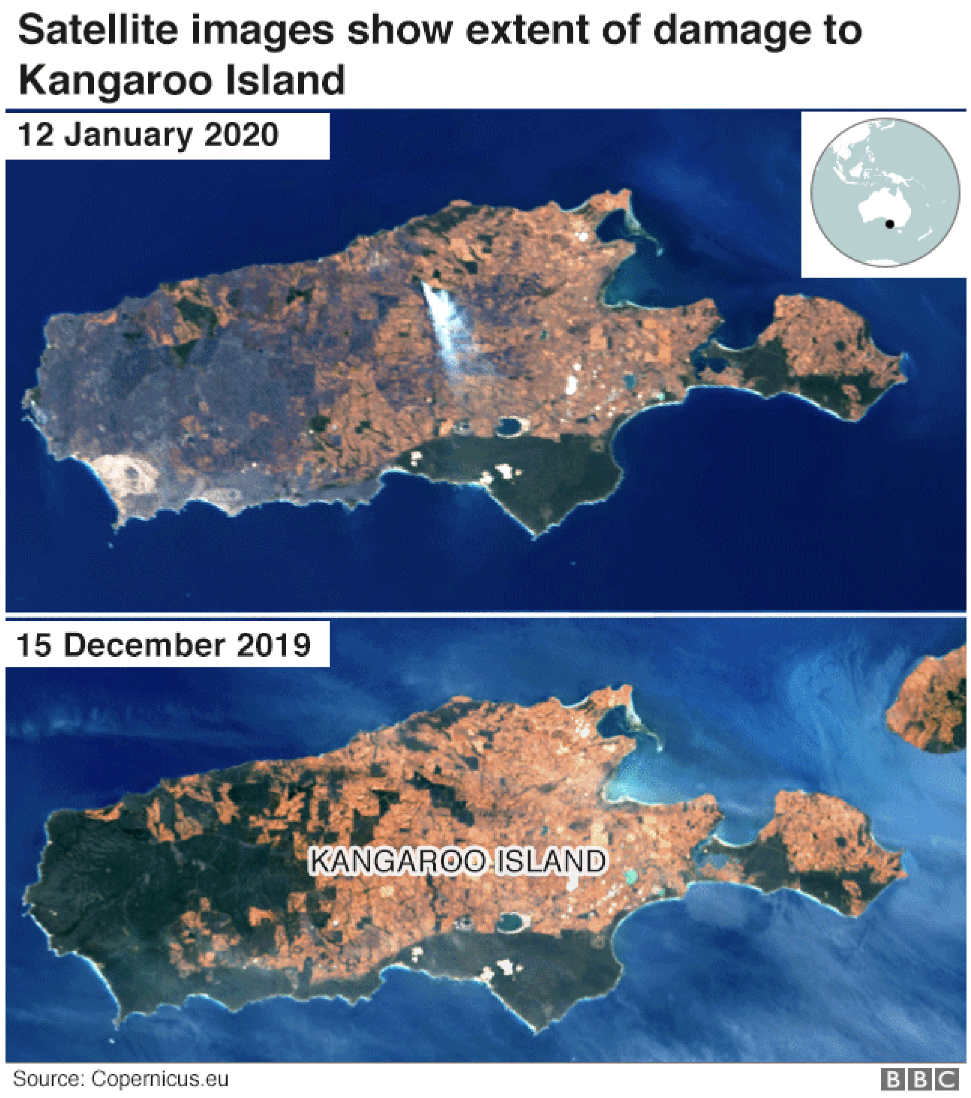

Figure 2: Satellite Image showing damage to Kangaroo Island

to implement a Semi-Autonomous Multiple Robot system to solve a complex problem. Swarm of robots cooperating with each other can be used in our scenario to mitigate and stop the spread of fires faster and without putting firefighters lives in danger. Instead of humans, the robots perform the high-risk tasks.

Using robotic swarms in remote areas gives us additional challenges that we need to keep in mind. Communication of robots in a remote area is difficult due to limited resources and methods available, thus using centralized cooperation methods is not efficient or scalable. Our task allocation algorithm should work in a decentralized manner. Due to lack of cloud support, computing cost is a scarce resource which should be reserved for the execution purposes rather than heavily being used to allocate tasks.

The main goal of the project is to investigate how swarms of drones can be allocated tasks in a firefighting scenario where the drones are disconnected from the cloud, and from human operators. The algorithms which will be developed as a part of this project should allow the swarm to autonomously carry out multiple tasks such as extinguishing fires and retrieving people that are trapped behind the fire front in parallel.

1.1 Structure

In the Section 2, the literature is reviewed. The areas of the literature review are divided into 5 sub-sections which conclude with a gap analysis of the current literature. Section 3 presents the research design and methodology of the minimal-viable artefact. Details of the artefact are presented along with important factors such as data collection policy, risk and error mitigation strategy and the research questions. Section 4 describes how the research design was used to implement the minimal viable artefact. The artefact structure is presented and discussed in respect to the code developed. Section 5 shows the experiment procedure and evaluates the artefacts qualitatively and quantitatively on the basis of research questions (RQs) and the experiment design. In the Section 6, the experiment results are discussed.. Section 7 discusses any threats to validity and Section 8 concludes the research report.

2 Literature Review

Task allocation techniques have been researched extensively over the years and are still evolving over time. This section discusses the relevant literature which were reviewed thoroughly and divided into few subcategories. Categories such as Multiple Agent Systems, Multiple Robot Systems, Multiple Robot Task Allocation, Task Allocation Models, and Task Allocation Algorithms are discussed in detail. These categories are essential to understand the scenario problem and find solutions by identifying gaps in the current literature and formulating research questions which are discussed in the next chapter.

2.1 Multiple Agent System (MAS)

A multiple agent system is a computerized system consisting of multiple agents interacting within an environment to solve a problem. MAS is commonly applied in the engineering and technology fields to solve problems that are difficult for individual agents to solve.

MAS plays a fundamental part in the development of our scenario problem and is the backbone of the Multiple Robot System (MRS) which will be discussed in section-2.2.

2.1.1 Intelligent Agents

One of the important parts of the MAS are the agents themselves which are often referred as “Intelligent agents” The agents that are used in the MAS are often called Intelligent agents. Stuart and Russell define an intelligent agent as:

“An intelligent agent is a physical (robot) or virtual (software program) entity that can autonomously perform actions on an environment while perceiving this environment, to accomplish a goal A rational agent seeks to perform actions that result in the best outcome.” [35]

One of the important underlying architecture behind an intelligent agent is its cognitive architecture, which can be thought of as its brain. Pat et al define the parts of this the cognitive architecture consisting of perception, reasoning, learning, decision making, problem solving, interaction and communication. “Its evaluation is based on domain specific performance measures, generality, versatility, rationality, optimality, efficiency, scalability, autonomy, and improvability.” [26]

Agents can also be categorized in terms of how the Sense, Act and React. Rizk et al distinguishes three types of agents: reactive, deliberative and hybrid agents. [34]

- Reactive agents simply react to environmental changes. Their workflow contains two primitives: sense (S) and act (A).
- Deliberative agents initiate actions without any external trigger and rely on planning. This sense-plan-act or sense-model-plan-act paradigm contains three primitives which are performed sequentially: sense (S), plan (P) and act (A).

- Hybrid agents perform actions based on a planning algorithm or react to current perceptions.

Rizk et al also visualizes the workflow of these different agents. This is displayed in the figure - 3. [34]

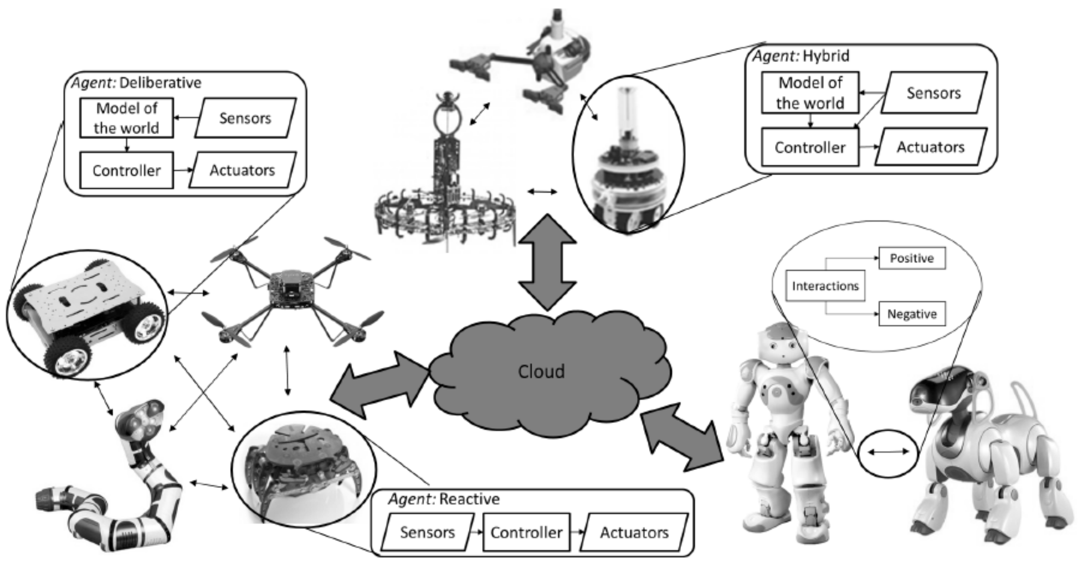

Figure 3: Three-tier heterogeneous MRS architecture: robot, locally connected MRS, group of MRS connected.

Pat et al also proposed a finer characterization model for distinguishing agents.

“A Finer categorization divides agents into: simple reflex (react to current sensory input), model-based reflex (keep an internal state of the environment), goal-based (perform actions to complete a goal), and utility-based (maximize a utility function) agents.” [26]

2.1.2 MAS Classification

MAS have been classified under different models using different criteria. One classification model by Stone an Veloso, classifies the MAS based on agents’ diversity and communication capabilities. This model consists of four classes: homogeneous non-communicative, homogeneous communicative, heterogeneous non-communicative, and heterogeneous communicative. [41]

The definitions for these criteria are as follows:

- Homogeneous = Similar types of agents with similar functions in the system.
- Heterogeneous = Different types of agents with different functions in the system.
- Communicative = agents can communicate with each other.
- Non-Communicative = agents lack the ability or are restricted in communication with other agents.

Another way to classify the MAS is based on how the communication of the agents in the system occurs. Lynne et al classifies the different MAS as centralized, hierarchical, decentralized or hybrid architectures. [32]

These classifications can be defined as follows:

- Centralized = Communication occurs through a single master agent to all the other sub agents.
- Hierarchical = Communication channels are divided in sub-categories where the information flows top to bottom.
- Decentralized = All the agents in this system are equal and communication agent is decentralized. Information can be passed through any agent.
- Hybrid = A combination of the centralized, hierarchical, and decentralized categories.

A broader classification model is also presented by Rizk et al, where the system can be described as positive versus negative where agents aid or do not interfere with each other versus actively impede other. For the focus of this project scenario, our focus is on the Cooperative MAS systems. [34]

Deciding on a particular MAS Is also important and depends on different criteria. Paul et al gives divides the evaluation criteria‘s into two categories, domain specific or domain invariant. [7]

- Domain specific criteria quantify performance. For search and rescue, performance measures include the number of rescued persons or extinguished fires.

- Domain invariant criteria include solution optimality, algorithm time and space complexity, load balancing, fairness, resource utilization and re-allocation quickness, communication overhead, robustness to noise and agent failures, and scalability.

2.2 Multiple Robot Systems (MRS)

The advancement of robots has come a long way since we started creating trivial robotic mechanism. As the technology behind robots has improved, so has their uses in different areas of problem solving. Many proposals have been made on how robots can solve new and interesting problems for us in the future. These problems require robots that are capable of performing certain tasks. In some of these advanced problems it is often difficult to create a single robot that can perform all the tasks necessary to achieve the final goal. Just as a human working in an office cannot solve all the problems without the help of a team of other humans, the same can be said for robots. The advance problems often require robots that are capable of cooperating with each other to solve a common goal.

Some benefits of MRS include but not limited to:

- Resolving task complexity: as discussed, some tasks are too complex for any single robot to accomplish but this task complexity can be resolved by subdivision of tasks.
- Increasing the performance: if the robots cooperate with each other, whether it be directly assisting each other or indirectly by communication to avoid conflicts.
- Increasing reliability: by having multiple robots in a system, its more reliable as it doesn't depend on any single agent to complete a task, thus reducing any bottlenecks especially in a critical situation.
- Simplicity in design: having smaller and simpler robots in a system is better to implement as they will be cheaper than having to build a single robot with all the functionality.

MRS is related to MAS and can be thought of a MAS system where the agents are restricted to being robots. This chapter will be focusing on the Multiple Robot System Architecture and other important areas which are related to it.

2.2.1 MRS Workflow

The MRS consists of some basic building blocks which are commonly present in the MRS Architecture. These building blocks form a workflow which shows how the system solves complex tasks through automation and human involvement.

Phillippe et al describes the main building blocks of a MRS workflow as task decomposition, coalition formation, task allocation, task execution/planning and control. [5]

- Task decomposition: division of complex tasks into simpler sub-tasks
- Coalition formation: formation of agent teams
- Task allocation: assignment of sub-tasks to agent teams for execution
- Task execution/planning and control: completion of a task by performing a sequence of actions on the environment.

Jutta and Oskar describe an MRS workflow (figure - 4) with a mixture of automation and human work. [21]

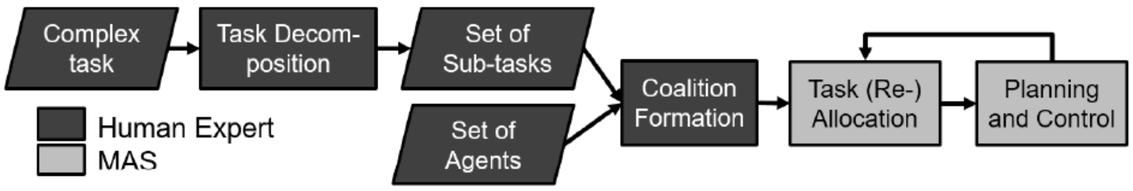

Figure 4: Workflow for the MRS proposed by Jutta and Oskar.

In the workflow described, some of the task's parts are handled by a human expert while others are done by MRS. In section-2.2.2, Automation levels in MRS are reviewed in depth.

1. Task Decomposition

Task Decomposition is the first step in the MRS workflow. It divides the main complex task into set of simpler sub-tasks that can be isolated or dependent on each other. For example, mapping a cave system is a complex task, the sub tasks can be mapping individual cave networks.

While a trivial MRS usually requires manual decomposition of tasks, there are ways to automate this process. Dias et al divides the process into three main categories, decompose-then-allocate, allocate-then-decompose and simultaneous decomposition and allocation. [10]

- Decompose-then-allocate algorithms first decompose a complex task into a list of sub-tasks then allocates these various sub-tasks to the available agents. “These approaches have found limited success due to the high level of domain-specific understanding required to understand the relationship between sub-tasks.” [34]
- Allocate-then-decompose algorithms, first allocate a list of tasks to agent groups and then each agent group divides these tasks into sub-tasks among their agents. “This approach allows agents to decompose tasks based on the agents’ specific skill set, allowing them to efficiently execute tasks.” [34]
- Simultaneous decomposition and allocation algorithms combine both the steps of MRS. “This approach produces more system specific decomposition’s by providing feedback to improve task decomposition based on agent capabilities. However, it could be more time consuming due to sub-task fine-tuning.” [34]

2. Coalition Formation

After a complex task is decomposed into a list of sub-tasks, these sub-tasks need to be allocated to a robot or group of robots for completion. Because of the nature of MRS, usually these tasks are allocated to a group of robots, before this allocation can occur, the robots need to be divided in sub-groups. This step of the MRS is called Coalition formation.

The agents that are being divided can be cooperative or non-cooperative. For our scenario, we will focus on cooperative coalition formation.

Coalition formation can also be performed offline to form static coalitions or online to form dynamic teams that can adjust to the environment. [15]

3. Task allocation

Task allocation assigns the sub-tasks/ tasks to an agent/group of agents and aims to find the most optimal task assignment solution. As task allocation is an important part of the project it is discussed in section-2.3 in more detail.

Most of the task allocation algorithms focus on ensuring that the distance travelled by the robot is minimized while also ensuring that the robot is capable to perform the task successfully. In section-2.4.3, solution model is discussed in more detail.

Task allocation algorithms are usually implemented on centralized, decentralized, or distributed topologies. The types of task allocation algorithms are discussed in more detail in section-2.3.2

- In the centralized topology, there is a single main node through which communicates on behalf of all the other nodes. While these are the simplest types of algorithms and often offer optimal solutions, they also are prone to failure and are poorly scalable.
- In the Decentralized topology, the nodes are divided in groups where each group has a group leader which communicates on the behalf of the group. These types of algorithms are a mixture of distributed and centralized topologies. Sometimes they are also referred to as distributed algorithms with a temporal agent.
- In the Distributed topology, all the nodes are equal, and communication is not reliant on any single node. This allows greater flexibility and scalability at the cost of solution optimization for the algorithm.

Another important part of task allocation is task re-allocation. In a case where the task allocated to the agent cannot be performed successfully, it must be re-allocated to another agent. "However, repeated preempting could cause the agents to never complete their tasks causing the whole system to fail. Questions like when a task should be preempted, how often should preemption be considered a suitable choice and others are important directions to investigate for effective task allocation in real-world environments." [34]

4. Task Planning and Execution

After the tasks have been allocated to agents, they still need to complete it successfully. This decision-making step in the MRS determines how the agents should plan and execute the tasks. The planning and execution of tasks is done using a decision-making model which can differ from each other in their use.

Some of the common models are Robotic Learning, Game theory and Swarm intelligence models.

- Swarm intelligence, inspired by social animals, models the behaviour of many decentralized cooperative autonomous agents. Such models are mainly characterized by self-organized and distributed behaviour of locally aware and locally interacting agents. [2]
- Game theory models include many partially observable stochastic games which are probabilistic in nature where the payoffs are unknown to agents but depend on their actions, and state of the game also depends on the previous world state and the agent's actions. [38]
- Robotic Learning models allows the agents to learn a policy or set of commands by rewarding good behaviour (correct execution of a step) and punishing bad behaviour (incorrect execution of a step).

2.2.2 MRS Automation Levels

Rizk et al proposes a model to distinguish MRS based on the complexity of the cooperative tasks they execute and their level of automation, from most to least automated.

"In the first level (least automated), only task execution is automated, while the second level also automates either task allocation or coalition formation but not both. The third level automates both coalition formation and task allocation but does not automate task decomposition. The fourth level automates the entire system." [34]

- Fourth Level of autonomation; in theory, this type of MRS model automates all the workflow to solve the complex task. No MRS models exist currently which achieves the 4th level of autonomation.
- Third Level of autonomation; A few MRS models exist where coalition formation, task allocation and task execution were done without any human interference.

[18] “These models have been tested in simulation environments or in small sized experiments but have shown promising results.”

- Second Level of autonomation; in these types of MRS models, two steps are automated and the other two are handled manually by a human operator. In theory these can be any two steps but the most common combinations are “task decomposition and coalitions are predefined by human experts but task allocation and execution are performed autonomously by MRS, or task decomposition and allocation are performed by human experts but coalition formation and task execution are performed by MRS.” [34]
- First Level of autonomation; this is the most basic MRS model where only one workflow step is accomplished using automation and the rest needs to be handled by a human operator.

2.3 Multiple Robot Task Allocation (MRTA)

One of the most challenging problems of MRS is the Multiple Robot Task Allocation, especially when it comes to heterogeneous robots with different hardware who are required to perform different tasks of different complexities in the most optimal way. The MRTA problem can be generalised into an optimal assignment problem where the objective is to assign a set of robots to a set of tasks such that the overall performance of a system can be optimized. In this chapter, Key areas of MRTA are discussed in detail.

2.3.1 MRTA Problem

The problem of MRTA is of a NP-hard nature except for the simplest models. That is, there is no expectancy of finding scalable, quick algorithms to solve the complete problem to optimality. Hence, most practical solutions must make compromises. [13]

According to Gerkey and Mataric, the Problem of MRTA can be simplified into two sub-problems.

- How a set of tasks is assigned to a set of robots?

- How the behaviour of a robot team is coordinated to achieve the tasks efficiently and reliably?

Also, because the problem of task allocation is a dynamic decision that can vary in time due to different environmental and system changes, the problem needs to be solved iteratively over time. This makes the problem of task allocation more difficult to solve optimally. [12]

1. Optimal Assignment Problem

Khamis formulates the MRTA problem as an optimal assignment problem where the objective is to optimally assign a set of robots to a set of tasks in such a way that optimizes the overall system performance subject to a set of constraints. Figure-5 shows the illustration as depicted in the literature. [20]

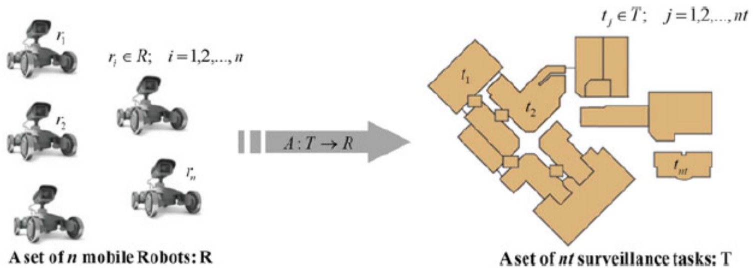
r_1
$r_i \in R; \quad i = 1,2,\ldots,n$
r_2
r_n
$A : T \to R$
$t_j \in T; \quad j = 1,2,\ldots,nt$
t_1
t_2
t_{nt}
A set of n mobile Robots: R
A set of nt surveillance tasks: T

Figure 5: MRTA Problem Formulation.

He further explains the problem mathematically by defining the following:

1. R: a team of mobile robots ri; i = 1, 2, . . . n. 2. T: a set of tasks ti j; j = 1, 2, . . . nt. 3. U: a set of robots' utilities, ui j is the utility of robot i to execute task j.

For a single sensor task, the problem is to find the optimal allocation of robots to tasks, which will be a set of robot and task pairs [9]:

For the general case, the problem is to find the optimal allocation of a set of tasks to a subset of robots, which will be responsible for accomplishing it [44]:

In some MRTA approaches such as market-based approaches, each robot r R can express its ability to execute a task t T, or a bundle of tasks G T through bids br (t) or br (G). The cost of a bundle of tasks can be simply computed as the sum of costs of the individual tasks:

where f is the number of tasks of the bundle G. The group's assignment determines the bundle G T of tasks that each robot r R receives. Mathematically, the problem can be stated: given an n ×n matrix W, find permutation of 1, 2, 3, . . . n for which:

"When there are more tasks than agents, OAP can still be used in its iterative form: in each computation, the optimal solution is found with one task per agent, leaving excess tasks unassigned. Assuming enough time elapses between task completions, this iterative algorithm can be used to find a new assignation upon each task completion." [29]

2. Discrete Fair Division

The MRTA problem can be seen as an example of a Fair Division Problem. [20]

Given a set of N robots (r1, r2, . . . rN) and a set of tasks S. It is required to divide S into N shares (s1, s2, . . . sN) so that each robot gets a fair share of S. A fair share is a share that, in the opinion of the robot receiving it, is worth 1/N of the total value of S.

2.3.2 MRTA Problem Approaches

The most common way to solve a MRTA problem is to use a Model based approach. This chapter covers the most relevant MRTA models including Cost-based, Deterministic, Stochastic, Auction-based, and Behavioral.

1. Cost models

The cost model works around principles such as costs, priority, fitness, and reward to formulate a model. A combination of these principles is used to calculate the utility or value of performing a task.

- Cost: a characterization of the cost that it takes for a robot to execute a task. Examples are time to reach a goal, distance traveled, energy consumed, etc. [8, 25]
- Fitness: a characterization of "how well" an agent can perform a task. Usually a normalized range is used, in which 0 naturally represents tasks undoable by an agent. [40]
- Reward: a characterization of the gain of completing a task, It acts as an incentive to perform tasks. [39]

- Priority: a characterization of the urgency of completing a task. Depending on how the system is designed, higher priority tasks can preempt all lesser tasks. [1]

Examples of commonly found utilities are:

$$\text{Utility}=\text{Reward}-\text{Cost}$$

$$\text{Utility}=\text{Fitness}-\text{Cost}$$

$$\text{Utility}=\text{Priority-Cost}$$

Essential it comes down to the ability, gain, need to do a task versus the cost of doing a task to calculate the utility. It is important to note that utility can never be negative, and a floor function is used to ensure that,

$$\text{Utility}=\max\left(\text{Reward}-\text{Cost}\ ,\ 0\right)$$

This gives two opposite ways of analyzing this model: [29]

- Minimization: when reasoning purely in terms of costs, higher values are worse, and the objective is to minimize the costs resulting from the allocation. This is often found in routing approaches where the bulk of the cost is goal reaching. [25]
- Maximization: when thinking in terms of utilities, higher values are better, and the objective is one of maximization. These are more naturally found where the underlying problem is also of maximization.

2. Deterministic models

Deterministic model assumes that the system will always follow the same procedure to allocate set of task to the set of robots. “The allocation process is already determined by existing and previous factors in the system.” The basis for this model comes from the philosophical theory of determinism which eludes that all events, including moral choices, are completely determined by previously existing causes.

“Determinism is usually understood to preclude free will because it entails that humans cannot act otherwise than they do.” In the case of robots this means that they cannot react otherwise and will always allocate tasks the same way regardless of when the simulation is run. This can change of course when something in the system is changed.

“Deterministic techniques include numerical and classical methods such as graphical methods, gradient and hessian-based methods, derivative-free approaches, quadratic programming, sequential quadratic programming, penalty methods, etc. They also include graph-based methods such as blind/uninformed search and informed search methods.” [20]

3. Stochastic models

Stochastic models have a random probability distribution when allocating tasks which can be analysed statistically but cannot be always predicted precisely. The model is helpful when trying to provide optimal control to the system in tightly coupled domains.

“The principal advantage of these techniques is its promise of providing optimal control in tightly coupled missions with uncertain world dynamics and perceptions. Its drawbacks are, however, numerous: They are computationally very expensive; in most cases include some degree of centralization to compute joint policies, simulation results in very small sized worlds are not straightforward to carry to real-life missions; finding the optimal policies is an iterative process that can be slow and that may require repeated exposure to the environment in which the action takes place.” [29]

4. Auction models

Auction Models are inspired from real life economic negotiations in an auction where there is a seller and multiple bidders. Auction models that closely resemble their real-life counterparts are also referred to market based but the negotiation pattern can differ based on the algorithm. Still, the simplest auction model can be generalised as a three-step negotiation: [20]

- Step 1: a task is published to the agents by some entity, usually called auctioneer. In many occasions the auctioneers are the agents themselves.
- Step 2: agents suited to the tasks reply to the auctioneer with a bid on the task.
- Step 3: the auctioneer awards the task to some agent, after evaluating the received bids.

The main advantages of auction-based models are the simplicity of the algorithm.

It is very easy to implement in most scenarios but has advantages such as lack of fault tolerance due to overt reliance on a single agent. Another disadvantage is highlighted when trying to scale the system.

Schneider introduces a form of learning via opportunity cost. This opportunity cost reflects the expected earnings per second of a robot. This is used to modify the auction mechanism, so robots try to maximize this opportunity cost. This has applications in time-discounted environments, where tasks lose value as time goes by. [37]

5. Behavioural models

Behaviour based models embed behaviour patterns into agents which are enabled or disabled in response to certain stimuli. These behaviours, when activated, will motivate/influence on robot to perform certain actions. Several behaviours can be active at the same time, but they must have rules regarding prioritization.

Two critical elements of Behaviour models are impatience –a robot gets impatient if he sees a task that nobody is executing, thus triggering its adequate behaviour to perform it– and acquiescence –that makes a robot to relinquish a task if it detects that its performance is below expectancy's. [29]

2.3.3 Taxonomy of MRTA

To correctly identify what approach our scenario MRTA problem can be solved by, we need to analyse the different components of our MRS. One way to do this is using taxonomy method to classify the components. The most widely used taxonomy method used in the MRS is by Gerkey and Matiric. [13]

They propose a taxonomy which is based on different component axes.

- Single-task robots (ST) versus multi-task robots (MT): ST means that each robot is capable of executing at most one task at a time, while MT means that some robots can execute multiple tasks simultaneously.
- Single-robot tasks (SR) versus multi-robot tasks (MR): SR means that each task requires exactly one robot to achieve it, while MR means that some tasks can require multiple robots.

- Instantaneous assignment (IA) versus time-extended assignment (TA): IA means that the available information concerning the robots, the tasks, and the environment permits only an instantaneous allocation of tasks to the robots, with no planning for future allocations. TA means that more information is available, such as the set of all tasks that will need to be assigned, or a model of how tasks are expected to arrive over time.

A 3D visual representation of the axes can be seen in Figure-6. [23]

"The goal of these axes is to show how various MRTA problems can be positioned in the resulting problem space and to explain how organizational theory relates to those problems and to propose solutions from the robotics literature. When designing a multi-robot system, it is essential to understand what kind of task allocation problem is present in order to solve it in a principled manner." [13]

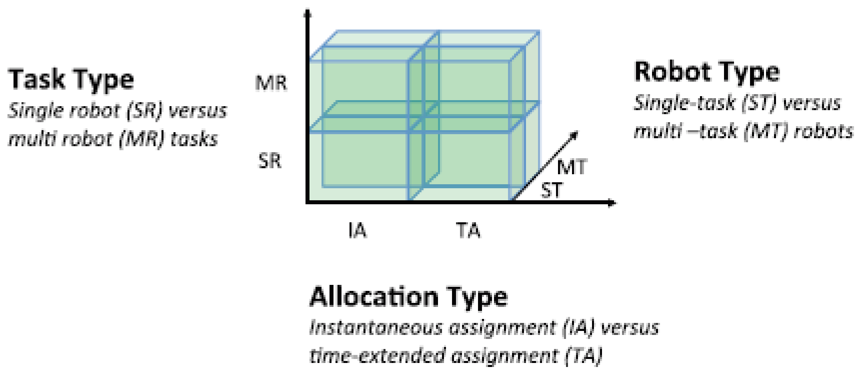
Task Type
Single robot (SR) versus multi robot (MR) tasks
MR
SR
Robot Type
Single-task (ST) versus multi –task (MT) robots
MT
ST
IA
TA
Allocation Type
Instantaneous assignment (IA) versus time-extended assignment (TA)

Figure 6: Visual representation of the three axes of Gerkey and Mataric's taxonomy.

Using the components, a MRTA problem can be denoted by a triple of two-letter abbreviations. For example, a problem in which Single-robot tasks must be allocated once to single-task robots in an instantaneous manner, the MRTA problem can be designated by ST-SR-IA. Using these axes, different types of MRTA problems and the algorithms can be classified and studied.

- ST–SR–IA is the simplest, as it is actually an instance of the optimal assignment problem and is the only problem in this space that can be solved in polynomial time. All the remaining problems are of a NP Hard Nature.
- ST–SR–TA: "When the system consists of more tasks than robots, or if there is a model of how tasks will arrive, and the robots' future utilities for the tasks can be predicted with some accuracy, then the problem is an instance of ST–SR–TA".

This problem involves determining a schedule of tasks for each robot which makes it an instance of a scheduling problem. [4]

- ST–MR–IA: "Many MRTA problems involve tasks that require the combined effort of multiple robots. In such cases, we must consider combined utilities of groups of robots, which are in general not sums over individual utilities; utility may be defined arbitrarily for each potential group. For example, if a task requires a particular skill or device, then any group of robots without that skill or device has zero utility with respect to that task, regardless of the capabilities of the other robots in the group." [13] This kind of problem is significantly more difficult than the previously two MRTA problems, which were restricted to single-robot tasks. "In the multi-agent community, the ST–MR–IA problem is referred to as coalition formation."

- ST–MR–TA: "The ST–MR–TA class of problems includes both coalition formation and scheduling. For example, consider the problem of delivering a number of packages of various sizes from a single distribution centre to different destinations. The number of packages and their destinations are known in advance, as is the size of each package, which determines the number of robots required to carry it. Given a pool of robots, the problem is to build a delivery schedule for the packages, while guaranteeing that a team of the appropriate size is assembled for each package." [13]

- MT–SR–IA and MT–SR–TA: both the problems are currently uncommon, as they assume robots that can each concurrently execute multiple tasks. "Today's mobile robots are generally actuator-poor. Their ability to affect the environment is typically limited to changing position, so they can rarely execute more than one task at a time. However, there are sensory and computational tasks that fit the MT–SR–IA or MT–SR–TA models quite well." [13] Solving the MT–SR–IA problem is equivalent to solving the ST–MR–IA problem with the robots and tasks interchanged in the formulation. Likewise, the MT– SR–TA problem is equivalent to the ST–MR–TA problem. Thus, the analysis for the multi-robot task problems also directly apply here to the multi-task robot problems.

- MT–MR–IA: When a system consists of both multi-task robots and multi-robot tasks, the result is an instance of the MT–MR–IA problem. "In the MT-MR-IA problem, the goal is to try to compute a coalition of robots to perform each task, where a given robot may be simultaneously assigned to more than one coalition (that is, a robot may work on more than one task). This problem can be expressed as an instance of the set-covering problem in combinatorial optimization. It is distinguished from the set-partitioning problem in that the subsets of robots need not be disjoint.” [23]

- MT–MR–TA: is an instance of a scheduling problem with multiprocessor tasks and multipurpose machines. This problem is extremely hard to solve it is a Hard case of the NP Problem. Gerkey and Mataric mention that they are not aware of any heuristic or approximation algorithms for this difficult problem. But a modified taxonomy approach by Korsah et al disagrees and presents a way to solve the problem. [23]

Gerkey and Mataric also discuss some important MRTA problems that are not captured by this taxonomy.

- Interrelated Utilities: they mention the problem of assigning target points to a team of robots that are cooperatively exploring an unknown environment. “Many targets maybe known at one time, and so it is possible to build a schedule of targets for each robot. Unfortunately, this problem is not an instance of ST–SR–TA, because the cost for a robot to visit target C depends on whether that robots first visits target A or target B.” Instead, this problem is an instance of the multiple traveling salesperson problem (MTSP); even in the restricted case of one salesperson, MTSP is strongly NP-hard [3]. If, as is often the case with exploration, it is possible to discover new targets over time, then the problem is an instance of the dynamic MTSP, which is clearly at least as difficult as the classical MTSP.
- Task Constraints: In addition to an assumption of independent utilities, our taxonomy also assumes independent tasks. There may instead be constraints among the tasks, such as sequential or parallel execution. In principle, each set of tasks with such constraints could be phrased as a single monolithic task that requires multiple robots. The allocation of these larger tasks could then be described by the presented taxonomy (e.g., ST–MR–IA). Unfortunately, the difficult problem of reasoning about task constraints is not removed, but simply shifted into the utility estimation for each potential multi-robot team. Thus, the taxonomy will not suffice in the presence of constraints among tasks.

2.3.4 Challenges of MRTA

There are several properties that are desirable in any task allocation algorithm. Due to the nature of the discussed scenario problem, not all of the properties can be achieved so some sacrifices have to be made. The following are the most important properties when accessing an allocation algorithm:

- Mathematical Soundness: a tried and tested mathematical theory is always better to have in an algorithm. This allows us to find quality bounds, known expected time of computation, rate of convergence to solution, and so on. [13]
- Distributability: the capability to disseminate the work among the most capable devices to allow optimal usage of the available resources will produce faster overall results in a system. [11]
- Decentralization: the elimination of centralization directly prevents bottlenecks and provides fault tolerance and resiliency to the system. Making roles transient or transferable is another way of achieving decentralization. [28]
- Scalability: the property of a system to function properly with increasing number of agents is important in MRS. Good scalability principles in our scenario MRTA algorithm is of a highest priority. One way to achieve scalability is by means of locality: only locally available or nearby reachable information is needed for an agent decisional process. Local algorithms have the advantage of putting less stress in the network and are in general more scalable. [19]
- Fault Tolerance: the ability to operate under partial failures is an important property and is linked with the scalability of the system. Due to the nature of our scenario, fault tolerance is another key property. One way for achieving fault tolerance in a scalable system is seamless integration and removal of agents in the system. [22]
- Responsiveness: the ability of the system to be responsive and achieve results after changes are introduced in the working system or mission conditions is important and is also directly related to the performance of the overall system. [6]
- Flexibility: sometimes different missions in a system environment can be performed by flexible robots. Flexibility is usually a bonus but for some scenarios can be important. [36]

2.4 Task Allocation Algorithm Models

In the previous chapters, we discussed the major background information needed to understand the problem scenario, the next step is to find a way to select the most optimal algorithm. This chapter will discuss the different models which can be used to narrow down the search for algorithms.

Jia and Meng discuss two models inspired from the MRTA taxonomy, Centralization/Decentralization, Homogeneity/Heterogeneity and Optimization objectives. [17, 13, 25]

2.4.1 Relevant Concepts

Before discussing the models, some background information about the model components is required. This chapter discusses this information.

1. Optimization Objectives of MRTA Algorithms

Every Algorithm needs to have an optimization objective which depends on the scenario of the problem. These objectives are discussed in this section with their advantages and disadvantages. [29]

- Minimize costs of the worst agent, also known as the MINMAX criterion. This criterion is of interest in time critical missions since it gives the shortest mission execution time-span. Its focus is to optimize the execution of the worst performing agent. [27]
- Maximize utility of the worst agent, also known as the egalitarian criterion in welfare related studies, is the dual optimization of the previous one. In this case we try to optimize the worst performing agent. [30]
- Minimize the sum of individual costs, also known as minimization of the MINSUM. It is relevant in efficiency contexts, for example optimization of fuel usage in transportation or delivery situations.
- Maximize the sum of individual utilities, dual of the previous one, is especially interesting in contexts where finalization of tasks gives some reward, since it gives the best possible allocation in this respect. It is usually found in market-based approaches because of the direct translation of economic profit characteristics. [39]

- Minimize the average cost per task, or latency when cost equals time. (MINAVE) This objective measures the average time since a task appears in the system until it is completed. It has relevancy in domains where task completion is more important than aggregated global costs, e.g. in rescue situations where time to victim location may be critical.

They also mention some effects that need to be considering when using some of these criteria.

- the MINMAX objective is only concerned with the worst agent, and hence all costs below this threshold are "hidden" in the final result, unless somehow included in the optimization. By using a pure MINMAX optimization process, grossly under-performing behaviours below the critical one may occur.
- conversely, in a pure MINSUM optimization, the bulk of the work may fall on a single agent, even if there is a number of idle robots. These issues, its root causes not always explicitly identified, are circumvented with additional elements in the cost functions.

2. Decentralization and Centralization

The multi-robot systems can be classified as centralized and decentralized (with and without a temporal agent) systems in terms of resource management. Jia and Meng discuss the differences, advantages, disadvantages of these types of algorithms. [17]

- In centralized multi-robot systems, a single element is responsible for managing all the available resources. On one side, one of the biggest advantages of these centralize algorithms is that they usually have more information available than decentralized methods, thus making it much easier for the system to find globally optimal solutions. On the other side, since each robot needs to compute its allocation with the knowledge of other robots by communicating with the centralized planner, a single point of failure will be introduced once the communication gets lost. Therefore, the centralized multi-robot system is not quite favourable for scalability and fault tolerance.
- In decentralized multi-robot systems, as it does not require a central controller or enough bandwidth to transmit all the information, the algorithms become more

scalable than the centralized ones. Moreover, the decentralized systems are more robust and flexible because there is no need to know the global topology and a failure does not compromise the whole system. We can further divide the decentralized algorithms into two categories according to whether or not the system possesses a temporal agent. An algorithm with a temporal agent still has some centralization as the agent acts as a leader for an agent group and in a big system, there will be many agent groups with these agents. If the temporal agent is replaceable by any other agent, the algorithm can become fully decentralized.

- The other kind of decentralized algorithms is what we call the fully decentralized algorithm. These algorithms do not need an agent and only involve local information and local communications. Under this approach we use multi-task selection instead of multi-task allocation because the agents or robots select the tasks instead of being assigned a task by a centralized planner or temporal agent. In particular, we follow the definition of a decentralized system given in [31]:
  - There is no central agent required for the operation.
  - There is no common communication facility; that is, information cannot be broadcasted to the whole team, and only local point-to-point communications between neighbours are considered.
  - The robots do not have a global knowledge about the team topology, and they only know about their local neighbours.

3. Homogeneity and Heterogeneity

As the name suggests, homogeneity represents same characteristics and heterogeneity represents different characteristics. In terms of robots, a homogeneous MRS will only consist of the same robots i.e. all the robots in the system are UGV‘s with the same purpose of navigating and mapping a forest. While, a heterogeneous MRS can consist of different robots with different capabilities such as UAV‘s and UGV‘s together mapping a forest.

4. Task Constraints

An allocation algorithm sometimes has to accommodate tasks that are restrictive in nature which can cause problems in the algorithm's execution. Problems may exhibit only a subset of these restrictions, simplifying the necessary framework. The most common of these constraints are as follows:

- Partial ordering, in which a task must be completed before or after a set of others (but not necessarily immediately so, since in this case there is no planning flexibility). An example would be opening a door before another task can be accomplished in another room.
- Time windows, in which a task must be completed in a given time frame, or before a certain deadline.
- Coupling, in which two or more tasks must be executed at the same time (e.g. in coordinated box-pushing of wide objects).
- Incompatibility, in which executing one task may preclude or obsolete the execution of others.

2.4.2 Task Model

As mentioned before, Jia and Meng propose two models based on different parameters. The first of these Models is called the Task Model. It combines the MRTA taxonomy, Centralization/Decentralization, Homogeneity/Heterogeneity together to filter out the algorithms.

As we know the criteria of our scenario already, we can further focus on just these factors.

- Algorithm follows the decentralized topology. It can have a temporal agent as long as that temporal agent is replaceable by any other robot in the system, although being completely decentralized is ideal as long as the key conditions are met.
- Suitable for heterogeneous robots, the algorithm should be able to work around different types of robots with different capabilities. If possible, it should also support task assignment for heterogeneous sub-teams.
- Can be classified as a ST-(MR/SR-IA) MRTA category. As the robots in the scenario don‘t have any need or capability in some instances to perform multi-tasking, they will be limited to handing a single task at a time. The tasks may also require multiple robots to complete in some cases so the robots may require the capability to perform the same task together. Finally, as the information that will be available to the robots is obtained real-time in most cases, instantaneous assignment should be possible.

2.4.3 Solution Model

The solution Model is based on the Optimization objectives in the MRTA algorithms which are discussed in detail in section-2.4.1. Some important objectives discussed by Jia and Meng are the MINISUM, MINIMAX, MINIAVE. For our scenario the most relevant of these objectives is MINIAVE which is short for minimizing the average costs per task or latency when costs equal to time.

"This objective measures the average time since a task appears in the system until it is completed. It has relevancy in domains where task completion is more important than aggregated global costs, e.g., in rescue situations where time to victim location may be critical." [17] As our scenario deals with extinguishing fires and rescuing humans, task completion is far more important than the final cost.

2.4.4 Ideal Algorithm

Using the two models defined in previously we can classify the ideal scenario algorithm to have the following parameters. This classification will assist in selecting the most applicable algorithms in section-2.5.

- Decentralized
- Heterogeneous
- ST- MR – IA
- MINIAVE

2.5 Task Allocation Algorithms

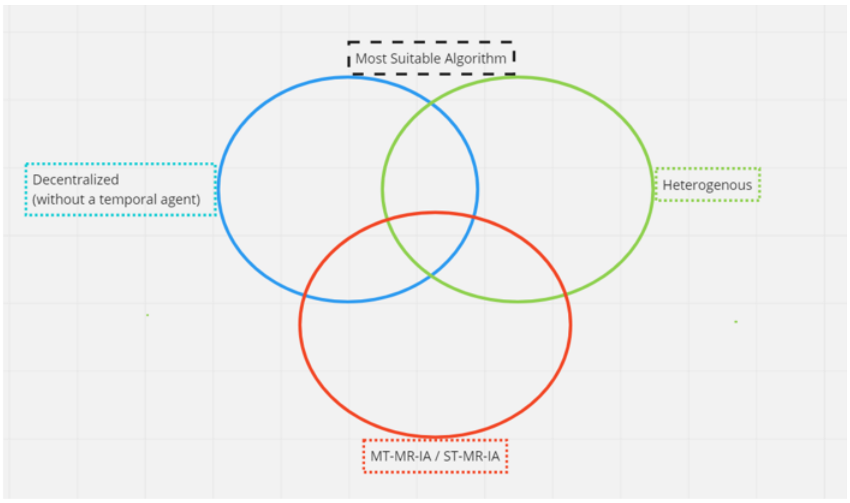

Figure 7: Venn Chart to depict the ideal algorithm.

Following the review of different algorithms using the parameters discussed in section-2.4, three algorithms are proposed. In this section only a high-level summary is presented, more detail will be discussed in the next chapters.

2.5.1 Distributed Bees Algorithm

The Distributed Bees Algorithm is a modified decentralized version of the original Bees algorithm. The Bees algorithm is inspired by real life behaviour of bees in nature. When the bees are foraging for food, the scout bees will head out and survey different areas and come back to the hive with their results. Depending on the quality or quantity of food found, they will waggle dance in front of other bees. The duration of this dance reflects the quality of the food. [33]

This same bio-model can be applied to robots without the limitation of the communication methods. For example, in our scenario, scout robots can survey the area to find out possible targets and communicate distance and quality of the target to the working robots.

Jevtic et al defines their algorithm suitable for Homogeneous, Distributed and ST-MR-IA solution. The task allocation scenario they define considers the environment to contain different number of tasks with equal or same importance and robots that are equally capable of performing each task but can only be assigned to one task at any given time. Each task has an associated quality which is a specific-scenario scalar value the represents the tasks priority or complexity where higher value requires more robots to be allocated. Each task also has a cost associated with it which is dependent on the Euclidean distance of any robot to the target location. This cost and quality are used to calculate the utility of the task where higher quality is good. [16]

2.5.2 S+T Algorithm

The S+T algorithm is base on a distributed market-based approach with temporal agents and could be considered an extension of the SIT algorithm.[42] The algorithm modifies the algorithm by introducing services with the idea of asking for help. For example, if the robot cannot execute a task by itself, it will ask for help and if possible, another robot will provide the required service. The services are generated dynamically and dependent on the robots and the system. [43]

They also present two different approaches for the algorithm. In the first one, services are only considered when none of robots can execute any particular task. In the second approach the tasks can be optimized for the execution time and the energy consumption of any particular tasks.

The modified algorithm is suitable for Decentralized, Heterogeneous and ST-MR-IA solutions. The author also mentions the disadvantage of the algorithm as overhead in the allocation process but can be solved by modification.

2.5.3 Decentralized Hungarian Method

The Decentralized Hungarian Method is a modification of the original Hungarian method which uses cost matrix's for graph networks to find the most optimal task assignment. [24]

A decentralized version of this algorithm is proposed by Giordani et al, which consists of many decision makers as the robots in the system. The robots autonomously perform the different sub steps of the Hungarian algorithm using the individual and shared information received from other robots in the system. The author assumes that each robot is aware of its distance from the targets in the environment. This inter-robot communication network is used to allow dynamic communication and find the optimal solution without using a centralized memory with the downside of the communication costs. [14]

The modified algorithm is suitable for Decentralized, Heterogeneous and ST-SR-IA solutions. The author also gives the computational cost of the algorithm with the global optimum solution in O(n3) cumulative time (O(n2) for each robot), with O(n3) number of messages exchanged among the n robots.

2.6 Gap Analysis

Most of the artefacts that were reviewed sufficiently address the high-level information that is related to our scenario. However, when it comes to the specific algorithms, it is difficult to find an algorithm that matches all the requirements of our scenario. More work can also be put in how to identify and filter out the algorithms depending on the scenario. There were few artefacts that aimed to properly address the system on how to decide the algorithm, most sources aimed to categorize and analyze the differences between them.

The key challenge is going to be how to modify the selected algorithm(s) for our scenario and how to negate the downsides of decentralization while keeping the advantages. It is also important that these algorithms can not only be tested in a simulation but also be viable in a real-world scenario.

3 Research Design & Methodology

3.1 Research Questions

As discussed in section-2.6, the gap analysis highlights the current gaps and challenges for task allocation related to our specific project scenario. In this section the aim is to formulate research questions that are relevant, specific, and answerable within the scope of this research using the literature review and gap analysis as the primary sources.

The refined research questions should be:

- Focused on a single problem/ issue/ topic.
- Answerable using the current technologies and knowledge available.
- Feasible to answer within the timeframe and the practical constraints of this project.
- Specific and easy to understand. The questions should not be vague.
- Relevant to the main project scenario and the answer should help answer the project defined problem to an extent.

Following are the research questions along with their description and their purpose in relation to the project scenario.

RQ1: How do the selected algorithms perform task allocation in relation to the defined evaluation constraints ?

- Description: The question will be the central focus of this project and the experiment methodology and design is influenced to answer this question. The task allocation algorithms discussed in section 2.5 of the literature review will be tested and evaluated using the defined evaluation constraints. These constraints are defined in section 3.2. These evaluation constraints will be used to separate the algorithms and select the best suited algorithm for the scenario problem.
- Purpose/Motivation: As mentioned in the project summary in section 1, "The algorithms which will be tested as a part of this project should allow the swarm to autonomously carry out multiple tasks such as extinguishing fires and retrieving people that are trapped behind the fire front in parallel." The motivation is to compare the algorithms which were selected using the primary resources discussed in the literature review and further narrow down these algorithms to find out if any/which of them is viable to solve the project scenario.

RQ2: How to optimize the advantages and disadvantages of the selected decentralized task allocation algorithms in relation to the project scenario ?

- Description: This question can be considered a secondary question which is related to the primary question. While the first question tries to evaluate which algorithm is the most optimal for the defined constraints, this question discusses how the algorithms suffer from the challenges of MRTA discussed in section 2.3.4 and how the disadvantages and advantages should be optimized and prioritized.

- Purpose/Motivation: One of the challenges mentioned in the project scenario was the cost of communication while operating in regional areas. This is one of the challenges that are considered a priority of our project and should be optimized. But as mentioned in section 2.3.1, the problem of MRTA is of a NP-hard nature so compromises need to be made if the solution is to be practical. A practical solution is vital to this research thus the challenges need to be optimized with respect to different algorithms.

RQ3: Which algorithm is best suited to solve the scenario problem?

- Description: This question is related to the other main questions and can be answered by evaluating the results of the both of them. Evaluation scores found in the experiment are combined with the advantages and disadvantages to propose the most suited algorithm for the project scenario.
- Purpose/Motivation: The other questions gave partial answers to the main problem of evaluating the best possible allocation algorithm. This question tries to combine the results found in the experiments and propose a single algorithm which is best suited to the scenario problem.

3.2 Experiment Methodology and design

An experiment design philosophy needs to be defined to answer the research questions. This design methodology will decide how the algorithms will be tested and what type of data will be collected to analyse. The analysis of this data will be used to present clear and concise answers to the questions.

3.2.1 Experiment Methodology

The experiment is mostly focused on a simulation model which will represent a 3D space to represent the real-life world. This simulation model will consist of various targets i.e. tasks and agents i.e. robots. The number of tasks and robots will be constant to keep the results consistent. The tasks will also be heterogeneous and will require robots with some specific skills to accomplish them. Similarly, the robots will also be heterogeneous and will have different skills and limitations.

The core philosophy about the simulation experiment is to compare the algorithms under constant parameters. The comparison metrics are covered in the data collection section of this chapter. These comparison metrics can be multiple and eventually will decide the feasibility of using any algorithm for our specific scenario.

3.2.2 Experiment Design

This section discusses the simulated environment design which represents a hypothetical remote region with different comparable zones to the real world. These zones have different characteristics which influences the task allocation directly. The possible tasks are also discussed and how they are received is mentioned. At the end the details about the robots in the system is discussed. This experiment mostly helps to solve the first research question.

1. Simulated Environment Design

As mentioned in the previous sections, the algorithms will be tested in a 3D Simulated square space. To give some context to this 3D space, it is divided into 4 equal square grids with each gird/zone having different environments which can have different possible tasks. The different zone in the space are as follows:

- Zone-1 is a Major town which is located in this hypothetical 3D space. This town acts as the main population hub of a regional area. The amount of flora in this area is the least among any zones but due to the human factor, fires can still occur or can be spread from other zones. Possible tasks in this zone are rescuing and controlling the fires. Due to the population of the zone, the priority of these tasks is maximum, and the robots will be allocated the tasks accordingly.
- Zone-2 is a Farmland of the town located in this hypothetical 3D space. This farmland is known for growing rare fruit varieties and acts as a major economic asset to the town. It has the 2nd highest population density in this hypothetical region. Possibility of fires are moderate but can occur due to the proximity of nearby zones. The priority of the tasks in this zone depends on fires in other zones but will be below zone-1 in any case generally.
- Zone-3 is part of a National park located in this hypothetical 3D space. This zone contains rare flora and fauna which helps the region to attract visitors. The fauna

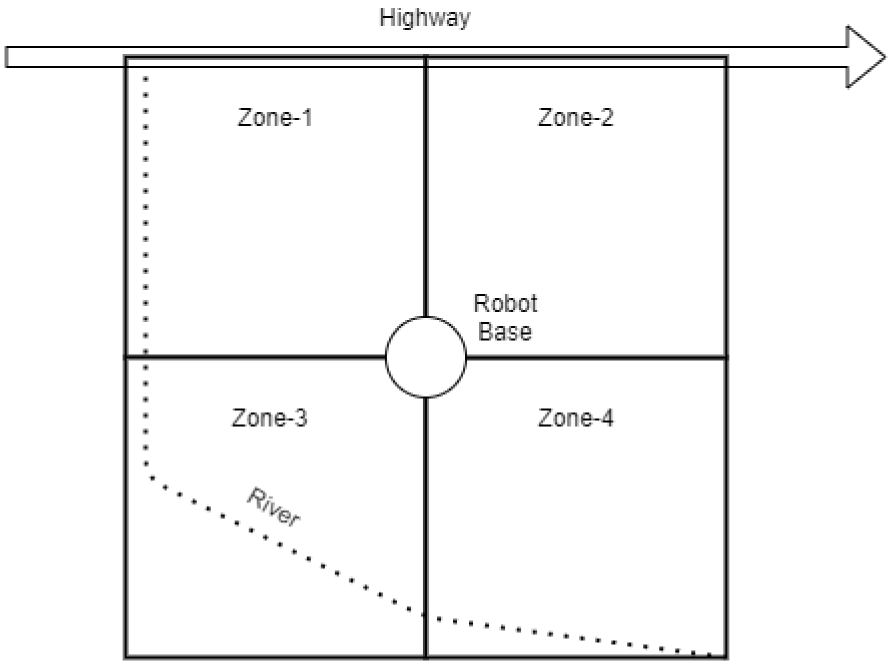

Figure 8: Representation of the zones in the simulation.

in this region is the most fire prone of any other zones. The zone is the least populated of the other zones. The priority of the tasks in this zone depends on fires in other zones but is below zone-1,2 generally.

- Zone-4 is part of a National park located in this hypothetical 3D space. This zone has historical significance to the region and has flora that is highly prone to fires. This zone is used as a camping hub for tourists and natives alike and has the 3rd highest population density in this hypothetical region. The priority of the tasks in this zone depends on fires in other zones but is below zone-1,2,3 generally.
- Robot deployment base is the location from where the robots are deployed. For the sake of simplicity, the base is located centrally on the intersection of all the other zones.

These zones will be discussed in detail more in the artefact development chapter later.

2. Possible Tasks / Targets

The type of tasks that will be allocated in this simulated environment are as follows:

- Extinguishing and controlling fires

  - Aerial extinguishing and diversion methods.
  - Ground extinguishing and diversion methods.

- Rescuing operations
  - Aerial pickups
  - Ground pickups

The experiment will use different sets of tasks to test for the scalability and performance in the algorithms. These sets will be constant for each test. The sets of robots are as follows:

- Set-1: 5 Tasks, 1 aerial extinguishing task, 3 ground extinguishing tasks, 1 ground pickup.
- Set-2: 10 Tasks, 1 aerial extinguishing task, 1 aerial rescue pickup, 6 ground extinguishing tasks, 2 ground pickups.
- Set-2: 15 Tasks, 2 aerial extinguishing tasks, 1 aerial rescue pickup, 9 ground extinguishing tasks, 3 ground pickups.

For this stage in the experiment the number of tasks and robots are same to make evaluation simple but this will vary in the future design of the task when the simulation model is deemed acceptable for testing and evaluation.

It is important to note that tasks such as locating, and reconnaissance of tasks will occur in background. For the sake of simplicity, how these tasks are found will not be discussed at this stage of the experiment and it is assumed that they are added into the possible tasks list.

3. Robots in the system

The experiment will use different sets of robots to test for scalability and performance in the algorithms. These sets will be constant for each test. The sets of robots are as follows:

- Set-1: 5 Robots, 1 aerial robot capable of surveying, rescuing and firefighting; 4 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.

- Set-2: 10 Robots, 2 aerial robots capable of surveying, rescuing and firefighting; 8 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.
- Set-3: 15 Robots, 3 aerial robots capable of surveying, rescuing and firefighting; 12 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.

Depending on the performance of a simulated robot in the testing environment, the sets can be increased to have higher number of robots to better represent real-life scalability situations.

For our hypothetical scenario these robots are housed in the central base but are deployed regularly to look for targets and add them in the tasks list. This is presumed to occur in the background is out of scope for the purposes of this experiment.

3.3 Data Collection

To Confidently answer the primary research questions, some metrics need to be defined on which we will access the selected algorithms. Some of these metrics are more important in determining a specific criteria than the others and that will be mentioned if it applies. These metrics are as follows:

- The time taken to allocate all the tasks: also called global run-time, helps to access if the algorithm matches the expectations. As there is no pre-defined time constant, the results of each algorithm will be compared to each other to find the minimum run-time. For some algorithms, the mathematical run-time is already defined and that can help to access whether the algorithm is performing as expected. The value is stored in seconds.
- The time taken to allocate a specific task: as the experiment parameters are constant for all the algorithms, a comparison can be made among the algorithms to see the difference of allocating a specific task. This metric can help access if the algorithm is putting priority into a high-importance task such as rescuing a human over firefighting. The value is stored in seconds.

- Is reallocation performed: this metric tests if any task reallocation was performed. Reallocation might be good or bad depending on the circumstances. If the parameters are defined in a way, that reallocation is expected and performed, then that algorithm will satisfy this metric. The value is stored as true or false.
- If reallocation is performed, time taken to reallocate: this metric measures the time taken to reallocate the specific algorithm. This metric is important as it reflects on a algorithms ability to be flexible. In our test environment a new high priority task can arrive in the queue, in which case reallocation is necessary. The value is stored in seconds.
- Are the tasks allocated to the most suitable robot?: this metric measures the effectiveness of an algorithm to allocate the tasks to the most capable robot. A perfect allocation is not to be expected as other factors such as the time to travel to the task location can mitigate the downsides of a less skilled robot if it is much closer to the target. Similar to the global run-time metric, it will also compare the percentage of suitable allocations for all the algorithms. The value is stored as a percentage.

A combination of these metrics will decide the most suitable algorithm.

3.4 Experiment Results

For the first research question, the experiment and the relevant data collected is sufficient to answer the question. A sort of table will be produced which makes the comparison among all the metrics for all the algorithms. Using this table, a conclusion can be made which will answer the first question.

The second question will be answered using the experiment metrics and the information researched in the literature review for each algorithm on how each handles the challenges of MRTA. Some of the important challenges can be answered using practical results from the experiment and the other less important areas will be discussed briefly.

3.5 Experiment risk and error Mitigation

Discussion about project risks, how they could impact the success of your project, and how you intend to mitigate these risks.

To prevent errors from misleading the experiment results a risk mitigation strategy is important. Some problems which can lead to errors can be differences in the experiment parameters when testing different algorithms, unexpected background processing times of the simulation environment can affect the run-time of different allocations and need to be accounted for. To prevent such errors, following precautions will be made:

- Each algorithm will be tested under the same conditions, i.e. same number and type of tasks, same number and type of robots and constant environment simulation.
- During data collection of time related metrics, background computational processing needs to be considered and monitored. If significant and variant processing times between the algorithms is observed it needs to be mentioned in the result. Due to the nature of the experiment, variance cannot be removed entirely and will always influence the times in some way. If the processing times affect all the algorithms then a safer conclusion can be made but if the processing time affect a specific algorithm, making a definite conclusion will be difficult if the times are close. Using other metrics will be helpful in determining the score in this case.
- To limit the background processing, the simulation environment will use minimal number of graphical components to put more focus on the allocation.
- An algorithms performance depends on the number of agents in the system. For some algorithms , low number of agents produce better results but as the number increases, the performance can differ. If there is a significant difference in how an algorithm behaves in a low agent system, the conclusion should reflect the same.
- Before testing the full scenario for each algorithm, initially limited tasks will be introduced to see if the algorithm is performing as expected. This is especially important if modifications need to be made to some algorithms.

4 Artefact Development Approach

In this section, the details of the artefact development plan and approach has been discussed. The development environment/tools are discussed with their relevance in the creation of the artefact. Scenarios are presented for the suitable algorithms that were discussed in the literature review. Initially only a version of the bee's algorithm is presented along with code structure of this artefact. Experiment process and procedure is also discussed and how the data will be collected to conduct validation and analysis.

4.1 Development Tools

The tools needed in the creation of the artefact simulation were chosen to be easy to use and implement. Python code is used to implement the prototype algorithms. For every algorithm that will be tested, there are some common classes in the code to represent the experiment parameters. These parameters are the robots, tasks, zones for the simulated area. Within these base classes, the code will also include the data structures that are needed to evaluate the algorithms and answer the research questions. To visually demonstrate the simulation and its components, gazebo will be used in the future.

4.2 Artefact Structure

In this section different artefact components are presented and discussed which together form a centralised version of the Bees algorithm discussed in section-2.5. The algorithms will have some common parameters such as robots, tasks etc. To represent these parameters base python classes are created which are used in the development of the main algorithm.

For the development sprint-1, a single scenario is developed which is based on a centralised version of the Bees Algorithm. Although it is not the ideal way to solve our scenario problem, it will act as the starting point to create a decentralised version in the next project sprints. It will also help to demonstrate the differences between the two versions and the reason a decentralized approach is important for this project.

4.2.1 Robot Class

This class is used to represent the functioning of a robot agent in the simulation. As mentioned in the experiment design section, these robots are primarily aerial and ground based at this stage of the project. Each robot object has the following characteristics:

- robotID = Used to identify the robot.

- robotType = Used to identify the type of the robot.
- robotLocation = Used to store the 3D coordinates of the robot in the simulation environment.
- taskAssigned = Used to keep track of the current task assigned to the robot.
- isBusy = Used to ascertain whether the robot is busy completing a task or navigating.
- isCapable = Used to define the tasks that this robot can complete.

The object members are used in various ways in forms of methods and properties in the development of the main centralised algorithm.

The code for the base class Robot can be found in section-9.1

4.2.2 Task Class

This class is used to represent the task or target available in the system. The tasks at this stage are simple and include aerial and ground versions of fire extinguishing, diversion and rescue pickups. Each task object has the following characteristics:

- taskID = Used to identify the task in the simulation.
- taskDescription = Used to describe the task and its requirements.
- taskQuality = Used to assign a arbitrary scaler value the is used to define the quality/priority of a task. For the purposes of this experiment, the value can range between 1 to 10 with a task of 10 holding the highest priority.
- taskRelativeQuality = Used to calculate the quality of the task relative to all the tasks quality. This value makes it easier to relate the priority of the tasks and calculate the final utility value.
- taskLocation = Used to store the 3D coordinates of the task in the simulation environment.
- taskType = Used to define the nature of the task, i.e ground firefighting, aerial rescue etc. This property of the task is useful in determining which robot can be allocated to it based on it capabilities.

- taskAllocated = Used to check if the task is allocated to the robots.
- timeAdded = Used to store the time when this task first appeared in the task queue.
- timeAllocated = Used to store the time when this task was allocated.
- robotAllocated = Used to store which robot is allocated to this task.
- timeCompleted = Used to tore the time when this task was completed. As the focus for this experiment is on allocation, this value doesn‘t test the robots ability to complete the task but is instead used to keep track of the task completion and remove it from the queue hence it is a arbitrary value. It is calculated by multiplying the tasks scalar quality to the tasks initial distance from the robot assigned.
- isReallocated = Used to check if the task was reallocated.
- timeReallocated = Used to store the time when this task was reallocated.To avoid the problem of looping re-allocations, the experiment design at this stage only allows the task to be reallocated once.

The object members are used in various ways in forms of methods and properties in the development of the main centralised algorithm.

The code for the base class Task can be found in section-9.2

4.2.3 Coordinate Class

This class is used to represent the coordinates of the robots and tasks in our simulated 3D environment. The coordinates are represented in terms of (X,Y,Z). Each coordinate object has the following characteristics:

- X = Used to store the X-axis coordinate of the object.
- Y = Used to store the Y-axis coordinate of the object.
- Z = Used to store the Z-axis coordinate of the object.

The object members are used in various ways in forms of methods and properties in the development of the main centralised algorithm.

The code for the base class Coordinate can be found in section-9.3

4.2.4 Centralised Bees Algorithm

To evaluate the artefact design and its effectiveness in answering the research questions, a centralised version of the Bees algorithm was developed as the main artefact. This algorithm uses the code from the base class to create the relevant objects. The properties of these objects are manipulated using methods to ultimately find the allocations.

The main goal of the algorithm is to calculate the task allocations based on the quality of the tasks, distance between the task and robot sets i.e based on the utility values of the possible allocation sets.

The methods, variables and other code of the algorithm are explained in the below sections:

1. create_robots(set):

This function is used to create the different sets of robots. The set parameter is used to check the quantity and quality of the robots. For the sprint-1 only a single set is considered which is mentioned in the ”Robots in the system” part of the section-3.2.2.

The function uses static default variables to help in the creation of the robot objects. These static variables are as follows:

- ”TYPE_GROUND” and TYPE_AERIAL variables contain a string used to define the type of robot that is being created.
- BASE_LOCATION_X, BASE_LOCATION_Y, BASE_LOCATION_Z variables contain the x,y,z coordinates of the main robot base where all robots start from. These coordinates are used in the BASE_COORDINATES to create a set of coordinates using the coordinate base class.
- TASK_LIST_GROUND and TASK_LIST_AERIAL variables contain the list of capable tasks a particular robot type can accomplish.

Using the static variables as parameters for the robot object constructor, the robots are created as defined in the set definition.

2. create_tasks(set):

Similar to the function to create the different sets of robots, this function creates a set of tasks based on the set parameter. For the sprint-1 only a single task set is considered.

The function uses static default variables to help in the creation of the task objects. These static variables are as follows:

- TASK_AERIAL_FIREFIGHT, TASK_GROUND_FIREFIGHT, TASK_AERIAL_RESCUE and TASK_GROUND_RESCUE variables contain a string which is used to define the type of tasks a particular robot can accomplish.
- QUALITY_AERIAL_FIREFIGHT, QUALITY_GROUND_FIREFIGHT, QUALITY_AERIAL_RESCUE and QUALITY_GROUND_RESCUE variables contain a integer value which represents the quality of the task between 1 to 10.
- SAMPLE_COORDINATE_X SAMPLE_COORDINATE_Y SAMPLE_COORDINATE_Z variables contain the coordinates of the task which are used in the SAMPLE_COORDINATES to create a set of coordinates using the coordinate base class. For this development sprints a single sample coordinate is used for all the tasks created to easily test the other functionality of the algorithm.

Using the static variables as parameters for the task object constructor, the tasks are created as defined in the set definition.

3. calculate_distance(robot_coordinates, task_coordinates):

This function is used to calculate the distance of the robot from a task. It uses the formula for 3D Euclidean point to point distance. It returns the distance calculated using the formula.

4. calculate_visibility(distance):

This function is used to calculate the visibility of a certain distance set. Visibility is defined as an inverse of distance. It is important in the final utility calculation as it is used alongside the quality of a task to find the probability value of a task being assigned to the robot. The more visible a robot is, higher the chance of the task being assigned to it. It returns the visibility calculated.

5. check_capability_and_calculate_visibility():

This function uses the calculate_distance and calculate_visibility functions to calculate the distance and visibility sets of all the task to the robots. If a robot is not capable of doing a certain task, it is given a distance value of -1 and a visibility value of -1 in their respective array sets. This function returns a 2d list which contains the visibility of all the tasks to all the robots.

6. calculate_relative_quality():

This function is used to calculate the relative quality of all the tasks. Relative quality of any task is calculated using the following formula:

$$RelativeQuality_T = Quality_T \div \sum_{T=1}^{Tasks} Quality_T$$

where T represents the task for which the relative quality is being calculated.

7. print_relative_quality():

This function is used to print out the relative qualities of the tasks. It is mostly used for output and debugging purposes.

8. calculate_utility(task_robot_visibility_set):

This function is used to calculate the utility values for the sets of robots and tasks. The parameter task_robot_visibility_set passed contains the visibility values for the sets of robots and tasks which are calculated using the check_capability_and_calculate_visibility() function.

The absolute utility value of any task T for the robot R is calculated using the formula:

$$Utility_T^R = RelativeQuality_T \times Visibility_T^R$$

If the $Visibility_T^R$ is -1, it means that the robot was not capable of doing this task. This negative value helps in the final calculations to avoid invalid allocations.

This function returns a 2D list which contains the utility values for the sets of tasks and robots.

9. calculate_utility_probabilities(task_robot_utility_set):

This function is used to calculate the probability of a task to be assigned to a certain robot. The robot with the highest probability to do a task is assigned to it.

The probability of a Robot R to be assigned to Task T is defined as:

$$Probability_T^R = Utility_T^R \div \sum_{R=1}^{R} Utility_T^R$$

This function returns the 2D list which contains the probability values for the sets of tasks and robots.

10. print_utility_probabilities(utility_probabilities):

This function is used to print out probability values for the sets of tasks and robots. It is mostly used for output and debugging purposes.

11. assign_allocations(utility_probabilities):

This function is used to find out the final allocations decided by the algorithm using the results from other functions mentioned before.

It finds a free robot with the highest probability value of performing a certain task. When the robot is found, the function checks if the robot is not assigned to any other task and the task is not assigned to any other robot. If the conditions are met, the task is assigned to this robot and the function continues to allocate other tasks.

At the end of the function, the function returns the allocations for the task and robot sets.

At this development stage, only a single assignment is possible but in the future sprints multiple assignment will be developed into the code.

12. print_time_taken_to_allocate():

This function is used to print out the global and local run-time values for the algorithm. It is mostly used for output and data collection purposes.

The code for the Centralised Bees Algorithm can be found in section-9.4

5 Empirical Evaluation

This section discusses the experiment procedure, what data is collected and how the experiment results are evaluated qualitatively and quantitatively in respect to the research scenario and goals.

5.1 Experiment Procedure

To run the code developed, a simple terminal command is used to run the main start point in the Centralised Bees Algorithm which is named CBA.py in the artefact folder. "python CBA.py" , starts the algorithm.

For this stage in the development, the algorithm simply returns a text based output for evaluation. In the future sprints, Gazebo can be used to show this output visually.

The output printed by the algorithm when robots and tasks are of set-1 can be found in section-9.5

From the terminal output, we can see the steps the algorithm performed to find the final allocations. The last log where we get the local time and global time of the algorithm is particularly useful for the data section which is used to evaluate the artifact.

5.2 Data collection

For this stage in the development of the artefact, two of the five metrics discussed in the section-3.3 are logged and stored by the algorithm.

The metrics which are collected are as follows:

- The time taken to allocate all the tasks: also called global run-time, helps to access if the algorithm matches the expectations. As there is no pre-defined time constant, the results of each algorithm will be compared to each other to find the minimum run-time. For some algorithms, the mathematical run-time is already defined and that can help to access whether the algorithm is performing as expected. The value is stored in seconds.

- The time taken to allocate a specific task: as the experiment parameters are constant for all the algorithms, a comparison can be made among the algorithms to see the difference of allocating a specific task. This metric can help access if the algorithm is putting priority into a high-importance task such as rescuing a human over firefighting. The value is stored in seconds.

From the terminal output of the algorithm, the last section (line 550 onward) displays the time taken to allocate each task in the task set and the total time taken to allocate the tasks.

As mentioned in the metric definition above, these metrics help to make practical comparisons among the algorithms when two or more algorithms need to be compared for performance.

5.3 Qualitative Evaluation

In this section, the experiment is qualitatively evaluated based on the research scenario and goals. It will aim to answer whether the experiment accomplished what it was meant to do as mentioned in the research design section-3.2.

5.3.1 Task allocation based on parameters

The artefact developed supports the task allocation of a swarm of robots based on the cost model discussed in section-2.3.2. The experiment uses the cost model parameters to allocate tasks.

- The quality of the tasks used in the algorithm relate to the Reward/Fitness/Priority parameters of the cost model. Similar to the practical model, the artefact uses a scalar value to assert the importance of any particular task.
- The visibility and distance of the task and robot sets respectively emulate the cost parameter of the model.
- The artefact uses these parameters to calculate the utility value of any possible allocation and calculates the probability of the task allocation.

The experiment successfully represents the cost based allocation model with support for having a parameter bias to change whether the algorithm should prioritise quality or visibility.

5.3.2 ST-SR-IA Task Allocation

The experiment structure fits into the ST-SR-TA MRTA taxonomy mentioned in the section-2.3.3.

- The algorithm supports robots which are capable of executing at most one task at a time.
- The algorithm supports tasks which require exactly one robot to accomplish.
- The algorithm supports instantaneous task allocation.

Comparing this developed version of the algorithm to the ideal algorithm structure in section-2.4.4, the experiment achieves 2 of the 3 taxonomy characteristics successfully.

5.3.3 Heterogeneous Robots Support

The system has full support for heterogeneous robots and the experiment takes the types of robots into consideration when trying to allocate tasks. The algorithm verifies if the robot is capable to perform a specific task type.

Comparing this developed version of the algorithm to the ideal algorithm structure in section-2.4.4, the experiment achieves the characteristic successfully and allows the system to be realistic and relate to the research scenario where heterogeneous robot support is needed.

5.3.4 3D simulation of a real-world regional area

The experiment uses 3D coordinates for its objects to simulate a real-world regional area. For this version of the system, only coordinate support is accomplished for the robots and tasks but the system is capable of supporting different zone characteristics which are mentioned in section-3.2.

5.3.5 Different Robot Type support

The system has full support for different types of robots. The robots can have different characteristics such as:

- Specific type of the robot such as Aerial or Ground robot.
- For each specific type, task capabilities are used by the system to ascertain the ability to accomplish a particular task.

The experiment also has support for static robot sets (section-3.2) which can be used to enforce constant comparisons between different types of algorithm in the future.

5.3.6 Different Task Type Support

The system has full support for different types of tasks. The tasks can have different characteristics such as:

- Specific type of tasks such as aerial rescue mission, ground fire diversions etc. This allows the task to only be assigned to robots which are capable of doing the specific type of task.
- Quality/Priority of a task which ranges from 1 to 10. This priority allows the enforcement of bias allocation for tasks which are deemed more critical than others.

These parameters allows the overall system to simulate realistic tasks which are of different types and complexities.

The experiment also has support for static task sets (section-3.2) which can be used to enforce constant comparisons between different types of algorithm in the future.

5.3.7 Metric Data Collection Support

The experiment is able to output and collect different data metrics which are useful in conducting quantitative evaluation of the algorithms.

The system was able to collect 2 out of 5 data metrics as mentioned in the section-3.3. These two metrics gives the ability to differentiate and sort algorithms based on the specific metrics. This is especially useful when identifying algorithms which take minimal local run-time to allocate a task of critical priority.

5.4 Quantitative Evaluation

In this section, the experiment is quantitatively evaluated based on the research scenario and goals. It will aim to answer how well the developed system uses the specific data metrics to answer the research questions.

5.4.1 Local Allocation Assignment Run-time

The experiment collects data about the time taken to allocate each individual task. The algorithm defines the run-time of any allocation as the time subtraction of when it was added in the queue and when it was allocated.

This data metric is one of the five metrics which allows us to answer the Research Question-1 to an certain extent.

The local run-time also helps in situations when two algorithms have similar global run-time values. It acts as a deciding factor between the two when identifying the better solution. As in some cases an algorithm might struggle to allocate more critical priority tasks, which is a negative considering our scenario.

This metric can also be useful in the future development when solving the MINIAVE optimization objective mentioned in the section-2.4.3. Minimizing the task completion is important for this project as in some critical tasks, time taken to complete is more important than any aggregated global costs. Such as in rescue situations where time to victim location may be critical.

5.4.2 Global Allocation Assignment Run-time

From the data collected in the experiment about the total run-time of the algorithm, individual performance run-time of algorithms can be conducted. The algorithm sums the local allocation run-time to produce the combine global run-time of the allocation.

This data metric is one of the five metrics which allows us to answer the Research Question-1 to an certain extent.

Compared to the local run-time, the global run-time is important but is not the most important criteria when judging an algorithm. As in some cases an algorithm might perform better overall but find it difficult to allocate critical priority tasks.

5.4.3 Experiment Analysis

Experiment subsets were run on the algorithm to test for different number of robots and tasks with constant and randomised coordinates. Based on the output from these min-experiments, local and allocation run-time is collected in tables and analysed in the form of graphs.

In the experiments the sets defined in ”Possible Tasks” and ”Robots in the systems” of the section-3.2.2 are used. These sets are as follows:

Task Sets

- Set-1: 5 Tasks, 1 aerial extinguishing task, 3 ground extinguishing tasks, 1 ground pickup.
- Set-2: 10 Tasks, 1 aerial extinguishing task, 1 aerial rescue pickup, 6 ground extinguishing tasks, 2 ground pickups.
- Set-2: 15 Tasks, 2 aerial extinguishing tasks, 1 aerial rescue pickup, 9 ground extinguishing tasks, 3 ground pickups.

Robot Sets

- Set-1: 5 Robots, 1 aerial robot capable of surveying, rescuing and firefighting; 4 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.
- Set-2: 10 Robots, 2 aerial robots capable of surveying, rescuing and firefighting; 8 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.

- Set-3: 15 Robots, 3 aerial robots capable of surveying, rescuing and firefighting; 12 ground robots capable of traversing difficult terrain and participate in rescuing and firefighting missions.

1. Experiment-1

In this experiment the coordinates for the robots and tasks are kept constant and the number of tasks and robots are increased in number according to the sets mentioned. The goal is to evaluate the performance of the algorithm both local and global when the number of tasks increase in the queue. Due to the current nature of the algorithm, the number of tasks cannot exceed the number of robots, so both the parameters needed to be increased simultaneously.

All the robots in the system start from the center point of the 3D Environment which in terms of coordinates is (100,100,0). Similarly the tasks in the system all start from a fixed location which in terms of coordinates is (150,150,0).

Due to the nature of the algorithm, each time it was executed the output varied in small amounts. To solve this issue, the algorithm was run ten times and the associated output values were used to calculate the average.

Executing this experiment produced the following outputs which are displayed in the tables - 1, 2, 3

The Output logs of this experiment can be found in section-9.6, 9.7, 9.8 of the Appendix.

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	0.17	0.28	0.14	0.14	0.14	0.14	0.26	0.14	0.28	0.25	0.19
Task-2	0.17	0.28	0.14	0.14	0.14	0.14	0.26	0.14	0.29	0.26	0.20
Task-3	0.17	0.28	0.14	0.14	0.14	0.14	0.26	0.14	0.29	0.26	0.20
Task-4	0.17	0.28	0.15	0.14	0.14	0.14	0.26	0.14	0.29	0.26	0.20
Task-5	0.18	0.29	0.15	0.15	0.14	0.15	0.27	0.15	0.29	0.26	0.20
Total Time	0.86	1.40	0.73	0.71	0.71	0.72	1.31	0.72	1.44	1.30	0.99

Table 1: Experiment-1: Local and Global run-times when the number of tasks are 5

From the data in the Tables - 1, 2, 3, Table - 4 was created for mean local and global run-time values.

In the Table-4, the ratio increase of global and local time is especially important as it can show the trend of the algorithm when the number of tasks in the system are increased.

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	0.70	0.59	0.66	0.65	0.76	0.62	0.62	0.72	0.62	1.02	0.70
Task-2	0.70	0.60	0.66	0.66	0.76	0.63	0.62	0.72	0.62	1.02	0.70
Task-3	0.70	0.60	0.66	0.66	0.77	0.63	0.62	0.72	0.62	1.02	0.70
Task-4	0.70	0.60	0.66	0.66	0.77	0.63	0.62	0.72	0.62	1.02	0.70
Task-5	0.70	0.60	0.66	0.66	0.77	0.63	0.63	0.72	0.63	1.02	0.70
Task-6	0.71	0.60	0.67	0.66	0.77	0.64	0.63	0.73	0.63	1.02	0.71
Task-7	0.71	0.60	0.67	0.66	0.77	0.64	0.63	0.73	0.63	1.02	0.71
Task-8	0.71	0.60	0.67	0.66	0.78	0.64	0.63	0.73	0.63	1.03	0.71
Task-9	0.71	0.60	0.67	0.66	0.78	0.64	0.63	0.73	0.63	1.03	0.71
Task-10	0.71	0.61	0.67	0.67	0.78	0.64	0.63	0.73	0.64	1.03	0.71
Total-time	7.04	5.99	6.64	6.60	7.71	6.34	6.26	7.24	6.28	10.21	7.03

Table 2: Experiment-1: Local and Global run-times when the number of tasks are 10

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	1.64	1.40	1.46	1.56	1.79	1.58	1.24	1.43	1.44	1.58	1.51
Task-2	1.64	1.41	1.47	1.56	1.79	1.58	1.24	1.44	1.44	1.58	1.52
Task-3	1.64	1.41	1.47	1.56	1.80	1.59	1.24	1.44	1.44	1.58	1.52
Task-4	1.65	1.41	1.47	1.57	1.80	1.59	1.28	1.44	1.45	1.58	1.52
Task-5	1.65	1.41	1.47	1.57	1.80	1.59	1.36	1.44	1.45	1.59	1.53
Task-6	1.65	1.41	1.47	1.57	1.81	1.59	1.38	1.44	1.45	1.59	1.54
Task-7	1.65	1.42	1.48	1.57	1.81	1.59	1.38	1.45	1.45	1.59	1.54
Task-8	1.66	1.42	1.48	1.57	1.81	1.59	1.38	1.45	1.46	1.59	1.54
Task-9	1.66	1.42	1.48	1.57	1.81	1.60	1.38	1.45	1.46	1.60	1.54
Task-10	1.66	1.42	1.48	1.58	1.81	1.60	1.38	1.45	1.46	1.60	1.54
Task-11	1.66	1.42	1.49	1.58	1.82	1.60	1.39	1.45	1.46	1.60	1.55
Task-12	1.66	1.44	1.48	1.58	1.82	1.60	1.39	1.45	1.46	1.60	1.55
Task-13	1.67	1.47	1.49	1.58	1.82	1.60	1.39	1.45	1.46	1.60	1.55
Task-14	1.67	1.54	1.49	1.58	1.82	1.60	1.40	1.46	1.47	1.60	1.56
Task-15	1.68	1.56	1.49	1.59	1.82	1.60	1.40	1.46	1.47	1.60	1.57
Total-Time	24.84	21.54	22.17	23.60	27.12	23.90	20.23	21.69	21.82	23.88	23.08

Table 3: Experiment-1: Local and Global run-times when the number of tasks are 15

Number of Tasks	5	10	15
Number of Robots	5	10	15
Mean Global Runtime	0.99	7.03	23.08
Mean Local Runtime	0.2	0.7	1.54
Ratio of Global Time Increase	N/A	7	3
Ratio of Local Time Increase	N/A	3	2

Table 4: Experiment-1: Mean Global and Local Run-times

From this small data set we can see when number of tasks are doubled, the ratio of global time increase is 7:1. When the number of tasks are increased 1.5 times, the ratio of global time increase is 3:1. Based on these two ratios, if we were to increase the tasks to 20, it is possible that the ratio would be around 7,6:1.

Thus we can assume that when the number of tasks are relatively small, the global performance of the algorithm shows a linear trend. The figure-9 created from this table also proves this assumption.

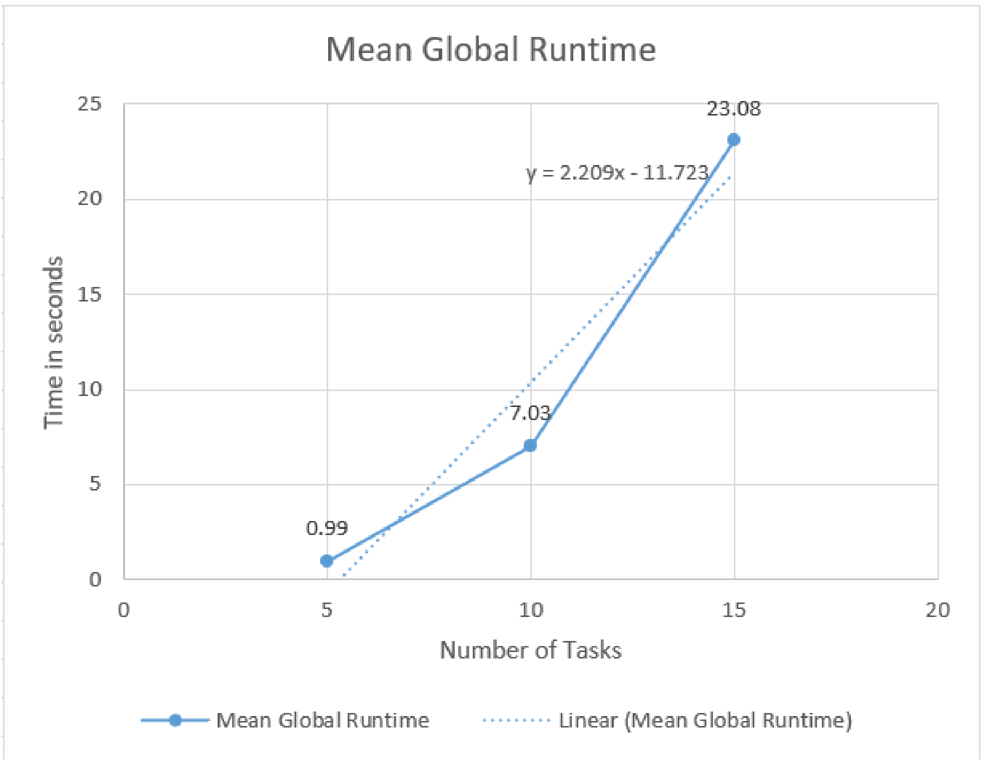

Figure 9: Experiment-1 Mean Global Run-time

Similarly for the local run-times, the same linear trend is observed for small increment in number of tasks. The figure-10 shows the plotted data and the trend.

2. Experiment-2

In this experiment the coordinates for the robots and tasks are determined randomly t and the number of tasks and robots are increased in number according to the sets mentioned. The goal is to evaluate the performance of the algorithm both local and global when the number of tasks increase in the queue and to test the algorithm performance when the coordinates are within the bounds of the 3D simulated space. Due to the current nature of the algorithm, the number of tasks cannot exceed the number of robots, so both the parameters needed to be increased simultaneously.

The robots and the tasks in the system can have coordinates which range from (0,0,0) to (200,200,200).

Due to the nature of the algorithm, each time it was executed the output varied in

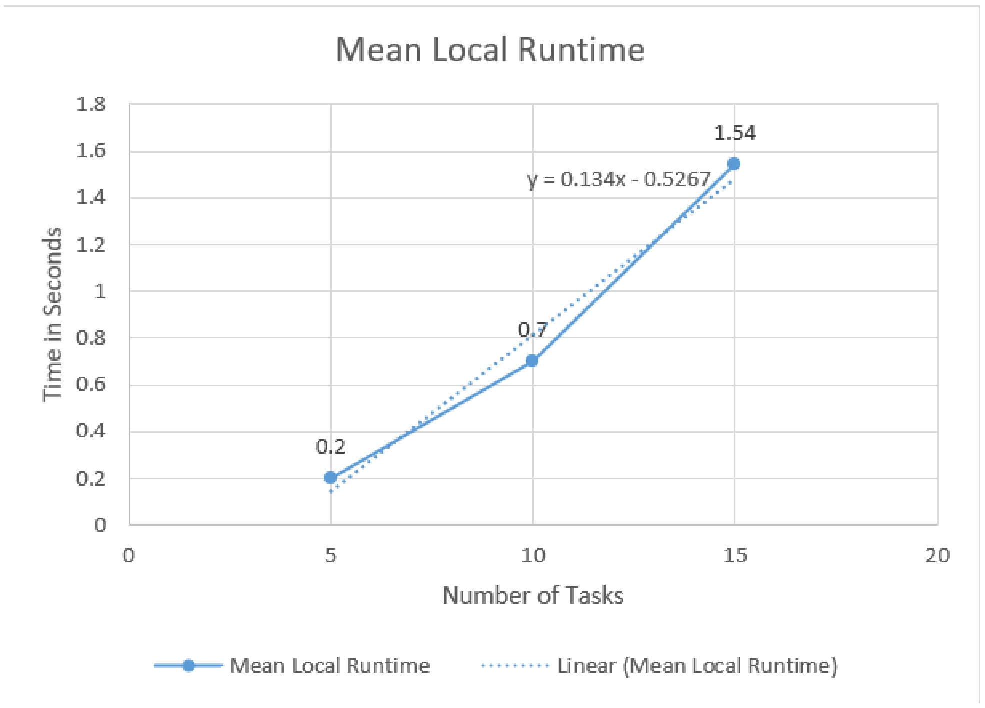

Figure 10: Experiment-1 Mean Local Run-time

small amounts. To solve this issue, the algorithm was run ten times and the associated output values were used to calculate the average.

Executing this experiment produced the following outputs which are displayed in the tables - 5, 6, 7

The Output logs of this experiment can be found in section-9.9, 9.10, 9.11 of the Appendix.

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	0.16	0.15	0.15	0.15	0.27	0.14	0.15	0.23	0.27	0.29	0.20
Task-2	0.17	0.15	0.16	0.15	0.27	0.14	0.16	0.23	0.27	0.29	0.20
Task-3	0.17	0.15	0.16	0.16	0.28	0.14	0.16	0.23	0.27	0.29	0.20
Task-4	0.17	0.16	0.16	0.16	0.28	0.14	0.16	0.24	0.27	0.29	0.20
Task-5	0.17	0.16	0.16	0.16	0.28	0.15	0.16	0.24	0.27	0.29	0.20
Total-Time	0.84	0.78	0.79	0.78	1.38	0.72	0.78	1.17	1.35	1.45	1.00

Table 5: Experiment-2: Local and Global run-times when the number of tasks are 5

From the data in the Tables - 5, 6, 7, Table - 8 was created for mean local and global run-time values.

Similarly in this experiment the Table-8 also shows the ratio increase of global and local time. This ration again demonstrates the trend discussed before but more

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	0.75	0.62	0.69	0.68	0.63	0.49	0.82	0.80	0.63	0.74	0.69
Task-2	0.75	0.62	0.70	0.69	0.63	0.50	0.82	0.80	0.63	0.74	0.69
Task-3	0.76	0.63	0.70	0.69	0.63	0.50	0.82	0.80	0.63	0.75	0.69
Task-4	0.76	0.63	0.70	0.69	0.63	0.50	0.83	0.80	0.63	0.75	0.69
Task-5	0.76	0.63	0.70	0.69	0.63	0.50	0.83	0.81	0.63	0.75	0.69
Task-6	0.77	0.63	0.70	0.69	0.63	0.50	0.83	0.81	0.64	0.75	0.70
Task-7	0.77	0.63	0.70	0.70	0.64	0.51	0.84	0.81	0.64	0.75	0.70
Task-8	0.77	0.64	0.71	0.70	0.64	0.51	0.84	0.81	0.64	0.75	0.70
Task-9	0.77	0.64	0.71	0.70	0.64	0.51	0.84	0.81	0.64	0.76	0.70
Task-10	0.77	0.64	0.71	0.70	0.64	0.51	0.84	0.81	0.65	0.76	0.70
Total-Time	7.64	6.30	7.01	6.93	6.34	5.02	8.31	8.06	6.36	7.50	6.95

Table 6: Experiment-2: Local and Global run-times when the number of tasks are 10

Run	1	2	3	4	5	6	7	8	9	10	Average
Task-1	1.11	1.22	1.32	1.30	1.48	1.18	2.19	1.38	1.12	1.16	1.35
Task-2	1.11	1.22	1.32	1.30	1.48	1.18	2.19	1.38	1.12	1.16	1.35
Task-3	1.11	1.23	1.32	1.31	1.48	1.19	2.19	1.38	1.12	1.16	1.35
Task-4	1.12	1.23	1.33	1.31	1.49	1.19	2.20	1.39	1.12	1.16	1.35
Task-5	1.12	1.23	1.33	1.31	1.49	1.19	2.20	1.39	1.13	1.16	1.36
Task-6	1.12	1.23	1.33	1.31	1.49	1.19	2.20	1.39	1.13	1.17	1.36
Task-7	1.12	1.23	1.34	1.31	1.49	1.20	2.20	1.39	1.13	1.17	1.36
Task-8	1.13	1.23	1.34	1.32	1.50	1.20	2.21	1.40	1.13	1.17	1.36
Task-9	1.13	1.24	1.34	1.32	1.50	1.20	2.22	1.40	1.14	1.17	1.37
Task-10	1.13	1.24	1.34	1.32	1.50	1.20	2.26	1.40	1.14	1.17	1.37
Task-11	1.13	1.24	1.35	1.32	1.50	1.21	2.27	1.40	1.14	1.18	1.37
Task-12	1.13	1.24	1.35	1.32	1.50	1.21	2.27	1.40	1.14	1.18	1.37
Task-13	1.13	1.25	1.35	1.32	1.50	1.21	2.27	1.40	1.14	1.18	1.38
Task-14	1.13	1.25	1.35	1.33	1.51	1.21	2.28	1.41	1.14	1.18	1.38
Task-15	1.14	1.25	1.36	1.33	1.51	1.22	2.28	1.41	1.15	1.18	1.38
Total-Time	16.86	18.52	20.08	19.73	22.42	17.98	33.43	20.91	16.99	17.55	20.45

Table 7: Experiment-2: Local and Global run-times when the number of tasks are 15

importantly it shows that the algorithm performs similarly when the coordinates are randomised. As the coordinates in the ideal real-world algorithms are always going to be dynamic, it is important for any tested algorithm not to behave erratically if any location changes are made.

Compared to the previous data set, we can see here when number of tasks are doubled, the ratio of global time increase is 6:1 and When the number of tasks are increased 1.5 times, the ratio of global time increase is 3:1. Comparing these ratios, it reaffirms that the algorithm shows a linear trend when the tasks the number of tasks are fairly low.

It is important to note that due to the small data set, we cannot make a presumption

Number of Tasks	5	10	15
Number of Robots	5	10	15
Mean Global Runtime	1	6.95	20.45
Mean Local Runtime	0.2	0.69	1.36
Ratio of Global Time Increase	N/A	6	2
Ratio of Local Time Increase	N/A	3	1

Table 8: Experiment-2: Mean Global and Local Run-times

that the trend will remain linear in nature. It is likely that with the a higher increase in tasks, the trend will change.

The figure-11 shows the Mean global run-time trend and the figure–12 shows the Mean local run-time.

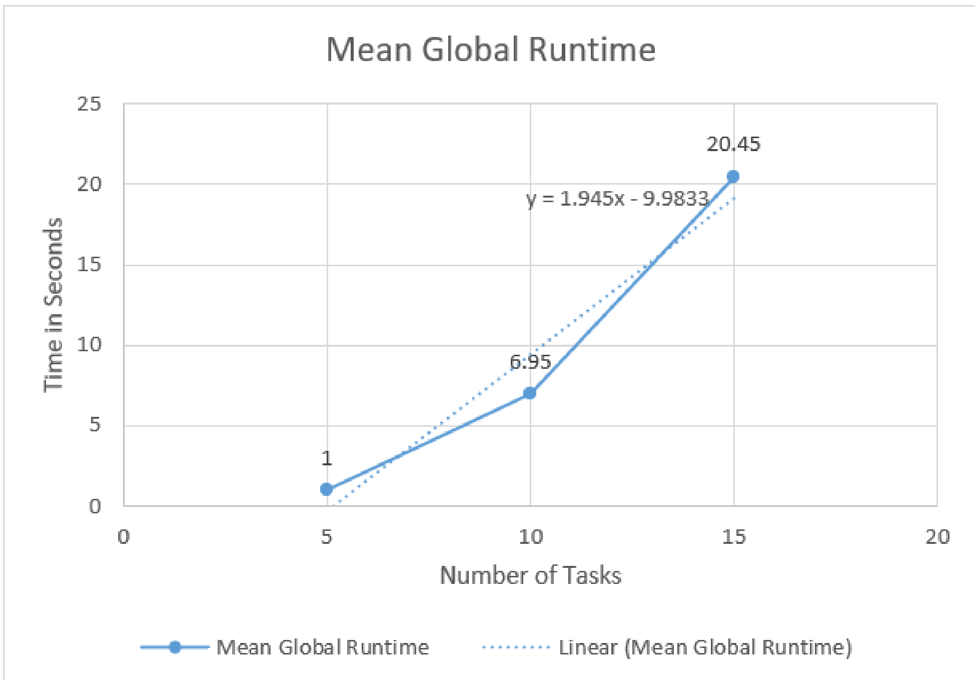

Figure 11: Experiment-2: Mean Global Run-time

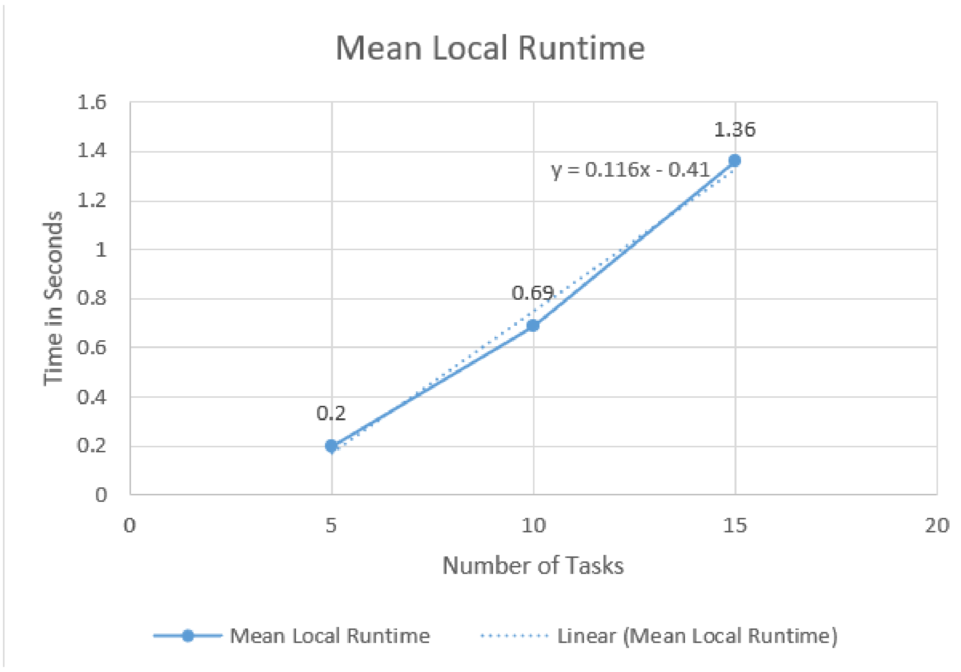

Figure 12: Experiment-2: Mean Local Run-time

6 Experiment Results & Discussion

Based on the experiment evaluation conducted in the section-5, the minimal viable artefact matched many of the set criteria needed for the system in which algorithms can be compared and thus help in deciding the most suitable algorithm for the research scenario and answer the research questions.

The experiment full-filled the qualitative characteristics that were mentioned in the design and methodology. In terms of the research questions, the experiment partially helped in setting the structure to differentiate between algorithms in terms of the defined evaluation constraints. Two of the Five constraints were successfully demonstrated quantitatively.

In the future development sprints, this artefact can be expanded and improved upon to enable algorithm comparison using all the defined metrics and ultimately answer research questions 1 and 3. The improved version of this artefact will allow decentralized task allocation which in term will show the advantages and disadvantages of centralization and decentralization at least in terms of scalability, task complexity and performance.

Thus partially addressing the research question 2.

7 Threats to Validity

To this best of my knowledge, there exist no threats to the validation of this research at this stage.

8 Conclusion & Future Work

8.1 Conclusion

At the start of the project, the main goals and key objectives of the research were presented. Using these goals and objectives, a literature review was conducted on the background and specific knowledge about the topics MAS, MRS and MRTA. In the literature review the definition for the ideal algorithm was given along with three suitable algorithms that matched the ideal algorithm. The literature review was concluded with a gap analysis of current research, using which the research questions were defined with their importance and relevance to the project.

A research design and methodology was created to enable the answering of the research questions. The experiment design was presented along with the data collection rules and how the data can be used to evaluate the experiment quantitatively. Possible errors and risks were also mentioned and the strategy on how to mitigate them.

Using the research design, a minimal viable artefact was developed. The development tools were touched on and the Centralised version of the Bees Algorithm was thoroughly discussed alongside the code logic and functions. How the algorithm allocated the task was broken down with the relevant mathematical functions and models.

The created artefact was then evaluated quantitatively and qualitatively. The evaluation was divided in the respective subsections with the qualitative evaluation aiming to answer whether the experiment accomplished its goals and the quantitative evaluation aiming to answer how well the experiment accomplishes the goals.

Two sub-experiments were presented and discussed in context of the research questions. Experiment-1 showed that when the amount of tasks were increased the algorithm displayed a linear global and local allocation time trend. Similarly Experiment showed that when the location of robots and tasks in the system was randomised the algorithm achieved the allocation in a similar liner fashion. It was noted that due to the small subset, the true task increase vs performance trend was not discussed rather its initial tangent when the number of tasks in the system were fairly low.

Finally the results of the experiment design, development and evaluation are discussed in relation to the research questions and how the artefact was able to answer the questions partially.

8.2 Future Work

With the artefact acting as the starting point, the aim is to develop a full scale visual simulation model alongside which can be used to answer the main questions thoroughly. The current centralised version of the algorithm will also be converted to a decentralised version along with the implementation of the S+T and Distributed Hungarian algorithms.

9 Appendix

9.1 Robot.py

```python
class Robot:
    '''Base class to represent the robot object type in the experiment.'''

    def __init__(self, robotID, robotType, robotLocation, isCapable):
        self.robotID = robotID
        self.robotType = robotType
        self.robotLocation = robotLocation
        self.taskAssigned = -1
        self.isBusy = False
        self.isCapable = isCapable

    def assign_task(self,taskID):
        self.taskAssigned = taskID
        self.isBusy = True

    def get_robot_id(self):
        return self.robotID

    def get_robot_type(self):
        return self.robotType

    def get_robot_location(self):
        return self.robotLocation

    def get_is_capable(self):
        return self.isCapable

    def get_robot_location(self):
        return self.robotLocation

    def get_robot_status(self):
        return self.isBusy

    def set_robot_location(self,newLocation):
        self.robotLocation = newLocation

    def set_robot_task(self,newTask):
        self.taskAssigned = newTask

    def set_robot_status(self,newStatus):
        self.isBusy = newStatus

    pass
```

9.2 Task.py

```python
from datetime import datetime
```

```
import time

class Task:
    '''Base class to represent the task object type in the experiment.'''

    def __init__(self, taskID, taskQuality, taskLocation, taskType):
        self.taskID = taskID
        self.taskQuality = taskQuality
        self.taskLocation = taskLocation
        self.taskType = taskType

        current_time = time.time()
        #now = datetime.now()
        #current_time = now
        #current_time = now.strftime("%H:%M:%S")
        self.timeAdded = current_time
        self.robotAllocated = 0
        self.taskAllocated = False

    def allocate_task(self,robotID):
        self.robotAllocated = robotID
        self.taskAllocated = True
        current_time = time.time()
        #now = datetime.now()
        #current_time = now.strftime("%H:%M:%S")
        self.timeAllocated = current_time

    def get_task_id(self):
        return self.taskID

    def get_task_quality(self):
        return self.taskQuality

    def get_task_type(self):
        return self.taskType

    def get_task_location(self):
        return self.taskLocation

    def get_task_quality(self):
        return self.taskQuality

    def get_robot_allocated(self):
        return self.robotAllocated

```

```python
    def get_time_added(self):
        return self.timeAdded

    def get_time_taken_to_allocate(self):
        return (self.timeAllocated - self.timeAdded )

    def set_task_relative_quality(self, relative_quality):
        self.taskRelativeQuality = relative_quality

    def get_task_relative_quality(self):
        return self.taskRelativeQuality

    def set_task_status(self, status):
        self.taskAllocated = status

    def set_robot_allocated(self, allocated):
        self.robotAllocated = allocated

    pass
```

9.3 Coordinate.py

```python
class Coordinate:
    '''Base class to represent the coordinates of the robots and the tasks.
   '''

    def __init__(self, X, Y, Z):
        self.X = X
        self.Y = Y
        self.Z = Z

    def get_coordinates(self):
        return self

    def get_x_coordinate(self):
        return self.X

    def get_y_coordinate(self):
        return self.Y

    def get_z_coordinate(self):
        return self.Z

    pass
```

9.4 CBA.py

```
from Robot import Robot
from Task import Task
from Coordinate import Coordinate
import math
import random

robot_list = []
task_list = []

# Function to create different sets of robots
def create_robots(set):
    print("")
    print("*****")
    print("Starting create_robots function")
    print("*****")
    print("")

    SET_ONE_ID = [1,2,3,4,5]
    SET_TWO_ID = [1,2,3,4,5,6,7,8,9,10]
    SET_THREE_ID = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

    TYPE_GROUND = "ground_robot"
    TYPE_AERIAL = "aerial_robot"

    TASK_LIST_GROUND = ["ground_fire_extinguish", "ground_rescue"]
    TASK_LIST_AERIAL = ["aerial_fire_extinguish", "aerial_rescue"]

    BASE_LOCATION_X = 100
    BASE_LOCATION_Y = 100
    BASE_LOCATION_Z = 0

    BASE_COORDINATES = Coordinate(BASE_LOCATION_X, BASE_LOCATION_Y,
    BASE_LOCATION_Z)

    global robot_list

    # set-1, 1 aerial robot, 4 ground robots
    if(set == 1):
        robot_list.append(Robot(SET_ONE_ID[0], TYPE_AERIAL,
    BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_ONE_ID[1], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_ONE_ID[2], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_ONE_ID[3], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_ONE_ID[4], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()
    ))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()
    ))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()
    ))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

    if(set == 2):
        robot_list.append(Robot(SET_TWO_ID[0], TYPE_AERIAL,
    BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_TWO_ID[1], TYPE_AERIAL,
    BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_TWO_ID[2], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[3], TYPE_GROUND,
    BASE_COORDINATES, TASK_LIST_GROUND))
```

```python
        robot_list.append(Robot(SET_TWO_ID[4], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[5], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[6], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[7], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[8], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_TWO_ID[9], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

    if(set == 3):
        robot_list.append(Robot(SET_THREE_ID[0], TYPE_AERIAL, BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_THREE_ID[1], TYPE_AERIAL, BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_THREE_ID[2], TYPE_AERIAL, BASE_COORDINATES, TASK_LIST_AERIAL))
        robot_list.append(Robot(SET_THREE_ID[3], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[4], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[5], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[6], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[7], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[8], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[9], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[10], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[11], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[12], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[13], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        robot_list.append(Robot(SET_THREE_ID[14], TYPE_GROUND, BASE_COORDINATES, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

```

```python
    if(set == 4):
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_ONE_ID[0], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_ONE_ID[1], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_ONE_ID[2], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_ONE_ID[3], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_ONE_ID[4], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

    if(set == 5):
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[0], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[1], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[2], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[3], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[4], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[5], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[6], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[7], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[8], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_TWO_ID[9], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
```

```python
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

    if(set == 6):
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[0], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[1], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[2], TYPE_AERIAL, random_coordinates, TASK_LIST_AERIAL))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[3], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[4], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[5], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[6], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[7], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[8], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[9], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[10], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[11], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[12], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[13], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        robot_list.append(Robot(SET_THREE_ID[14], TYPE_GROUND, random_coordinates, TASK_LIST_GROUND))
        print("Created Set-" + str(set) + " of Robot Sets.")
        print("")
        for robot in robot_list:
            print("*****")
            print("Robot ID: " + str(robot.get_robot_id()))
            print("Robot Type: " + robot.get_robot_type())
            print("Robot Location: ")
            print("X: " + str(robot.get_robot_location().get_x_coordinate()))
            print("Y: " + str(robot.get_robot_location().get_y_coordinate()))
            print("Z: " + str(robot.get_robot_location().get_z_coordinate()
    ))
            print("Robot Task Capabilities: ")
            for capability in robot.get_is_capable():
                print(capability)
            print("*****")
            print("")
        print("*****")
        print("Ending create_robots function")
        print("*****")
        print("")

# Function to create different sets of tasks
def create_tasks(set):
    print("")
    print("*****")
    print("Starting create_tasks function")
    print("*****")
    print("")

    SET_ONE_ID = [1,2,3,4,5]
    SET_TWO_ID = [1,2,3,4,5,6,7,8,9,10]
    SET_THREE_ID = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

    TASK_AERIAL_FIREFIGHT = "aerial_fire_extinguish"
    TASK_GROUND_FIREFIGHT = "ground_fire_extinguish"
    TASK_AERIAL_RESCUE = "aerial_rescue"
    TASK_GROUND_RESCUE = "ground_rescue"

    QUALITY_AERIAL_FIREFIGHT = 6
    QUALITY_GROUND_FIREFIGHT = 4
    QUALITY_AERIAL_RESCUE = 10
    QUALITY_GROUND_RESCUE = 8

    SAMPLE_COORDINATE_X = 150
    SAMPLE_COORDINATE_Y = 150
    SAMPLE_COORDINATE_Z = 0
    SAMPLE_COORDINATES = Coordinate(SAMPLE_COORDINATE_X,
    SAMPLE_COORDINATE_Y, SAMPLE_COORDINATE_Z)

    global task_list

    # set-1, 1 aerial firefight, 3 ground firefight, 1 ground rescue
    if(set == 1):
```

```python
        task_list.append(Task(SET_ONE_ID[0], QUALITY_AERIAL_FIREFIGHT, SAMPLE_COORDINATES, TASK_AERIAL_FIREFIGHT))
        task_list.append(Task(SET_ONE_ID[1], QUALITY_GROUND_RESCUE, SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_ONE_ID[2], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_ONE_ID[3], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_ONE_ID[4], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))

        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

    if(set == 2):
        task_list.append(Task(SET_TWO_ID[0], QUALITY_AERIAL_RESCUE, SAMPLE_COORDINATES, TASK_AERIAL_RESCUE))
        task_list.append(Task(SET_TWO_ID[1], QUALITY_AERIAL_FIREFIGHT, SAMPLE_COORDINATES, TASK_AERIAL_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[2], QUALITY_GROUND_RESCUE, SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_TWO_ID[3], QUALITY_GROUND_RESCUE, SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_TWO_ID[4], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[5], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[6], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[7], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[8], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_TWO_ID[9], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

    if(set == 3):
        task_list.append(Task(SET_THREE_ID[0], QUALITY_AERIAL_RESCUE,
SAMPLE_COORDINATES, TASK_AERIAL_RESCUE))
        task_list.append(Task(SET_THREE_ID[1], QUALITY_AERIAL_FIREFIGHT,
SAMPLE_COORDINATES, TASK_AERIAL_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[2], QUALITY_AERIAL_FIREFIGHT,
SAMPLE_COORDINATES, TASK_AERIAL_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[3], QUALITY_GROUND_RESCUE,
SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_THREE_ID[4], QUALITY_GROUND_RESCUE,
SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_THREE_ID[5], QUALITY_GROUND_RESCUE,
SAMPLE_COORDINATES, TASK_GROUND_RESCUE))
        task_list.append(Task(SET_THREE_ID[6], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[7], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[8], QUALITY_GROUND_FIREFIGHT,
SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
```

```python
        task_list.append(Task(SET_THREE_ID[9], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[10], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[11], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[12], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[13], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        task_list.append(Task(SET_THREE_ID[14], QUALITY_GROUND_FIREFIGHT, SAMPLE_COORDINATES, TASK_GROUND_FIREFIGHT))
        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

    if(set == 4):
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_ONE_ID[0], QUALITY_AERIAL_FIREFIGHT, random_coordinates, TASK_AERIAL_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_ONE_ID[1], QUALITY_GROUND_RESCUE, random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_ONE_ID[2], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_ONE_ID[3], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_ONE_ID[4], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))

        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

    if(set == 5):
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[0], QUALITY_AERIAL_RESCUE, random_coordinates, TASK_AERIAL_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[1], QUALITY_AERIAL_FIREFIGHT, random_coordinates, TASK_AERIAL_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[2], QUALITY_GROUND_RESCUE, random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[3], QUALITY_GROUND_RESCUE, random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[4], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[5], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[6], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[7], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[8], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_TWO_ID[9], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

    if(set == 6):
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[0], QUALITY_AERIAL_RESCUE,
    random_coordinates, TASK_AERIAL_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[1], QUALITY_AERIAL_FIREFIGHT,
    random_coordinates, TASK_AERIAL_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[2], QUALITY_AERIAL_FIREFIGHT,
    random_coordinates, TASK_AERIAL_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[3], QUALITY_GROUND_RESCUE,
    random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[4], QUALITY_GROUND_RESCUE,
    random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[5], QUALITY_GROUND_RESCUE,
    random_coordinates, TASK_GROUND_RESCUE))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[6], QUALITY_GROUND_FIREFIGHT,
    random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[7], QUALITY_GROUND_FIREFIGHT,
    random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[8], QUALITY_GROUND_FIREFIGHT,
    random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[9], QUALITY_GROUND_FIREFIGHT,
    random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random
    .random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[10], QUALITY_GROUND_FIREFIGHT,
    random_coordinates, TASK_GROUND_FIREFIGHT))
```

```python
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[11], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[12], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[13], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        random_coordinates = Coordinate(200 * random.random(), 200 * random.random(), 200 * random.random())
        task_list.append(Task(SET_THREE_ID[14], QUALITY_GROUND_FIREFIGHT, random_coordinates, TASK_GROUND_FIREFIGHT))
        print("Created Set-" + str(set) + " of Task Sets.")
        print("")
        for task in task_list:
            print("*****")
            print("Task ID: " + str(task.get_task_id()))
            print("Task Type: " + task.get_task_type())
            print("Task Quality: "+ str(task.get_task_quality()))
            print("Task Location: ")
            print("X: " + str(task.get_task_location().get_x_coordinate()))
            print("Y: " + str(task.get_task_location().get_y_coordinate()))
            print("z: " + str(task.get_task_location().get_z_coordinate()))
            #print(task.get_time_added())
            print("*****")
            print("")
        print("*****")
        print("Ending create_tasks function")
        print("*****")
        print("")

# function to calculate the distances of all the robots to all the tasks
def calculate_distance(robot_coordinates, task_coordinates):
    x_axis_distance = robot_coordinates.get_x_coordinate() - task_coordinates.get_x_coordinate()
    x_axis_distance = pow(x_axis_distance, 2)
    y_axis_distance = robot_coordinates.get_y_coordinate() - task_coordinates.get_y_coordinate()
    y_axis_distance = pow(y_axis_distance, 2)
    z_axis_distance = robot_coordinates.get_z_coordinate() - task_coordinates.get_z_coordinate()
    z_axis_distance = pow(z_axis_distance, 2)
    distance = math.sqrt(x_axis_distance + y_axis_distance + z_axis_distance)
    return distance

# function to calculate the visibility of all the robots to all the tasks
def calculate_visibility(distance):
    visibility = 1/distance
    return visibility

# function to check the capability of the robots and find the visibility sets
def check_capability_and_calculate_visibility():
    task_robot_visibility_set = [[]]

    print("*****")
    print("Checking capability and calculating visibility sets")
    print("*****")
    print("")

    # Calculating capability, distance sets, visibility sets
    for task in task_list:
        task_type = task.get_task_type()
        print("*****")
        print("Task-" + str(task.get_task_id()) + " type is: " + task_type)
        print("*****")
        print("")
        robot_distance_from_task = []
        robot_visibility_from_task = []

        robot_capable = False # enforce false value for correct checking of capability
        #check to see if the tasks in the queue can be accomplished by the current robots available
        for robot in robot_list:
            capable_task_list = robot.get_is_capable()
            print("-----")
            print("Robot-" + str(robot.get_robot_id()) + " is capable of tasks: ")

            for capable_task in capable_task_list:
                print(capable_task)
                if(capable_task == task_type):
                    robot_capable = True

            # calculate the distances of all the robots to the tasks in the task queue
            if(robot_capable == True):
                print("Robot is Capable of doing this task.")
                distance = calculate_distance(robot.get_robot_location(), task.get_task_location())
                print("Distance of Robot-" + str(robot.get_robot_id()) + " to the task-" + str(task.get_task_id()) + " is " + str(distance))
                robot_distance_from_task.append(distance)
                visibility = calculate_visibility(distance)
                print("Visibility of Robot-" + str(robot.get_robot_id()) + " to the task-" + str(task.get_task_id()) + " is " + str(visibility))
                robot_visibility_from_task.append(visibility)
                robot_capable = False # reset the bool for the loop

            # Robot not capable, invalid distances and visibility
            else:
                print("Robot is not capable of doing this task")
                distance = -1
                visibility = -1
                robot_distance_from_task.append(distance)
                robot_visibility_from_task.append(visibility)
            print("-----")
            print("")


        # Adding sets of robot visibility for this task to the main 2d array
        task_robot_visibility_set.append([robot_visibility_from_task])

    print("")
    print("*****")
    print("Visibility sets calculated")
    print("*****")
    print("")

    return task_robot_visibility_set

# function to calculate the relative quality of the tasks
def calculate_relative_quality():
    quality_list = []
    total_quality = 0

    # Get all qualities and add them in the list
    for task in task_list:
        quality = task.get_task_quality()
        quality_list.append(quality)

    # Calculate total quality
    for quality in quality_list:
        total_quality = quality + total_quality

    # Calculate relative quality
    for task in task_list:
        quality = task.get_task_quality()
        relative_quality = quality/total_quality
        task.set_task_relative_quality(relative_quality)

# function to print the relative quality of all the tasks
def print_relative_quality(task_list):
    print("*****")
    for task in task_list:
        print("Task-" +str(task.get_task_id()) + " Relative Quality is: " + str(task.get_task_relative_quality()))
    print("*****")
    print("")

# function to calculate the utility values from the visibility sets and the quality of the tasks
def calculate_utility(task_robot_visibility_set):
    print("*****")
    print("Calculating utility")
    print("*****")
    print("")
    task_robot_utility_set = [[]]

    task_index = 0
    for visibility_set in task_robot_visibility_set:
        #print(visibility_set)
        print("*****")

        for visibility in visibility_set:
            print("Calculating utility set for Task-" + str(task_list[task_index].get_task_id()))
            #print(visibility)
            task_utility_set = []
            robot_index = 0
            print("")
            for visible in visibility:
                if(visible != -1):
```

```python
                    print("Robot-" + str(robot_list[robot_index].get_robot_id()) + " has as visibility value with this task")
                    relative_quality = float(task_list[task_index].get_task_relative_quality())
                    utility = visible * relative_quality
                    print("Utility of Task-" + str(task_list[task_index].get_task_id()) + " to the Robot-" + str(robot_list[robot_index].get_robot_id()) + " is = " + str(utility))
                    task_utility_set.append(utility)
                    print("")
                else:
                    utility = -1
                    task_utility_set.append(utility)
                robot_index = robot_index + 1
            task_index = task_index + 1

            task_robot_utility_set.append([task_utility_set])
        print("*****")
        print("")

    return task_robot_utility_set

# function to calculate the utility probability sets
def calculate_utility_probabilities(task_robot_utility_set):
    print("")
    print("*****")
    print("Calculating utility probabilities")
    print("*****")
    print("")
    utility_probabilities = []
    task_index = 0
    for task_robot_utility in task_robot_utility_set:
        print("*****")
        #print(task_robot_utility)
        for utility_set in task_robot_utility:
            print("Calculating utility probabilities for Task-" + str(task_list[task_index].get_task_id()))
            #print(utility_set)
            robot_index = 0
            utility_sum = 0
            task_assignment_probabilities = []
            #print(task_assignment_probabilities)

            for utility in utility_set:
                if(utility != -1):
                    utility_sum += utility
                #print(utility)
            #print(utility_sum)

            for utility in utility_set:
                if(utility != -1):
                    assignment_probability = utility/utility_sum

                    #enforce final assignment
                    task_assignment_probabilities.append(assignment_probability)

                else:
                    assignment_probability = -1
                    task_assignment_probabilities.append(assignment_probability)
                robot_index += 1

            #print(task_assignment_probabilities)
            utility_probabilities.append(task_assignment_probabilities)
            #print(utility_probabilities)
            task_index = task_index + 1

    return utility_probabilities

# function to print hte utility probability sets
def print_utility_probabilities(utility_probabilities):
    print("")
    print("*****")
    print("Utility Probabilities are:")
    print("*****")
    print("")
    task_index = 0
    for utility_probability in utility_probabilities:
        print("*****")
        #print(utility_probability)
        robot_index = 0
        print("Probability of Task-" + str(task_list[task_index].get_task_id()) + " being assigned to the robots are")
        for probability in utility_probability:
            #print(probability)
            print("Robot-" + str(robot_list[robot_index].get_robot_id()) + " = " + str(probability))
            robot_index += 1
        task_index += 1
```

```

# function to decide allocations based on the utility probability sets
def assign_allocations(utility_probabilities):
    print("")
    print("*****")
    print("Allocations decided are:")
    print("*****")
    print("")
    task_index = 0
    for utility_probability in utility_probabilities:
        print("*****")
        print("Assignment probabilities for task-" + str(task_list[
    task_index].get_task_id()) + " are: " + str(utility_probability))
        robot_index = 0

        highest_probability = -1
        assigned_robot_index = -1

        for probability in utility_probability:
            #print(probability)
            if(probability > highest_probability):
                # if the robot is not busy
                if(robot_list[robot_index].get_robot_status() == False):
                    highest_probability = probability
                    assigned_robot_index = robot_index
            robot_index += 1

        if(task_list[task_index].taskAllocated  & robot_list[
    assigned_robot_index].isBusy == False):
            # Allocating the task to this robot and changing its  status to
     true
            task_list[task_index].allocate_task(robot_list[
    assigned_robot_index].get_robot_id())
            robot_list[assigned_robot_index].assign_task(task_list[
    task_index].get_task_id())

            print("Task-" + str(task_list[task_index].get_task_id()) + " is
     assigned to robot-" + str(task_list[task_index].get_robot_allocated())
    + " with a probability of " + str(highest_probability))
        print("*****")
        print("")
        task_index += 1

# function to print the global and local runtime of the algorithm
def print_time_taken_to_allocate():
    print("")
    print("*****")
    print("Time taken to allocate tasks:")
    print("*****")
    print("")

    print("Time taken to allocate for particular tasks:")
    print("")
    total_time = 0
    for task in task_list:
        print("Time taken to allocate task-" + str(task.get_task_id())  + "
 was " + str(round(task.get_time_taken_to_allocate(), 2)) + " seconds")
        total_time += task.get_time_taken_to_allocate()

    print("")
    print("Total time taken to allocate all the tasks was " + str(round(
   total_time, 2)) + " seconds")
    print("")

def main():

    # create robot sets
    create_robots(1)
    # create task sets
    create_tasks(1)

    # Calculate visibility sets
    task_robot_visibility_set = check_capability_and_calculate_visibility()

    # Calculate relative quality
    calculate_relative_quality()

    # print relative quality
    print_relative_quality(task_list)

    # Calculate utilities
    task_robot_utility_set = calculate_utility(task_robot_visibility_set)

    #Calculate utility probabilities
    utility_probabilities = calculate_utility_probabilities(
   task_robot_utility_set)

    # print the utility probability sets
    print_utility_probabilities(utility_probabilities)

    # decide allocations and print them
    assign_allocations(utility_probabilities)

    # print the local and global time taken to allocate tasks
    print_time_taken_to_allocate()

if __name__ == '__main__':
    main()
```

9.5 Output.txt

```
$ python CBA.py

*****
Starting create_robots function
*****

Created Set-1 of Robot Sets.

*****
Robot ID: 1
Robot Type: aerial_robot
Robot Location:
X: 100
Y: 100
Z: 0
Robot Task Capabilities:
aerial_fire_extinguish
aerial_rescue
*****

*****
Robot ID: 2
Robot Type: ground_robot
Robot Location:
X: 100
Y: 100
Z: 0
Robot Task Capabilities:
ground_fire_extinguish
ground_rescue
```

```
*****

*****
Robot ID: 3
Robot Type: ground_robot
Robot Location:
X: 100
Y: 100
Z: 0
Robot Task Capabilities:
ground_fire_extinguish
ground_rescue
*****

*****
Robot ID: 4
Robot Type: ground_robot
Robot Location:
X: 100
Y: 100
Z: 0
Robot Task Capabilities:
ground_fire_extinguish
ground_rescue
*****

*****
Robot ID: 5
Robot Type: ground_robot
Robot Location:
X: 100
Y: 100
Z: 0
Robot Task Capabilities:
ground_fire_extinguish
ground_rescue
*****

*****
Ending create_robots function
*****


*****
Starting create_tasks function
```

```
*****

Created Set-1 of Task Sets.

*****
Task ID: 1
Task Type: aerial_fire_extinguish
Task Quality: 6
Task Location:
X: 150
Y: 150
z: 0
*****

*****
Task ID: 2
Task Type: ground_rescue
Task Quality: 8
Task Location:
X: 150
Y: 150
z: 0
*****

*****
Task ID: 3
Task Type: ground_fire_extinguish
Task Quality: 4
Task Location:
X: 150
Y: 150
z: 0
*****

*****
Task ID: 4
Task Type: ground_fire_extinguish
Task Quality: 4
Task Location:
X: 150
Y: 150
z: 0
*****

*****
```

```
Task ID: 5
Task Type: ground_fire_extinguish
Task Quality: 4
Task Location:
X: 150
Y: 150
z: 0
*****

*****
Ending create_tasks function
*****

*****
Checking capability and calculating visibility sets
*****

*****
Task-1 type is: aerial_fire_extinguish
*****

-----
Robot-1 is capable of tasks:
aerial_fire_extinguish
aerial_rescue
Robot is Capable of doing this task.
Distance of Robot-1 to the task-1 is 70.71067811865476
Visibility of Robot-1 to the task-1 is 0.01414213562373095
-----

-----
Robot-2 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is not capable of doing this task
-----

-----
Robot-3 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is not capable of doing this task
-----

-----
```

```
Robot-4 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is not capable of doing this task
-----

-----
Robot-5 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is not capable of doing this task
-----

*****
Task-2 type is: ground_rescue
*****

-----
Robot-1 is capable of tasks:
aerial_fire_extinguish
aerial_rescue
Robot is not capable of doing this task
-----

-----
Robot-2 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-2 to the task-2 is 70.71067811865476
Visibility of Robot-2 to the task-2 is 0.01414213562373095
-----

-----
Robot-3 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-3 to the task-2 is 70.71067811865476
Visibility of Robot-3 to the task-2 is 0.01414213562373095
-----

-----
Robot-4 is capable of tasks:
ground_fire_extinguish
```

```
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-4 to the task-2 is 70.71067811865476
Visibility of Robot-4 to the task-2 is 0.01414213562373095
-----

-----
Robot-5 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-5 to the task-2 is 70.71067811865476
Visibility of Robot-5 to the task-2 is 0.01414213562373095
-----

*****
Task-3 type is: ground_fire_extinguish
*****

-----
Robot-1 is capable of tasks:
aerial_fire_extinguish
aerial_rescue
Robot is not capable of doing this task
-----

-----
Robot-2 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-2 to the task-3 is 70.71067811865476
Visibility of Robot-2 to the task-3 is 0.01414213562373095
-----

-----
Robot-3 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-3 to the task-3 is 70.71067811865476
Visibility of Robot-3 to the task-3 is 0.01414213562373095
-----

-----
```

```
Robot-4 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-4 to the task-3 is 70.71067811865476
Visibility of Robot-4 to the task-3 is 0.01414213562373095
-----

-----
Robot-5 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-5 to the task-3 is 70.71067811865476
Visibility of Robot-5 to the task-3 is 0.01414213562373095
-----

*****
Task-4 type is: ground_fire_extinguish
*****

-----
Robot-1 is capable of tasks:
aerial_fire_extinguish
aerial_rescue
Robot is not capable of doing this task
-----

-----
Robot-2 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-2 to the task-4 is 70.71067811865476
Visibility of Robot-2 to the task-4 is 0.01414213562373095
-----

-----
Robot-3 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-3 to the task-4 is 70.71067811865476
Visibility of Robot-3 to the task-4 is 0.01414213562373095
-----
```

```

-----
Robot-4 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-4 to the task-4 is 70.71067811865476
Visibility of Robot-4 to the task-4 is 0.01414213562373095
-----

-----
Robot-5 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-5 to the task-4 is 70.71067811865476
Visibility of Robot-5 to the task-4 is 0.01414213562373095
-----

*****
Task-5 type is: ground_fire_extinguish
*****

-----
Robot-1 is capable of tasks:
aerial_fire_extinguish
aerial_rescue
Robot is not capable of doing this task
-----

-----
Robot-2 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-2 to the task-5 is 70.71067811865476
Visibility of Robot-2 to the task-5 is 0.01414213562373095
-----

-----
Robot-3 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-3 to the task-5 is 70.71067811865476
```

```
Visibility of Robot-3 to the task-5 is 0.01414213562373095
-----

-----
Robot-4 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-4 to the task-5 is 70.71067811865476
Visibility of Robot-4 to the task-5 is 0.01414213562373095
-----

-----
Robot-5 is capable of tasks:
ground_fire_extinguish
ground_rescue
Robot is Capable of doing this task.
Distance of Robot-5 to the task-5 is 70.71067811865476
Visibility of Robot-5 to the task-5 is 0.01414213562373095
-----


*****
Visibility sets calculated
*****

*****
Task-1 Relative Quality is: 0.23076923076923078
Task-2 Relative Quality is: 0.3076923076923077
Task-3 Relative Quality is: 0.15384615384615385
Task-4 Relative Quality is: 0.15384615384615385
Task-5 Relative Quality is: 0.15384615384615385
*****

*****
Calculating utility
*****

*****
*****

*****
Calculating utility set for Task-1

Robot-1 has as visibility value with this task
```

```
Utility of Task-1 to the Robot-1 is = 0.003263569759322527

*****

*****
Calculating utility set for Task-2

Robot-2 has as visibility value with this task
Utility of Task-2 to the Robot-2 is = 0.00435142634576337

Robot-3 has as visibility value with this task
Utility of Task-2 to the Robot-3 is = 0.00435142634576337

Robot-4 has as visibility value with this task
Utility of Task-2 to the Robot-4 is = 0.00435142634576337

Robot-5 has as visibility value with this task
Utility of Task-2 to the Robot-5 is = 0.00435142634576337

*****

*****
Calculating utility set for Task-3

Robot-2 has as visibility value with this task
Utility of Task-3 to the Robot-2 is = 0.002175713172881685

Robot-3 has as visibility value with this task
Utility of Task-3 to the Robot-3 is = 0.002175713172881685

Robot-4 has as visibility value with this task
Utility of Task-3 to the Robot-4 is = 0.002175713172881685

Robot-5 has as visibility value with this task
Utility of Task-3 to the Robot-5 is = 0.002175713172881685

*****

*****
Calculating utility set for Task-4

Robot-2 has as visibility value with this task
Utility of Task-4 to the Robot-2 is = 0.002175713172881685

Robot-3 has as visibility value with this task
```

```
Utility of Task-4 to the Robot-3 is = 0.002175713172881685

Robot-4 has as visibility value with this task
Utility of Task-4 to the Robot-4 is = 0.002175713172881685

Robot-5 has as visibility value with this task
Utility of Task-4 to the Robot-5 is = 0.002175713172881685

*****

*****
Calculating utility set for Task-5

Robot-2 has as visibility value with this task
Utility of Task-5 to the Robot-2 is = 0.002175713172881685

Robot-3 has as visibility value with this task
Utility of Task-5 to the Robot-3 is = 0.002175713172881685

Robot-4 has as visibility value with this task
Utility of Task-5 to the Robot-4 is = 0.002175713172881685

Robot-5 has as visibility value with this task
Utility of Task-5 to the Robot-5 is = 0.002175713172881685

*****


*****
Calculating utility probabilities
*****

*****
*****
Calculating utility probabilities for Task-1
*****
Calculating utility probabilities for Task-2
*****
Calculating utility probabilities for Task-3
*****
Calculating utility probabilities for Task-4
*****
Calculating utility probabilities for Task-5

*****
```

```
Utility Probabilities are:
*****

*****
Probability of Task-1 being assigned to the robots are
Robot-1 = 1.0
Robot-2 = -1
Robot-3 = -1
Robot-4 = -1
Robot-5 = -1
*****
Probability of Task-2 being assigned to the robots are
Robot-1 = -1
Robot-2 = 0.25
Robot-3 = 0.25
Robot-4 = 0.25
Robot-5 = 0.25
*****
Probability of Task-3 being assigned to the robots are
Robot-1 = -1
Robot-2 = 0.25
Robot-3 = 0.25
Robot-4 = 0.25
Robot-5 = 0.25
*****
Probability of Task-4 being assigned to the robots are
Robot-1 = -1
Robot-2 = 0.25
Robot-3 = 0.25
Robot-4 = 0.25
Robot-5 = 0.25
*****
Probability of Task-5 being assigned to the robots are
Robot-1 = -1
Robot-2 = 0.25
Robot-3 = 0.25
Robot-4 = 0.25
Robot-5 = 0.25

*****
Allocations decided are:
*****

*****
Assignment probabilities for task-1 are: [1.0, -1, -1, -1, -1]
```

```
Task-1 is assigned to robot-1 with a probability of 1.0
*****

*****
Assignment probabilities for task-2 are: [-1, 0.25, 0.25, 0.25, 0.25]
Task-2 is assigned to robot-2 with a probability of 0.25
*****

*****
Assignment probabilities for task-3 are: [-1, 0.25, 0.25, 0.25, 0.25]
Task-3 is assigned to robot-3 with a probability of 0.25
*****

*****
Assignment probabilities for task-4 are: [-1, 0.25, 0.25, 0.25, 0.25]
Task-4 is assigned to robot-4 with a probability of 0.25
*****

*****
Assignment probabilities for task-5 are: [-1, 0.25, 0.25, 0.25, 0.25]
Task-5 is assigned to robot-5 with a probability of 0.25
*****


*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.27 seconds
Time taken to allocate task-2 was 0.27 seconds
Time taken to allocate task-3 was 0.27 seconds
Time taken to allocate task-4 was 0.28 seconds
Time taken to allocate task-5 was 0.28 seconds

Total time taken to allocate all the tasks was 1.37 seconds
```

9.6 Experiment-1.1: Log result

```
Run-1
```

```

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.17 seconds
Time taken to allocate task-2 was 0.17 seconds
Time taken to allocate task-3 was 0.17 seconds
Time taken to allocate task-4 was 0.17 seconds
Time taken to allocate task-5 was 0.18 seconds

 Total time taken to allocate all the tasks was 0.86 seconds

Run-2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.28 seconds
Time taken to allocate task-2 was 0.28 seconds
Time taken to allocate task-3 was 0.28 seconds
Time taken to allocate task-4 was 0.28 seconds
Time taken to allocate task-5 was 0.29 seconds

 Total time taken to allocate all the tasks was 1.4 seconds

Run-3


*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.15 seconds
Time taken to allocate task-5 was 0.15 seconds
```

```

Total time taken to allocate all the tasks was 0.73 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.14 seconds
Time taken to allocate task-5 was 0.15 seconds

Total time taken to allocate all the tasks was 0.71 seconds

Run - 5

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.14 seconds
Time taken to allocate task-5 was 0.14 seconds

Total time taken to allocate all the tasks was 0.71 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.14 seconds
Time taken to allocate task-5 was 0.15 seconds

 Total time taken to allocate all the tasks was 0.72 seconds

Run - 7

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.26 seconds
Time taken to allocate task-2 was 0.26 seconds
Time taken to allocate task-3 was 0.26 seconds
Time taken to allocate task-4 was 0.26 seconds
Time taken to allocate task-5 was 0.27 seconds

 Total time taken to allocate all the tasks was 1.31 seconds

Run - 8

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.14 seconds
Time taken to allocate task-5 was 0.15 seconds

 Total time taken to allocate all the tasks was 0.72 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:
```

```

Time taken to allocate task-1 was 0.28 seconds
Time taken to allocate task-2 was 0.29 seconds
Time taken to allocate task-3 was 0.29 seconds
Time taken to allocate task-4 was 0.29 seconds
Time taken to allocate task-5 was 0.29 seconds

 Total time taken to allocate all the tasks was 1.44 seconds

Run - 10


*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.25 seconds
Time taken to allocate task-2 was 0.26 seconds
Time taken to allocate task-3 was 0.26 seconds
Time taken to allocate task-4 was 0.26 seconds
Time taken to allocate task-5 was 0.26 seconds

 Total time taken to allocate all the tasks was 1.3 seconds
```

9.7 Experiment-1.2: Log result

```
Run - 1


*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.7 seconds
Time taken to allocate task-2 was 0.7 seconds
Time taken to allocate task-3 was 0.7 seconds
Time taken to allocate task-4 was 0.7 seconds
Time taken to allocate task-5 was 0.7 seconds
Time taken to allocate task-6 was 0.71 seconds
Time taken to allocate task-7 was 0.71 seconds
Time taken to allocate task-8 was 0.71 seconds
Time taken to allocate task-9 was 0.71 seconds
Time taken to allocate task-10 was 0.71 seconds

Total time taken to allocate all the tasks was 7.04 seconds

Run - 2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.59 seconds
Time taken to allocate task-2 was 0.6 seconds
Time taken to allocate task-3 was 0.6 seconds
Time taken to allocate task-4 was 0.6 seconds
Time taken to allocate task-5 was 0.6 seconds
Time taken to allocate task-6 was 0.6 seconds
Time taken to allocate task-7 was 0.6 seconds
Time taken to allocate task-8 was 0.6 seconds
Time taken to allocate task-9 was 0.6 seconds
Time taken to allocate task-10 was 0.61 seconds

Total time taken to allocate all the tasks was 5.99 seconds

Run - 3

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.66 seconds
Time taken to allocate task-2 was 0.66 seconds
Time taken to allocate task-3 was 0.66 seconds
Time taken to allocate task-4 was 0.66 seconds
Time taken to allocate task-5 was 0.66 seconds
Time taken to allocate task-6 was 0.67 seconds
Time taken to allocate task-7 was 0.67 seconds
Time taken to allocate task-8 was 0.67 seconds
Time taken to allocate task-9 was 0.67 seconds
Time taken to allocate task-10 was 0.67 seconds

 Total time taken to allocate all the tasks was 6.64 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.65 seconds
Time taken to allocate task-2 was 0.66 seconds
Time taken to allocate task-3 was 0.66 seconds
Time taken to allocate task-4 was 0.66 seconds
Time taken to allocate task-5 was 0.66 seconds
Time taken to allocate task-6 was 0.66 seconds
Time taken to allocate task-7 was 0.66 seconds
Time taken to allocate task-8 was 0.66 seconds
Time taken to allocate task-9 was 0.66 seconds
Time taken to allocate task-10 was 0.67 seconds

 Total time taken to allocate all the tasks was 6.6 seconds

Run - 5

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.76 seconds
Time taken to allocate task-2 was 0.76 seconds
Time taken to allocate task-3 was 0.77 seconds
Time taken to allocate task-4 was 0.77 seconds
Time taken to allocate task-5 was 0.77 seconds
Time taken to allocate task-6 was 0.77 seconds
Time taken to allocate task-7 was 0.77 seconds
Time taken to allocate task-8 was 0.78 seconds
Time taken to allocate task-9 was 0.78 seconds
Time taken to allocate task-10 was 0.78 seconds
```

```

 Total time taken to allocate all the tasks was 7.71 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.62 seconds
Time taken to allocate task-2 was 0.63 seconds
Time taken to allocate task-3 was 0.63 seconds
Time taken to allocate task-4 was 0.63 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.64 seconds
Time taken to allocate task-7 was 0.64 seconds
Time taken to allocate task-8 was 0.64 seconds
Time taken to allocate task-9 was 0.64 seconds
Time taken to allocate task-10 was 0.64 seconds

 Total time taken to allocate all the tasks was 6.34 seconds

Run - 7

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.62 seconds
Time taken to allocate task-2 was 0.62 seconds
Time taken to allocate task-3 was 0.62 seconds
Time taken to allocate task-4 was 0.62 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.63 seconds
Time taken to allocate task-7 was 0.63 seconds
Time taken to allocate task-8 was 0.63 seconds
Time taken to allocate task-9 was 0.63 seconds
Time taken to allocate task-10 was 0.63 seconds

 Total time taken to allocate all the tasks was 6.26 seconds

```

```
Run - 8

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.72 seconds
Time taken to allocate task-2 was 0.72 seconds
Time taken to allocate task-3 was 0.72 seconds
Time taken to allocate task-4 was 0.72 seconds
Time taken to allocate task-5 was 0.72 seconds
Time taken to allocate task-6 was 0.73 seconds
Time taken to allocate task-7 was 0.73 seconds
Time taken to allocate task-8 was 0.73 seconds
Time taken to allocate task-9 was 0.73 seconds
Time taken to allocate task-10 was 0.73 seconds

 Total time taken to allocate all the tasks was 7.24 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.62 seconds
Time taken to allocate task-2 was 0.62 seconds
Time taken to allocate task-3 was 0.62 seconds
Time taken to allocate task-4 was 0.62 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.63 seconds
Time taken to allocate task-7 was 0.63 seconds
Time taken to allocate task-8 was 0.63 seconds
Time taken to allocate task-9 was 0.63 seconds
Time taken to allocate task-10 was 0.64 seconds

 Total time taken to allocate all the tasks was 6.28 seconds

Run - 10

*****
```

```
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.02 seconds
Time taken to allocate task-2 was 1.02 seconds
Time taken to allocate task-3 was 1.02 seconds
Time taken to allocate task-4 was 1.02 seconds
Time taken to allocate task-5 was 1.02 seconds
Time taken to allocate task-6 was 1.02 seconds
Time taken to allocate task-7 was 1.02 seconds
Time taken to allocate task-8 was 1.03 seconds
Time taken to allocate task-9 was 1.03 seconds
Time taken to allocate task-10 was 1.03 seconds

Total time taken to allocate all the tasks was 10.21 seconds
```

9.8 Experiment-1.3: Log result

```
Run - 1

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.64 seconds
Time taken to allocate task-2 was 1.64 seconds
Time taken to allocate task-3 was 1.64 seconds
Time taken to allocate task-4 was 1.65 seconds
Time taken to allocate task-5 was 1.65 seconds
Time taken to allocate task-6 was 1.65 seconds
Time taken to allocate task-7 was 1.65 seconds
Time taken to allocate task-8 was 1.66 seconds
Time taken to allocate task-9 was 1.66 seconds
Time taken to allocate task-10 was 1.66 seconds
Time taken to allocate task-11 was 1.66 seconds
Time taken to allocate task-12 was 1.66 seconds
Time taken to allocate task-13 was 1.67 seconds
Time taken to allocate task-14 was 1.67 seconds
Time taken to allocate task-15 was 1.68 seconds

Total time taken to allocate all the tasks was 24.84 seconds

Run - 2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.4 seconds
Time taken to allocate task-2 was 1.41 seconds
Time taken to allocate task-3 was 1.41 seconds
Time taken to allocate task-4 was 1.41 seconds
Time taken to allocate task-5 was 1.41 seconds
Time taken to allocate task-6 was 1.41 seconds
Time taken to allocate task-7 was 1.42 seconds
Time taken to allocate task-8 was 1.42 seconds
Time taken to allocate task-9 was 1.42 seconds
Time taken to allocate task-10 was 1.42 seconds
Time taken to allocate task-11 was 1.42 seconds
Time taken to allocate task-12 was 1.44 seconds
Time taken to allocate task-13 was 1.47 seconds
Time taken to allocate task-14 was 1.54 seconds
Time taken to allocate task-15 was 1.56 seconds

Total time taken to allocate all the tasks was 21.54 seconds

Run - 3

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.46 seconds
Time taken to allocate task-2 was 1.47 seconds
Time taken to allocate task-3 was 1.47 seconds
Time taken to allocate task-4 was 1.47 seconds
Time taken to allocate task-5 was 1.47 seconds
Time taken to allocate task-6 was 1.47 seconds
Time taken to allocate task-7 was 1.48 seconds
Time taken to allocate task-8 was 1.48 seconds
Time taken to allocate task-9 was 1.48 seconds
Time taken to allocate task-10 was 1.48 seconds
Time taken to allocate task-11 was 1.49 seconds
Time taken to allocate task-12 was 1.48 seconds
Time taken to allocate task-13 was 1.49 seconds
Time taken to allocate task-14 was 1.49 seconds
Time taken to allocate task-15 was 1.49 seconds

 Total time taken to allocate all the tasks was 22.17 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.56 seconds
Time taken to allocate task-2 was 1.56 seconds
Time taken to allocate task-3 was 1.56 seconds
Time taken to allocate task-4 was 1.57 seconds
Time taken to allocate task-5 was 1.57 seconds
Time taken to allocate task-6 was 1.57 seconds
Time taken to allocate task-7 was 1.57 seconds
Time taken to allocate task-8 was 1.57 seconds
Time taken to allocate task-9 was 1.57 seconds
Time taken to allocate task-10 was 1.58 seconds
Time taken to allocate task-11 was 1.58 seconds
Time taken to allocate task-12 was 1.58 seconds
Time taken to allocate task-13 was 1.58 seconds
Time taken to allocate task-14 was 1.58 seconds
Time taken to allocate task-15 was 1.59 seconds

 Total time taken to allocate all the tasks was 23.6 seconds

Run - 5

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:
```

```

Time taken to allocate task-1 was 1.79 seconds
Time taken to allocate task-2 was 1.79 seconds
Time taken to allocate task-3 was 1.8 seconds
Time taken to allocate task-4 was 1.8 seconds
Time taken to allocate task-5 was 1.8 seconds
Time taken to allocate task-6 was 1.81 seconds
Time taken to allocate task-7 was 1.81 seconds
Time taken to allocate task-8 was 1.81 seconds
Time taken to allocate task-9 was 1.81 seconds
Time taken to allocate task-10 was 1.81 seconds
Time taken to allocate task-11 was 1.82 seconds
Time taken to allocate task-12 was 1.82 seconds
Time taken to allocate task-13 was 1.82 seconds
Time taken to allocate task-14 was 1.82 seconds
Time taken to allocate task-15 was 1.82 seconds

 Total time taken to allocate all the tasks was 27.12 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.58 seconds
Time taken to allocate task-2 was 1.58 seconds
Time taken to allocate task-3 was 1.59 seconds
Time taken to allocate task-4 was 1.59 seconds
Time taken to allocate task-5 was 1.59 seconds
Time taken to allocate task-6 was 1.59 seconds
Time taken to allocate task-7 was 1.59 seconds
Time taken to allocate task-8 was 1.59 seconds
Time taken to allocate task-9 was 1.6 seconds
Time taken to allocate task-10 was 1.6 seconds
Time taken to allocate task-11 was 1.6 seconds
Time taken to allocate task-12 was 1.6 seconds
Time taken to allocate task-13 was 1.6 seconds
Time taken to allocate task-14 was 1.6 seconds
Time taken to allocate task-15 was 1.6 seconds

 Total time taken to allocate all the tasks was 23.9 seconds

```

```
Run - 7

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.24 seconds
Time taken to allocate task-2 was 1.24 seconds
Time taken to allocate task-3 was 1.24 seconds
Time taken to allocate task-4 was 1.28 seconds
Time taken to allocate task-5 was 1.36 seconds
Time taken to allocate task-6 was 1.38 seconds
Time taken to allocate task-7 was 1.38 seconds
Time taken to allocate task-8 was 1.38 seconds
Time taken to allocate task-9 was 1.38 seconds
Time taken to allocate task-10 was 1.38 seconds
Time taken to allocate task-11 was 1.39 seconds
Time taken to allocate task-12 was 1.39 seconds
Time taken to allocate task-13 was 1.39 seconds
Time taken to allocate task-14 was 1.4 seconds
Time taken to allocate task-15 was 1.4 seconds

 Total time taken to allocate all the tasks was 20.23 seconds

Run - 8

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.43 seconds
Time taken to allocate task-2 was 1.44 seconds
Time taken to allocate task-3 was 1.44 seconds
Time taken to allocate task-4 was 1.44 seconds
Time taken to allocate task-5 was 1.44 seconds
Time taken to allocate task-6 was 1.44 seconds
Time taken to allocate task-7 was 1.45 seconds
Time taken to allocate task-8 was 1.45 seconds
Time taken to allocate task-9 was 1.45 seconds
Time taken to allocate task-10 was 1.45 seconds
Time taken to allocate task-11 was 1.45 seconds
Time taken to allocate task-12 was 1.45 seconds
Time taken to allocate task-13 was 1.45 seconds
Time taken to allocate task-14 was 1.46 seconds
Time taken to allocate task-15 was 1.46 seconds

 Total time taken to allocate all the tasks was 21.69 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.44 seconds
Time taken to allocate task-2 was 1.44 seconds
Time taken to allocate task-3 was 1.44 seconds
Time taken to allocate task-4 was 1.45 seconds
Time taken to allocate task-5 was 1.45 seconds
Time taken to allocate task-6 was 1.45 seconds
Time taken to allocate task-7 was 1.45 seconds
Time taken to allocate task-8 was 1.46 seconds
Time taken to allocate task-9 was 1.46 seconds
Time taken to allocate task-10 was 1.46 seconds
Time taken to allocate task-11 was 1.46 seconds
Time taken to allocate task-12 was 1.46 seconds
Time taken to allocate task-13 was 1.46 seconds
Time taken to allocate task-14 was 1.47 seconds
Time taken to allocate task-15 was 1.47 seconds

 Total time taken to allocate all the tasks was 21.82 seconds

Run - 10

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.58 seconds
Time taken to allocate task-2 was 1.58 seconds
Time taken to allocate task-3 was 1.58 seconds
Time taken to allocate task-4 was 1.58 seconds
Time taken to allocate task-5 was 1.59 seconds
Time taken to allocate task-6 was 1.59 seconds
Time taken to allocate task-7 was 1.59 seconds
Time taken to allocate task-8 was 1.59 seconds
Time taken to allocate task-9 was 1.6 seconds
Time taken to allocate task-10 was 1.6 seconds
Time taken to allocate task-11 was 1.6 seconds
Time taken to allocate task-12 was 1.6 seconds
Time taken to allocate task-13 was 1.6 seconds
Time taken to allocate task-14 was 1.6 seconds
Time taken to allocate task-15 was 1.6 seconds

Total time taken to allocate all the tasks was 23.88 seconds
```

9.9 Experiment-2.1: Log result

```
Run - 1

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.16 seconds
Time taken to allocate task-2 was 0.17 seconds
Time taken to allocate task-3 was 0.17 seconds
Time taken to allocate task-4 was 0.17 seconds
Time taken to allocate task-5 was 0.17 seconds

Total time taken to allocate all the tasks was 0.84 seconds

Run - 2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.15 seconds
Time taken to allocate task-2 was 0.15 seconds
Time taken to allocate task-3 was 0.15 seconds
Time taken to allocate task-4 was 0.16 seconds
Time taken to allocate task-5 was 0.16 seconds

 Total time taken to allocate all the tasks was 0.78 seconds

Run - 3

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.15 seconds
Time taken to allocate task-2 was 0.16 seconds
Time taken to allocate task-3 was 0.16 seconds
Time taken to allocate task-4 was 0.16 seconds
Time taken to allocate task-5 was 0.16 seconds

 Total time taken to allocate all the tasks was 0.79 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.15 seconds
Time taken to allocate task-2 was 0.15 seconds
Time taken to allocate task-3 was 0.16 seconds
Time taken to allocate task-4 was 0.16 seconds
Time taken to allocate task-5 was 0.16 seconds

 Total time taken to allocate all the tasks was 0.78 seconds

Run - 5

*****
Time taken to allocate tasks:
*****

```

```
Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.27 seconds
Time taken to allocate task-2 was 0.27 seconds
Time taken to allocate task-3 was 0.28 seconds
Time taken to allocate task-4 was 0.28 seconds
Time taken to allocate task-5 was 0.28 seconds

 Total time taken to allocate all the tasks was 1.38 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.14 seconds
Time taken to allocate task-2 was 0.14 seconds
Time taken to allocate task-3 was 0.14 seconds
Time taken to allocate task-4 was 0.14 seconds
Time taken to allocate task-5 was 0.15 seconds

 Total time taken to allocate all the tasks was 0.72 seconds

Run - 7

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.15 seconds
Time taken to allocate task-2 was 0.16 seconds
Time taken to allocate task-3 was 0.16 seconds
Time taken to allocate task-4 was 0.16 seconds
Time taken to allocate task-5 was 0.16 seconds

 Total time taken to allocate all the tasks was 0.78 seconds

Run - 8

*****
```

```
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.23 seconds
Time taken to allocate task-2 was 0.23 seconds
Time taken to allocate task-3 was 0.23 seconds
Time taken to allocate task-4 was 0.24 seconds
Time taken to allocate task-5 was 0.24 seconds

 Total time taken to allocate all the tasks was 1.17 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.27 seconds
Time taken to allocate task-2 was 0.27 seconds
Time taken to allocate task-3 was 0.27 seconds
Time taken to allocate task-4 was 0.27 seconds
Time taken to allocate task-5 was 0.27 seconds

 Total time taken to allocate all the tasks was 1.35 seconds

Run - 10

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.29 seconds
Time taken to allocate task-2 was 0.29 seconds
Time taken to allocate task-3 was 0.29 seconds
Time taken to allocate task-4 was 0.29 seconds
Time taken to allocate task-5 was 0.29 seconds

 Total time taken to allocate all the tasks was 1.45 seconds
```

9.10 Experiment-2.2: Log result

```
Run - 1

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.75 seconds
Time taken to allocate task-2 was 0.75 seconds
Time taken to allocate task-3 was 0.76 seconds
Time taken to allocate task-4 was 0.76 seconds
Time taken to allocate task-5 was 0.76 seconds
Time taken to allocate task-6 was 0.77 seconds
Time taken to allocate task-7 was 0.77 seconds
Time taken to allocate task-8 was 0.77 seconds
Time taken to allocate task-9 was 0.77 seconds
Time taken to allocate task-10 was 0.77 seconds

Total time taken to allocate all the tasks was 7.64 seconds

Run - 2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.62 seconds
Time taken to allocate task-2 was 0.62 seconds
Time taken to allocate task-3 was 0.63 seconds
Time taken to allocate task-4 was 0.63 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.63 seconds
Time taken to allocate task-7 was 0.63 seconds
Time taken to allocate task-8 was 0.64 seconds
Time taken to allocate task-9 was 0.64 seconds
Time taken to allocate task-10 was 0.64 seconds

Total time taken to allocate all the tasks was 6.3 seconds
```

```

Run - 3

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.69 seconds
Time taken to allocate task-2 was 0.7 seconds
Time taken to allocate task-3 was 0.7 seconds
Time taken to allocate task-4 was 0.7 seconds
Time taken to allocate task-5 was 0.7 seconds
Time taken to allocate task-6 was 0.7 seconds
Time taken to allocate task-7 was 0.7 seconds
Time taken to allocate task-8 was 0.71 seconds
Time taken to allocate task-9 was 0.71 seconds
Time taken to allocate task-10 was 0.71 seconds

 Total time taken to allocate all the tasks was 7.01 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.68 seconds
Time taken to allocate task-2 was 0.69 seconds
Time taken to allocate task-3 was 0.69 seconds
Time taken to allocate task-4 was 0.69 seconds
Time taken to allocate task-5 was 0.69 seconds
Time taken to allocate task-6 was 0.69 seconds
Time taken to allocate task-7 was 0.7 seconds
Time taken to allocate task-8 was 0.7 seconds
Time taken to allocate task-9 was 0.7 seconds
Time taken to allocate task-10 was 0.7 seconds

 Total time taken to allocate all the tasks was 6.93 seconds

Run - 5

```

```
*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.63 seconds
Time taken to allocate task-2 was 0.63 seconds
Time taken to allocate task-3 was 0.63 seconds
Time taken to allocate task-4 was 0.63 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.63 seconds
Time taken to allocate task-7 was 0.64 seconds
Time taken to allocate task-8 was 0.64 seconds
Time taken to allocate task-9 was 0.64 seconds
Time taken to allocate task-10 was 0.64 seconds

 Total time taken to allocate all the tasks was 6.34 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.49 seconds
Time taken to allocate task-2 was 0.5 seconds
Time taken to allocate task-3 was 0.5 seconds
Time taken to allocate task-4 was 0.5 seconds
Time taken to allocate task-5 was 0.5 seconds
Time taken to allocate task-6 was 0.5 seconds
Time taken to allocate task-7 was 0.51 seconds
Time taken to allocate task-8 was 0.51 seconds
Time taken to allocate task-9 was 0.51 seconds
Time taken to allocate task-10 was 0.51 seconds

 Total time taken to allocate all the tasks was 5.02 seconds

Run - 7

*****
Time taken to allocate tasks:
*****
```

```

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.82 seconds
Time taken to allocate task-2 was 0.82 seconds
Time taken to allocate task-3 was 0.82 seconds
Time taken to allocate task-4 was 0.83 seconds
Time taken to allocate task-5 was 0.83 seconds
Time taken to allocate task-6 was 0.83 seconds
Time taken to allocate task-7 was 0.84 seconds
Time taken to allocate task-8 was 0.84 seconds
Time taken to allocate task-9 was 0.84 seconds
Time taken to allocate task-10 was 0.84 seconds

 Total time taken to allocate all the tasks was 8.31 seconds

Run - 8

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.8 seconds
Time taken to allocate task-2 was 0.8 seconds
Time taken to allocate task-3 was 0.8 seconds
Time taken to allocate task-4 was 0.8 seconds
Time taken to allocate task-5 was 0.81 seconds
Time taken to allocate task-6 was 0.81 seconds
Time taken to allocate task-7 was 0.81 seconds
Time taken to allocate task-8 was 0.81 seconds
Time taken to allocate task-9 was 0.81 seconds
Time taken to allocate task-10 was 0.81 seconds

 Total time taken to allocate all the tasks was 8.06 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

```

```
Time taken to allocate task-1 was 0.63 seconds
Time taken to allocate task-2 was 0.63 seconds
Time taken to allocate task-3 was 0.63 seconds
Time taken to allocate task-4 was 0.63 seconds
Time taken to allocate task-5 was 0.63 seconds
Time taken to allocate task-6 was 0.64 seconds
Time taken to allocate task-7 was 0.64 seconds
Time taken to allocate task-8 was 0.64 seconds
Time taken to allocate task-9 was 0.64 seconds
Time taken to allocate task-10 was 0.65 seconds

Total time taken to allocate all the tasks was 6.36 seconds

Run - 10

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 0.74 seconds
Time taken to allocate task-2 was 0.74 seconds
Time taken to allocate task-3 was 0.75 seconds
Time taken to allocate task-4 was 0.75 seconds
Time taken to allocate task-5 was 0.75 seconds
Time taken to allocate task-6 was 0.75 seconds
Time taken to allocate task-7 was 0.75 seconds
Time taken to allocate task-8 was 0.75 seconds
Time taken to allocate task-9 was 0.76 seconds
Time taken to allocate task-10 was 0.76 seconds

Total time taken to allocate all the tasks was 7.5 seconds
```

9.11 Experiment-2.3: Log result

```
Run - 1

*****
Time taken to allocate tasks:
*****
```

```

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.11 seconds
Time taken to allocate task-2 was 1.11 seconds
Time taken to allocate task-3 was 1.11 seconds
Time taken to allocate task-4 was 1.12 seconds
Time taken to allocate task-5 was 1.12 seconds
Time taken to allocate task-6 was 1.12 seconds
Time taken to allocate task-7 was 1.12 seconds
Time taken to allocate task-8 was 1.13 seconds
Time taken to allocate task-9 was 1.13 seconds
Time taken to allocate task-10 was 1.13 seconds
Time taken to allocate task-11 was 1.13 seconds
Time taken to allocate task-12 was 1.13 seconds
Time taken to allocate task-13 was 1.13 seconds
Time taken to allocate task-14 was 1.13 seconds
Time taken to allocate task-15 was 1.14 seconds

 Total time taken to allocate all the tasks was 16.86 seconds

Run - 2

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.22 seconds
Time taken to allocate task-2 was 1.22 seconds
Time taken to allocate task-3 was 1.23 seconds
Time taken to allocate task-4 was 1.23 seconds
Time taken to allocate task-5 was 1.23 seconds
Time taken to allocate task-6 was 1.23 seconds
Time taken to allocate task-7 was 1.23 seconds
Time taken to allocate task-8 was 1.23 seconds
Time taken to allocate task-9 was 1.24 seconds
Time taken to allocate task-10 was 1.24 seconds
Time taken to allocate task-11 was 1.24 seconds
Time taken to allocate task-12 was 1.24 seconds
Time taken to allocate task-13 was 1.25 seconds
Time taken to allocate task-14 was 1.25 seconds
Time taken to allocate task-15 was 1.25 seconds

```

```
Total time taken to allocate all the tasks was 18.52 seconds

Run - 3

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.32 seconds
Time taken to allocate task-2 was 1.32 seconds
Time taken to allocate task-3 was 1.32 seconds
Time taken to allocate task-4 was 1.33 seconds
Time taken to allocate task-5 was 1.33 seconds
Time taken to allocate task-6 was 1.33 seconds
Time taken to allocate task-7 was 1.34 seconds
Time taken to allocate task-8 was 1.34 seconds
Time taken to allocate task-9 was 1.34 seconds
Time taken to allocate task-10 was 1.34 seconds
Time taken to allocate task-11 was 1.35 seconds
Time taken to allocate task-12 was 1.35 seconds
Time taken to allocate task-13 was 1.35 seconds
Time taken to allocate task-14 was 1.35 seconds
Time taken to allocate task-15 was 1.36 seconds

Total time taken to allocate all the tasks was 20.08 seconds

Run - 4

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.3 seconds
Time taken to allocate task-2 was 1.3 seconds
Time taken to allocate task-3 was 1.31 seconds
Time taken to allocate task-4 was 1.31 seconds
Time taken to allocate task-5 was 1.31 seconds
Time taken to allocate task-6 was 1.31 seconds
Time taken to allocate task-7 was 1.31 seconds
Time taken to allocate task-8 was 1.32 seconds
Time taken to allocate task-9 was 1.32 seconds
Time taken to allocate task-10 was 1.32 seconds
Time taken to allocate task-11 was 1.32 seconds
Time taken to allocate task-12 was 1.32 seconds
Time taken to allocate task-13 was 1.32 seconds
Time taken to allocate task-14 was 1.33 seconds
Time taken to allocate task-15 was 1.33 seconds

 Total time taken to allocate all the tasks was 19.73 seconds

Run - 5

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.48 seconds
Time taken to allocate task-2 was 1.48 seconds
Time taken to allocate task-3 was 1.48 seconds
Time taken to allocate task-4 was 1.49 seconds
Time taken to allocate task-5 was 1.49 seconds
Time taken to allocate task-6 was 1.49 seconds
Time taken to allocate task-7 was 1.49 seconds
Time taken to allocate task-8 was 1.5 seconds
Time taken to allocate task-9 was 1.5 seconds
Time taken to allocate task-10 was 1.5 seconds
Time taken to allocate task-11 was 1.5 seconds
Time taken to allocate task-12 was 1.5 seconds
Time taken to allocate task-13 was 1.5 seconds
Time taken to allocate task-14 was 1.51 seconds
Time taken to allocate task-15 was 1.51 seconds

 Total time taken to allocate all the tasks was 22.42 seconds

Run - 6

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.18 seconds
Time taken to allocate task-2 was 1.18 seconds
Time taken to allocate task-3 was 1.19 seconds
Time taken to allocate task-4 was 1.19 seconds
Time taken to allocate task-5 was 1.19 seconds
Time taken to allocate task-6 was 1.19 seconds
Time taken to allocate task-7 was 1.2 seconds
Time taken to allocate task-8 was 1.2 seconds
Time taken to allocate task-9 was 1.2 seconds
Time taken to allocate task-10 was 1.2 seconds
Time taken to allocate task-11 was 1.21 seconds
Time taken to allocate task-12 was 1.21 seconds
Time taken to allocate task-13 was 1.21 seconds
Time taken to allocate task-14 was 1.21 seconds
Time taken to allocate task-15 was 1.22 seconds

 Total time taken to allocate all the tasks was 17.98 seconds

Run - 7

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 2.19 seconds
Time taken to allocate task-2 was 2.19 seconds
Time taken to allocate task-3 was 2.19 seconds
Time taken to allocate task-4 was 2.2 seconds
Time taken to allocate task-5 was 2.2 seconds
Time taken to allocate task-6 was 2.2 seconds
Time taken to allocate task-7 was 2.2 seconds
Time taken to allocate task-8 was 2.21 seconds
Time taken to allocate task-9 was 2.22 seconds
Time taken to allocate task-10 was 2.26 seconds
Time taken to allocate task-11 was 2.27 seconds
Time taken to allocate task-12 was 2.27 seconds
Time taken to allocate task-13 was 2.27 seconds
Time taken to allocate task-14 was 2.28 seconds
Time taken to allocate task-15 was 2.28 seconds

 Total time taken to allocate all the tasks was 33.43 seconds

Run - 8

*****
```

```
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.38 seconds
Time taken to allocate task-2 was 1.38 seconds
Time taken to allocate task-3 was 1.38 seconds
Time taken to allocate task-4 was 1.39 seconds
Time taken to allocate task-5 was 1.39 seconds
Time taken to allocate task-6 was 1.39 seconds
Time taken to allocate task-7 was 1.39 seconds
Time taken to allocate task-8 was 1.4 seconds
Time taken to allocate task-9 was 1.4 seconds
Time taken to allocate task-10 was 1.4 seconds
Time taken to allocate task-11 was 1.4 seconds
Time taken to allocate task-12 was 1.4 seconds
Time taken to allocate task-13 was 1.4 seconds
Time taken to allocate task-14 was 1.41 seconds
Time taken to allocate task-15 was 1.41 seconds

 Total time taken to allocate all the tasks was 20.91 seconds

Run - 9

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.12 seconds
Time taken to allocate task-2 was 1.12 seconds
Time taken to allocate task-3 was 1.12 seconds
Time taken to allocate task-4 was 1.12 seconds
Time taken to allocate task-5 was 1.13 seconds
Time taken to allocate task-6 was 1.13 seconds
Time taken to allocate task-7 was 1.13 seconds
Time taken to allocate task-8 was 1.13 seconds
Time taken to allocate task-9 was 1.14 seconds
Time taken to allocate task-10 was 1.14 seconds
Time taken to allocate task-11 was 1.14 seconds
Time taken to allocate task-12 was 1.14 seconds
Time taken to allocate task-13 was 1.14 seconds
Time taken to allocate task-14 was 1.14 seconds
Time taken to allocate task-15 was 1.15 seconds

 Total time taken to allocate all the tasks was 16.99 seconds

Run - 10

*****
Time taken to allocate tasks:
*****

Time taken to allocate for particular tasks:

Time taken to allocate task-1 was 1.16 seconds
Time taken to allocate task-2 was 1.16 seconds
Time taken to allocate task-3 was 1.16 seconds
Time taken to allocate task-4 was 1.16 seconds
Time taken to allocate task-5 was 1.16 seconds
Time taken to allocate task-6 was 1.17 seconds
Time taken to allocate task-7 was 1.17 seconds
Time taken to allocate task-8 was 1.17 seconds
Time taken to allocate task-9 was 1.17 seconds
Time taken to allocate task-10 was 1.17 seconds
Time taken to allocate task-11 was 1.18 seconds
Time taken to allocate task-12 was 1.18 seconds
Time taken to allocate task-13 was 1.18 seconds
Time taken to allocate task-14 was 1.18 seconds
Time taken to allocate task-15 was 1.18 seconds

 Total time taken to allocate all the tasks was 17.55 seconds
```

References

[1] G. Antonelli, F. Arrichiello, and S. Chiaverini, The null-space-based behavioral control for mobile robots, in 2005 International Symposium on Computational Intelligence in Robotics and Automation, IEEE, 2005, pp. 15–20.

[2] G. Beni and J. Wang, Swarm intelligence in cellular robotic systems, in Robots and Biological Systems: Towards a New Bionics?, Springer, 1993, pp. 703–712.

[3] K. Bernhard and J. Vygen, Combinatorial optimization: Theory and algorithms, Springer, Third Edition, 2005., (2008).

[4] P. Brucker, Scheduling algorithms, Journal-Operational Research Society, 50 (1999), pp. 774–774.

[5] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape, and M. Yim, Indoor automation with many mobile robots, in EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, IEEE, 1990, pp. 67–72.

[6] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, Multi-robot task-allocation through vacancy chains, in 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422), vol. 2, IEEE, 2003, pp. 2293–2298.

[7] P. Davidsson, S. Johansson, and M. Svahnberg, Characterization and evaluation of multi-agent system architectural styles, in International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Springer, 2005, pp. 179–188.

[8] M. De Longueville, A Course in Topological Combinatorics, Springer Science & Business Media, 2012.

[9] M. B. Dias, Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments, Robotics Institute, (2004), p. 153.

[10] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, Market-based multirobot coordination: A survey and analysis, Proceedings of the IEEE, 94 (2006), pp. 1257–1270.

[11] B. P. Gerkey, R. Mailler, and B. Morisset, Commbots: Distributed control of mobile communication relays, in AAAI Workshop on Auction Mechanisms for Robot Coordination, 2006, pp. 51–57.

[12] B. P. Gerkey and M. J. Mataric, A framework for studying multi-robot task allocation, (2003).

[13] B. P. Gerkey and M. J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, The International journal of robotics research, 23 (2004), pp. 939–954.

[14] S. Giordani, M. Lujak, and F. Martinelli, A distributed algorithm for the multi-robot task allocation problem, in International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer, 2010, pp. 721–730.

[15] T. Gunn and J. Anderson, Dynamic heterogeneous team formation for robotic urban search and rescue, Journal of Computer and System Sciences, 81 (2015), pp. 553–567.

[16] A. Jevtic, A. Gutiérrez, D. Andina, and M. Jamshidi, Distributed bees algorithm for task allocation in swarm of robots, IEEE Systems Journal, 6 (2011), pp. 296–304.

[17] X. Jia and M. Q.-H. Meng, A survey and analysis of task allocation algorithms in multi-robot systems, in 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), IEEE, 2013, pp. 2280–2285.

[18] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz, Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks, in Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., IEEE, 2006, pp. 570–575.

[19] N. Kalra and A. Stentz, A market approach to tightly-coupled multi-robot coordination: First results, in Proceedings of the ARL Collaborative Technologies Alliance Symposium, 2003.

[20] A. Khamis, A. Hussein, and A. Elmogy, Multi-robot task allocation: A review of the state-of-the-art, Cooperative Robots and Sensor Networks 2015, (2015), pp. 31–51.

[21] J. Kiener and O. Von Stryk, Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots, Robotics and Autonomous Systems, 58 (2010), pp. 921–929.

[22] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, and B. Stewart, Centibots: Very large scale distributed robotic teams, in Experimental Robotics IX, Springer, 2006, pp. 131–140.

[23] G. A. Korsah, A. Stentz, and M. B. Dias, A comprehensive taxonomy for multi-robot task allocation, The International Journal of Robotics Research, 32 (2013), pp. 1495–1512.

[24] H. W. Kuhn, The Hungarian method for the assignment problem, Naval research logistics quarterly, 2 (1955), pp. 83–97.

[25] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. J. Kleywegt, S. Koenig, C. A. Tovey, A. Meyerson, and S. Jain, Auction-Based Multi-Robot Routing., in Robotics: Science and Systems, vol. 5, Rome, Italy, 2005, pp. 343–350.

[26] P. Langley, J. E. Laird, and S. Rogers, Cognitive architectures: Research issues and challenges, Cognitive Systems Research, 10 (2009), pp. 141–160.

[27] T. Lemaire, R. Alami, and S. Lacroix, A distributed tasks allocation scheme in multi-UAV context, in IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, vol. 4, IEEE, 2004, pp. 3622–3627.

[28] M. T. Long, R. R. Murphy, and L. E. Parker, Distributed multi-agent diagnosis and recovery from sensor failures, in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 3, IEEE, 2003, pp. 2506–2513.

[29] A. R. Mosteo and L. Montano, A survey of multi-robot task allocation, Instituto de Investigacin en Ingenier\la de Aragn (I3A), Tech. Rep, (2010).

[30] A.-I. Mouaddib, Multi-objective decision-theoretic path planning, in IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, vol. 3, IEEE, 2004, pp. 2814–2819.

[31] E. Nettleton, H. Durrant-Whyte, and S. Sukkarieh, A robust architecture for decentralised data fusion, in Proc. of the International Conference on Advanced Robotics (ICAR), 2003.

[32] L. E. Parker, D. Rus, and G. S. Sukhatme, Multiple mobile robot systems, in Springer Handbook of Robotics, Springer, 2016, pp. 1335–1384.

[33] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, The bees algorithm—a novel tool for complex optimisation problems, in Intelligent Production Machines and Systems, Elsevier, 2006, pp. 454–459.

[34] Y. Rizk, M. Awad, and E. W. Tunstel, Cooperative heterogeneous multi-robot systems: A survey, ACM Computing Surveys (CSUR), 52 (2019), pp. 1–31.

[35] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach Pearson, London, (2010).

[36] A. Sanfeliu, Cooperative robotics in urban sites, in Interantional Symposium on Network Robot Systems, 2008.

[37] J. Schneider, D. Apfelbaum, D. Bagnell, and R. Simmons, Learning opportunity costs in multi-robot market based planners, in Proceedings of the 2005 IEEE International Conference on Robotics and Automation, IEEE, 2005, pp. 1151–1156.

[38] L. S. Shapley, Stochastic games, Proceedings of the national academy of sciences, 39 (1953), pp. 1095–1100.

[39] A. Stentz and M. B. Dias, A free market architecture for coordinating multiple robots, tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1999.

[40] P. Stone and M. Veloso, Task decomposition and dynamic role assignment for real-time strategic teamwork, in International Workshop on Agent Theories, Architectures, and Languages, Springer, 1998, pp. 293–308.

[41] ——, Multiagent systems: A survey from a machine learning perspective, Autonomous Robots, 8 (2000), pp. 345–383.

[42] A. Viguria, I. Maza, and A. Ollero, SET: An algorithm for distributed multirobot task allocation with dynamic negotiation based on task subsets, in Proceedings 2007 IEEE International Conference on Robotics and Automation, IEEE, 2007, pp. 3339–3344.

[43] ——, S+ T: An algorithm for distributed multirobot task allocation based on services for improving robot cooperation, in 2008 IEEE International Conference on Robotics and Automation, IEEE, 2008, pp. 3163–3168.

[44] R. Zlot and A. Stentz, Market-based multirobot coordination for complex tasks, The International Journal of Robotics Research, 25 (2006), pp. 73–101.