A Decentralized Strategy for Swarm Robots to Manage Spatially Distributed Tasks

by

Rohit Sheth

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfilment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

by

May 2017

APPROVED:

Professor Carlo Pinciroli, Thesis Advisor

Professor Jie Fu, Thesis Committee Member

Professor Eugene Eberbach, Thesis Committee Member

Abstract

Large-scale scenarios such as search-and-rescue operations, agriculture, warehouse, surveillance, and construction consist of multiple tasks to be performed at the same time. These tasks have non-trivial spatial distributions. Robot swarms are envisioned to be efficient, robust, and flexible for such applications. We model this system such that each robot can service a single task at a time; each task requires a specific number of robots, which we refer to as 'quota'; task allocation is instantaneous; and tasks do not have inter-dependencies. This work focuses on distributing robots to spatially distributed tasks of known quotas in an efficient manner. Centralized solutions which guarantee optimality in terms of distance travelled by the swarm exist. Although potentially scalable, they require non-trivial coordination; could be computationally expensive; and may have poor response time when the number of robots, tasks and task quotas increase. For a swarm to efficiently complete tasks with a short response time, a decentralized approach provides better parallelism and scalability than a centralized one. In this work, we study the performance of a weight-based approach which is enhanced to include spatial aspects. In our approach, the robots share a common table that reports the task locations and quotas. Each robot, according to its relative position with respect to task locations, modifies weights for each task and randomly chooses a task to serve. Weights increase for tasks that are closer and have high quota as opposed to tasks which are far away and have low quota. Tasks with higher weights have a higher probability of being selected. This results in each robot having its own set of weights for all tasks. We introduce a distance-bias parameter, which determines how sensitive the system is to relative robot-task locations over task quotas. We focus on evaluating the distance covered by the swarm, number of inter-task switches, and time required to completely allocate all tasks and study the performance of our approach in several sets of simulated experiments.

Acknowledgements

I would like to express my gratitude to my advisor Prof. Carlo Pinciroli for his guidance, direction, criticism, and enthusiasm during the course of this work. I thank him for introducing me to the wonderful and elegant world of swarm robotics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots, if not already, will soon become a part of everyday life for humans. Robots can be thought of as advanced tools created by humans. Robots are designed with three intentions in mind: to aid humans in performing tasks; to outperform humans; or to perform tasks which humans are incapable of undertaking.

Research in robotics includes the field of multi-robot systems (MRS) where robots can execute multiple tasks simultaneously or significantly improve the performance over one executed by a single robot. MRS can be classified based on local or complete awareness; centralized and decentralized decision-making; and local or global communication. Robotic swarms is a branch of MRS which concentrates on decentralized decision making, local awareness and local communication for each robot in the swarm. Robotic swarms focus on an emergent behaviour from simple interactions between robots and the environment. Motivation for robotic swarm arose from examples in nature such as ant-colonies, flocking of birds, etc.

The first mention of robotic swarm can be traced back to 1986 when Craig Reynolds developed a program called 'Boids'. 'Boids', a contraction of 'bird-oid object', is a computer simulation that mimics the flocking behaviour in birds. Swarm robots are not dependent on an individual robot, thus the death of a robot or addition of a robot does not affect collective behaviour. Robotic swarms adapt well to change in environmental conditions. Design complexity is reduced as building multiple simplistic robots with varying abilities is easier compared to building one powerful robot. Robotic swarms execute one or multiple tasks in parallel which improves the overall efficiency of the system. Robotic swarm systems are designed to be inherently scalable and achieve similar behaviour within acceptable changes

in swarm size. This makes swarm robust, scalable, efficient, simplistic, and flexible.

It is with these advantages that swarming robots are anticipated to be effective in search and rescue, surveillance, mining, agriculture, construction and warehouse applications. Such applications span over large area and require multiple tasks to be executed simultaneously. Additionally the requirement of the number of robots; sensing and actuation requirements of tasks; location; and duration is dynamic and non-deterministic. In such situations, swarm systems can efficiently complete these tasks with a short response time and provide better parallelism and scalability.

The above mentioned applications often have multiple smaller tasks distributed in space which together form a big task. Robotic swarms are efficient when performing multiple small tasks in order to serve a bigger purpose. Consider an example of spraying pesticides over a partially disease affected field. It is important to identify all regions with diseased crop, carry pesticide to affected regions, and spray in an area around the affected region. Limited battery life is a bottleneck when covering large fields. Therefore it is vital to minimize distance travelled when performing robot-task assignment. Additionally, multiple robots are required to spray pesticide over a substantially large affected region. Such an assignment for hundreds of robots is not intuitive and therefore task allocation must be studied in swarms.

Task allocation by itself is a vast field. Some of the areas where task allocation is important communication networks, multi-processor systems, operations research and management, surveillance and security, service based industry, and multi-robot systems. Clients, allocators and performers is one way to visualize task allocation. Clients request for task to be performed, allocators process task requests and assign tasks to performers who execute the tasks. Task allocation is difficult as the number of ways in which tasks can be assigned to performers scale logarithmically. Therefore, design of a fast, and efficient allocator to generate is not always possible.

Communication is a means to achieve a multi-robot system with a single allocator in the above mentioned applications. However, loss of communication, non-deterministic robot failure, and erroneous sensor data further complicate the design of an efficient allocator. Such a system is also expected to provide fast response to the aforementioned problems and handle dynamic requests for tasks. This is the primary reason to turn towards inspiration in nature from social insects such as ants, termites, and bees where performers have a say in the task assignment process and the system does not depend on a single allocator. The intention of this work

is to provide a framework for a decentralized task allocation strategy that allows performers to make independent decisions. The work, then, focuses on exploiting the spatial information of tasks to improve the strategy and evaluates the performance in different sets of simulated experiment.

1.1 Problem statement

The main focus of this work is provide a decentralized strategy for task allocation that makes use of the spatial task information in robotic swarms. The difficulty of task allocation increases when number of robots required by tasks are non-deterministic, total number of robots and tasks is high. The aim of developing such a strategy is to facilitate a scalable, fast, and efficient solution. The strategy focuses on utilizing spatial information about tasks and the number of robots required to achieve efficiency and a decentralized approach to achieve scalability.

The problem is to divide a group of robots into smaller sub-groups and assign these sub-groups to various tasks. The number of sub-groups is equal to the number of tasks and the size of sub-group assigned to a task is equal to number of robots required in the task. The tasks are static in time, space, and have constant task size. The robotic swarm is made of homogeneous robots that are capable of serving one task at a time.

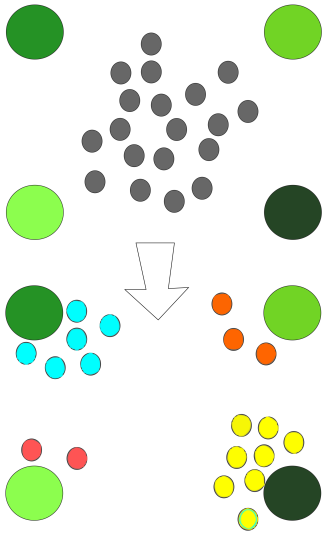

Figure 1.1: Problem Statement Illustration

Figure 1.1 illustrates the problem statement. Considering the fact that top half of the figure indicates and initial stage where tasks are the green coloured circles and robots are clustered in the centre as grey dots. Darkness of the shade of green is proportional to the number of robots required by the task. The problem at hand is to allocate the grey dots in smaller sub-groups and assign the sub-groups to tasks as shown by the figure below. The sub-groups are highlighted with different colour to show assignment to different tasks.

1.2 Outline

Chapter 2 introduces the multi-robot task allocation problem and explores various taxonomies to categorize the problem. Understanding the problem category aids in modelling the problem with existing techniques and also relates better with existing literature. First part of Chapter 3 describes a framework to perform decentralized task allocation in swarm robots. The focus of this work is on utilizing spatial task information for task allocation and hence simplifications made to the proposed framework are described in this chapter. The second part describes in detail the proposed design for improving task allocation. Chapter 4 explores the metrics for evaluating task allocation and also describes the parameters used in setting up various experiments. The chapter concludes with a section on results, comparison and analysis. Chapter 5 is the final chapter which follows up analysis with concluding remarks and points the reader in directions this work can be expanded in the future.

Chapter 2

Literature

Section 2.1 introduces two taxonomies that provide a broad overview of Multi-robot Task Allocation (MRTA). Section 2.2 formulates the problem statement, introduces the concept of coalitions, and analyses the complexity of the problem. Section 2.3 discusses multiple approaches to model the problem using existing methods. Section 2.4 highlights existing research in MRTA relevant to robotic swarms.

2.1 Taxonomy for Task Allocation

Taxonomy in MRTA aim at categorizing MRTA problems based on robot and task specifications. This provides an improved understanding of the problem by comparing it with existing mathematical formulations and models for a particular category of problem.

The taxonomy of Gerkey (2003) concentrates on measuring a robot's ability to perform tasks and also temporal nature of tasks. This is a benchmark taxonomy for understanding problems in MRTA. Korsah et al. (2013) studied the interrelated constraints and dependencies between robots and tasks and added a layer of dependencies upon Gerkey's taxonomy. Nunes et al. (2016) proposed a taxonomy which extended Gerkey's taxonomy to include hard and soft constraints on task deadlines and ordering of tasks. Gerkey's and Korsah's taxonomies are sufficient to categorize the problem and are explored in further detail in the following section.

2.1.1 Gerkey's Taxonomy

Utility Factor

Gerkey defined a *Utility* Factor U as a measure of a robot's ability to perform a task. Utility factor is defined by considering two factors, cost C of performing a task, and quality Q of the task.

Given a robot R and a task T, if R is capable of executing T, and if cost of executing the task C_{RT} and the quality Q_{RT} is defined, then utility factor of the task is given as

$$U_{RT} = \begin{cases} Q_{RT} - C_{RT}, & \text{if } R \text{ is capable of executing } T \\ 0, & \text{otherwise} \end{cases}$$

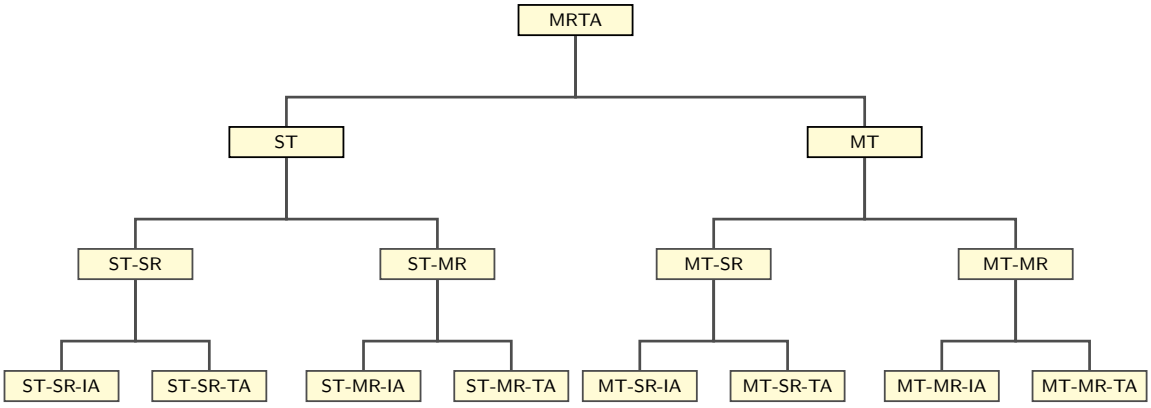

Figure 2.1: Gerkey's Taxonomy

Criteria for Gerkey's Taxonomy

Gerkey divides MRTA problems (see Figure 2.1) based on the following three criteria:

- **Single-Task vs Multi-Task robots [ST-MT]**: ST robots indicates that each robot in MRS can execute only one task at any given time while MT robots can perform multiple tasks simultaneously.
- **Single-Robot vs Multi-Robot tasks [SR-MR]**: SR tasks indicate that only one robot is required to completely perform a task while MR tasks require more than one robot to complete a task.

- **Instantaneous Assignment vs Time-extended Assignment [IA-TA]**: Instantaneous allocation refers to the allocation process performed using information about currently existing tasks and robot states without planning, while Time-extended assignment takes into account the task information, addition and deletion of tasks, and robot states at a future point in time.

ST-SR Domain:

The problems in ST-SR-IA can be modelled as Optimal Assignment Problem, where given a set of robots and a set of tasks, assign at most one task to each robot such that sum of Utility Factor 2.1.1 for all robots is maximised. The ST-SR-TA version of the problem takes into account *future utilities* of robots and tasks. This problem is proven to be *NP-hard*.

One of the ways to look at this problem is to model the *time-extended cost* schedule for each robot by assigning weights and further minimizing the total weighted cost. Another approach is to first solve the initial problem as optimal assignment problem and secondly use a greedy algorithm to assign remaining tasks. This method basically ignores the time-extended element and approximates the problem as an iterated ST-SR-IA problem. However, such an approach is feasible when the number of robots is greater than the number of tasks.

ST-MR / MT-SR Domain:

The ST-MR-IA problem (formally known as *Coalition Problem*) splits a set of robots R is into disjoint sets (called '*coalitions*') where the number of coalitions is equal to the number of tasks and each coalition is assigned to at most one task. The splitting of a set into disjoint sets such that the union of split sets is the original set is also known as the Set Partitioning Problem (Hoffman and Padberg, 2001) . The complexity of this problem is *strongly NP-hard* and is studied in further detail in Section 2.2.2.

A decentralized strategy to solve ST-MR-IA related task allocation problems that returns an efficient solution in response time and total distance travelled is the focus of this work. The ST-MR-IA problem can be modelled in a number of ways as seen in Section 2.3. Further details about existing methods and strategies to address Coalition Formation Problem is mentioned in Section 2.4.

The ST-MR-TA problem includes both, coalition formation and schedule

components. This problem is *NP-Hard*. Like the ST-SR-TA solution, modelling an iterative solution that repeatedly solves ST-MR-IA for small time steps.

The category of MT-SR-IA and The MT-SR-TA problems allows robots to perform multiple tasks simultaneously while each task requires only one robot. The analysis of this problem is similar to those its ST-MR counterparts with robots and tasks interchanged in the problem formulation.

MT-MR Domain:

When robots simultaneously perform multiple tasks that requires more than one robots for complete execution, the problem belongs to MT-MR-IA category. The ST-MR-IA category was modelled as a Set Partition or Coalition Formation Problem. However, in the MT-MR-IA problem, the set of robots R is split in non-disjoint sets. The subsets of R can now be intersecting sets and is popularly known in mathematics as the *Set Covering Problem*. This problem is strongly *NP-hard*. The MT-MR-TA problem is an instance of scheduling problem with Set Covering problem.

2.1.2 Korsah's Taxonomy

Gerkey's taxonomy is important to understand different flavours of MRTA, however, it does not capture all the MRTA problems. Gerkey's taxonomy is unable to account for problems with interrelated utilities and constraints. Korsah et al. (2013) gives the example of a multi-Travelling Salesman Problem (mTSP) where robots have to visit multiple target locations which results in utilities (see 2.1.1) being related to the cost of the route instead.

Gerkey's definition of utility is unable to accurately handle the problem statement in Section 1.1 where the utility for a robot-task pair depends on the actions of other robots. For example, task t_i requires j robots and if more than j robots (say k) are assigned to the task. Gerkey's definition assigns positive utility for all k robots. However, only the j robots that arrive first and begin performing the task have positive utility while the remaining $k - j$ robots' utility is modelled incorrectly.

Korsah et al. (2013) introduced the definition of *effective* utility ${}^{e}U$ to model utility for a subset of robots and is given by

$$U_{RT} = \sum_{r \in R} \sum_{t \in T} {}^{e}U_{rt}^{RT}$$

Korsah states that the equality does not hold for problems with interrelated utilities. Korsah proposes a modified version of taxonomy called **iTax** that introduces an additional layer on Gerkey's taxonomy (see Figure 2.2). The additional layer is useful to address the interrelated utilities and constraints.

Task Decomposition

Task decomposition enables division of tasks into sub-tasks which can be performed as individual tasks or further partitioned in smaller tasks. Pini et al. (2011) propose self-organised task decomposition in which swarm decides whether to partition a task into sub-tasks. The additional layer previously mentioned includes the task decomposition terminology provided by Zlot and Stentz (2006) .

- **Simple Task:** A task at its lowest or *atomic* level and cannot be broken down into sub-tasks.
- **Compound Task:** A task that can be completely decomposed into a set of simple tasks.
- **Complex Task:** Task decomposition takes place in a set of multi-allocatable subtasks in at least one way. These subtasks may be simple, compound or complex where a complex task can have multiple complete decompositions.

iTax Layer:

Based on the decomposition of tasks and the definition of utility for a subset of robots, Korsah categorizes the intertask dependencies in the following manner:

- **No dependencies:** Tasks are simple or compound with independent agent-task utilities in this category. In other words, the effective utility of an agent for a task doesn't depend on any other tasks or robots in the system.

- **In-Schedule Dependencies:** The tasks in this set of problems have scheduling dependencies. The utilities for a robot depend on the order of tasks performed by a robot. Constraints placed on task schedules affect the utility of the individual robot. The nature of tasks remains simple or compound.

- **Cross-schedule Dependencies:** The set of problems where the utilities for robots depend on the tasks executed by other robots, and on the order of tasks performed by the robot itself fall in this category. Constraints that exist between schedules of multiple robots and optimization of individual robot schedule cannot be decoupled from that of other robots. The nature of tasks remains simple or compound.

- **Complex Dependencies:** This is the only case when the nature of tasks is complex. These are task allocation problems where utilities depend on the interrelated task schedules between robots which is additionally dependent on the decomposition chosen for the complex tasks.

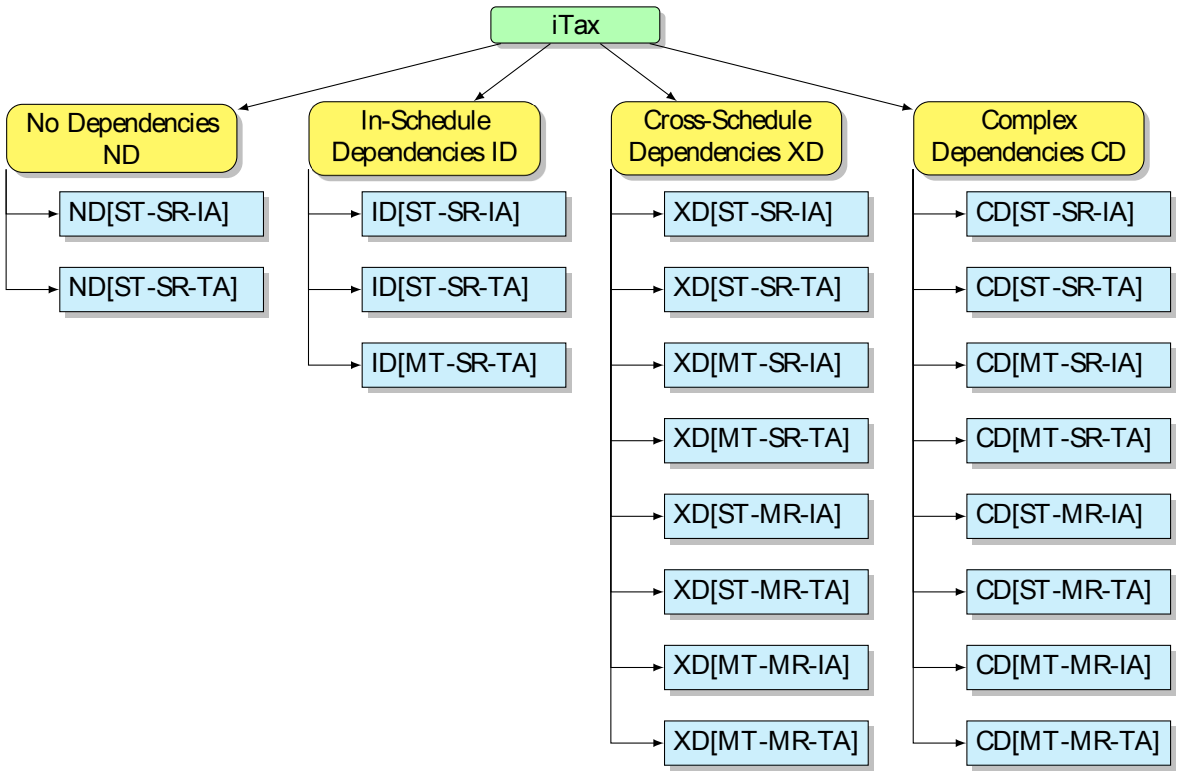

Figure 2.2: iTax: Korsah's Taxonomy for MRTA

2.2 Problem Formulation

If the nature of tasks is known, then the problem of task allocation has two aspects (McLurkin and Yamins, 2005) :

1. Calculating the number of robots required by each task
2. Splitting the set of robots in sub-teams and assigning a sub-team to a task

The number of robots required for each task is provided to the system in order to simplify the problem. With reference to Section 2.1, the second of the above mentioned aspects is analogous to Coalition Formation Problem or Set Partitioning Problem. Shehory and Kraus (1998) studied task allocation as a coalition formation problem with inspiration from Distributed Artificial Intelligence methods present in 1990s. Coalition Problem is a well studied problem, especially in the areas of Operations Research(Padberg, 1972) and mathematics. Section 2.2.2 shows the difficulty in solving the Coalition Formation Problem.

2.2.1 Mathematical Formulation

Coalition: A non-empty set C is said to be a coalition of set A if

$$C \subseteq A \ ,$$

Coalition Structure: A coalition structure (CS) is a partition of set A, into disjoint, exhaustive coalitions. Each element from set A belongs to only one coalition set C_i and the union of coalition sets is equal to set A.

$$A = \bigcup C_i$$

Let us consider an example with three robots as shown in Figure 2.3.
$N_c = 7$: {1,2,3},{1,2},{2,3},{1,3},{1},{2},{3}
$N_{CS} = 5$:[{1},{2},{3}],[{1,2},{3}],[{2,3},{1}], [{1,3},{2}],[{1,2,3}]

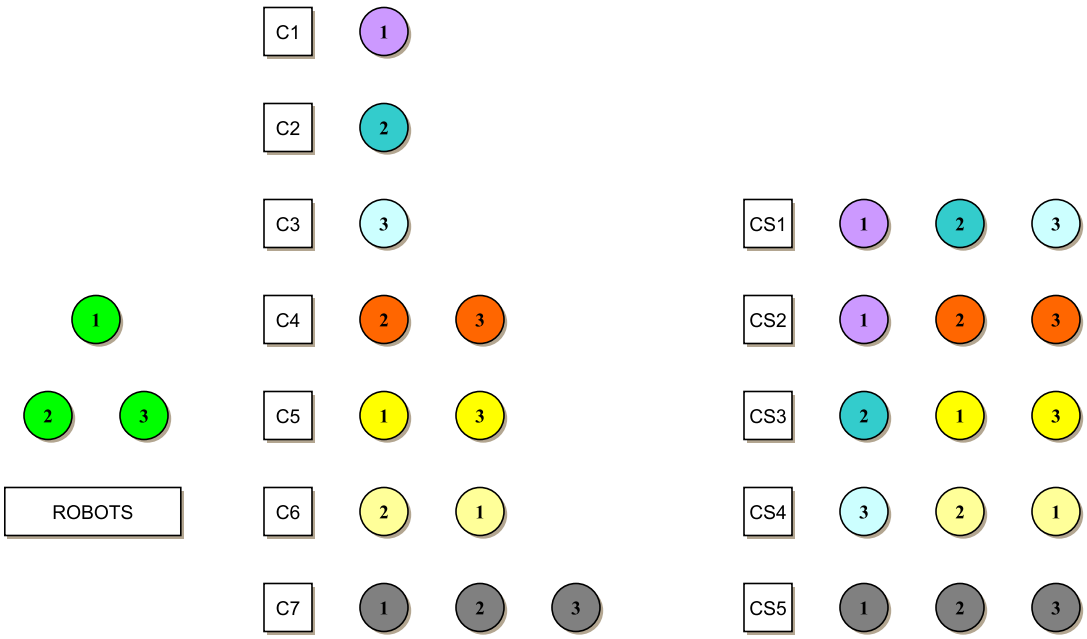

Figure 2.3: Coalitions and Coalition Structures

Formulation

Given

A set of n Robots $R = r_1, r_2...r_n$

A set of m Tasks $T = t_1, t_2...t_m$

For a set of Task Sizes $N = n_1, n_2...n_m$ where $\sum_{i=1}^{m} n_i = n$

At m Locations $L = (x_1, y_1), (x_m, y_m), ...(x_m, y_m)$

The set of robots R must be divided into m Coalitions $s.t.$

Set of Coalitions $C = C_1, C_2...C_m$ form a Coalition Structure

where Coalition $C_i \Rightarrow T_i$

while minimizing total distance travelled D and time taken for complete allocation of all tasks S

$$D = \sum_{i=1}^{n}(d_i) \quad ; d_i = \text{distance travelled by } r_i$$

$$S = \max(s_1, s_2, s_3.....s_m) \; ; \; s_i = \text{time required for allocation of task } t_i$$

2.2.2 Complexity Analysis

The aim of the Coalition Formation problem is to select the right coalition for a given task under certain conditions. The complexity of the problem lies in the number of coalition structures that can be formed. Sandholm et al. (1999) coalition structures and is briefly described below.

Focus is initially given to the number of coalitions before selecting a coalition structure. A coalition is a non-empty subset of elements from a superset of all elements (also referred to as '*agents* in Game Theory). This is a combinatorial problem where number of ways to form a coalition of a given size s is $\binom{n}{s}$. Thus the total number of Coalitions N_c is given by

$$N_c = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} ..... \binom{n}{n} = \sum_{i=1}^{n} \binom{n}{i} \text{ where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$N_c = 2^n - 1 \text{ (Sum of Binomial Coefficients)}$$

The number of coalition structures scale rapidly in comparison to number of Coalitions. The total number of coalition structures is

$$\sum_{i=1}^{n} Z(n,i)$$

where $Z(n,i)$ is the number of coalition structures formed by i coalitions. The following recurrence sum provides an alternate way to capture the number of coalition

$$Z(n,i) = iZ(n-1,i) + Z(n-1,i-1),$$

$$Z(n,n) = Z(n,1) = 1$$

Consider a game of $(n-1)$ agents where the first term, $iZ(n-1,i)$ counts the number of coalition structures formed by adding a new agent to existing coalitions. The addition of a new agent is done in i ways for existing i coalitions. The second term $Z(n-1,i-1)$ considers adding the agent in a coalition of its own, and therefore considers existing coalition structures that are formed with $(i-1)$ coalitions.

Sandholm et al. state that the number of coalition structures is $O(n^n)$ and $\omega(n^{n/2})$. Figure 2.4 illustrates how $2^n - 1$, n^n, and $n^{n/2}$ scale with respect to n.

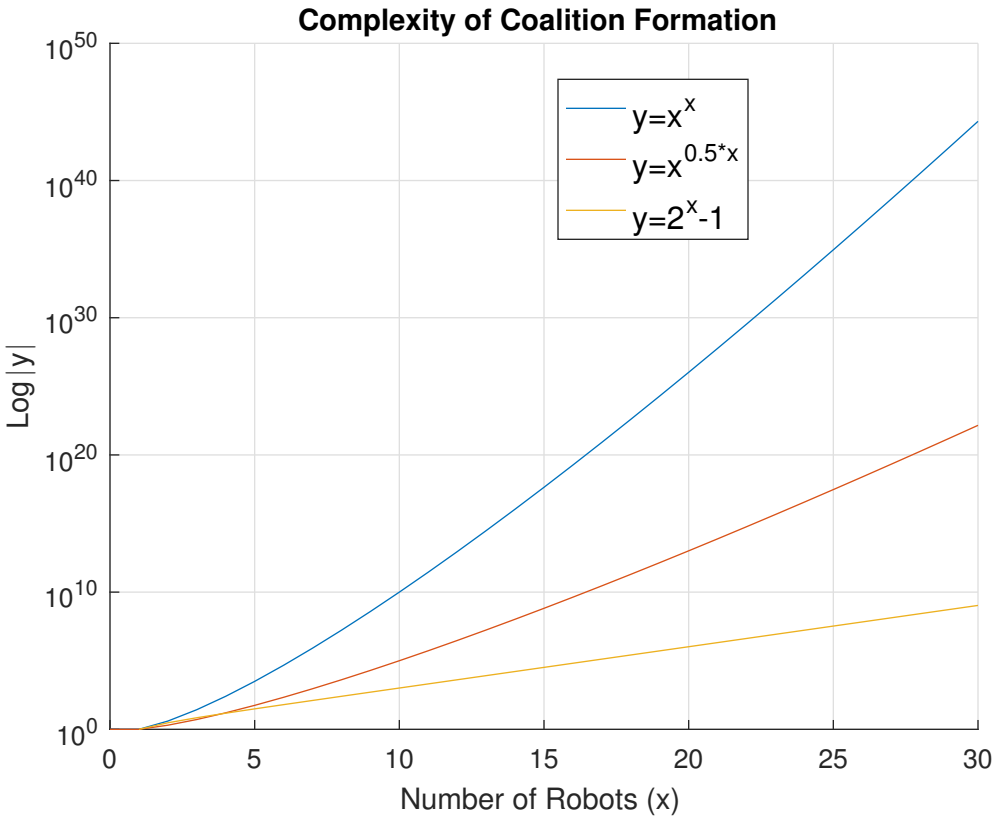

Figure 2.4: Complexity of Coalition Formation

2.3 Modelling Coalition Formation Problem

Coalition Formation Problem can be formulated as another existing problem. This helps in implementing existing techniques to solve the coalition formation. Popular ways to model the problem are explored in this section.

2.3.1 Multiple Travelling Salesman Problem

Multiple Travelling Salesman Problem (mTSP) is a generalized case of the popular Travelling Salesman Problem(TSP). In TSP, an optimal route must be computed for a single salesman to visit multiple cities. For the mTSP, there are m salesmen travelling to n cities where the total route of m salesmen is optimized.

Coalition Formation is modelled as mTSP (Bektas, 2006) where robots play the role of salesmen and tasks are modelled as cities. The formulation includes two modifications: the number of salesmen that visit each city can be more than one; and the tour ends once a salesman visits a city. In order to remove the first modification, the number of tasks spawned at a location are equal to the task size. This helps to retain the initial formulation of each salesman visiting one city.

A popular solution for solving the mTSP is Kuhn (1955) 's *Hungarian Method* for assignment problems . Dorigo and Gambardella (1997) used swarm intelligence technique called *Ant Colony Optimization* to solve the TSP. Zhang et al. (2006) improved upon the Ant Colony Optimization and proposed a method to solve mTSP using the Ant Colony Optimization. Bektas (2006) covers various methods of modelling and solving the mTSP.

2.3.2 Optimal Assignment Problem

Optimal Assignment Problem is a classical problem where there are m workers and n weighted jobs, where each job requires one worker. Each worker has the ability to estimate it's non-negative efficiency to perform a task. The efficiency of every robot-task pair is taken in to account to perform optimal assignment that maximises performance of the system.

Optimal assignment in the case of MRTA can be observed as robots being workers and tasks in place of jobs where a robot can perform multiple tasks and a task may require multiple robots. The system should have the ability to estimate the efficiency of robots to perform tasks. One example, is to use the *Utility* definition and maximize utility.

Optimal assignment problems can also be modelled as minimization problems where the total cost to perform tasks, time required or distance travelled is minimized.

2.3.3 Multi-robot Task Allocation as an Economic Game

A broker sells tasks in the system where the cost of task j is c_j. Robot i acts in a greedy manner and submits a bid h_{ij} for task j. Every robot bids for all tasks in the market. The problem at hand is setting the initial price p_i of task, that is higher than broker value c_i but not so high that robot bids would not lead to purchase of the task.

A market is assumed to be at equilibrium when no two robots purchase the same task. Multiple instances of the same task are sold for tasks that require multiple robots. Profits must be made by all agents at equilibrium to reach an optimal solution. The broker is empowered to resolve conflicts when more than one robots attempt to purchase a particular task. Conflict consume valuable time in assignment process and therefore it is important to model bidding of robots such that conflicts are reduced. This concept of task marketplace gave rise to auction based centralized and decentralized methods.

Gerkey and Matarić (2002) proposed an auction-based task allocation system MURDOCH, for multi-robot dynamic task allocation. The purpose of MURDOCH is to show that distributed negotiation for auction based systems are effective for coordinating multi-robot systems. Zlot and Stentz (2006) modelled an auction based approach for *complex tasks* where individually work on smaller partitions of the task to complete a *complex task*.

Vig and Adams (2006) introduce RACHNA system to handle dynamic tasks in the ST-MR problem category. The system creates two types of agents, a task agent to auction new tasks and service agents to bid on tasks. Vig and Adams (2006) modified *Utility* definition to facilitate bids based on *utility* value for such a market based approach. Choi et al. (2009) proposed a consensus based-bundle algorithm (CBAA) for decentralized multi-robot auctions. CBBA is able to resolve conflicting winning bids via local interactions.

2.4 State of the Art for Decentralized Allocation of Spatially Distributed Tasks

Jevtić et al. (2012) proposed a *Distributed Bees Algorithm* to address the problem of task allocation. The highlight of this algorithm is that decision-making is distributed and robot choose assignments autonomously. Such a system does not depend on coordination with other robots and robots make decisions purely on the knowledge of tasks requirements and task locations.

Ak and Akn (2016) work on a hybrid approach to solve spatial task allocation problems. Tasks are initially clustered based on relative distances to robots. Robots are then assigned to the best cluster based on optimization for distance travelled by all robots. Robots independently plan for the tasks present in the assigned clusters.

Ducatelle et al. (2009) proposed an algorithm in which a client announces tasks to selective robots in space. The aim is to divide tasks among other robots in the swarm. Two methods are used to achieve task assignment: one relies on simple attraction and repulsion to light, while in the other method, task information is spread to the swarm via local communication with a gossip based model.

Parker et al. (2016) 's work focused on dividing heterogeneous team of robots into dynamic task teams to handle time dependent tasks. Parker's word accounted for uncertainties present in robot and task locations, robot life and communication errors.

Claes et al. (2015) used a Multi-agent Markov Decision Framework to allocate spatially distributed tasks. Initially, the robots estimate a suitable task assignment and then estimate the task assignment for other robots in the system. Thus a robot predicts the quantity in which all tasks that will be serviced and makes a decision based on the prediction.

Khaluf and Rammig (2013) focused on the temporal aspect of task allocation while considering task size. The author introduces the idea of allocating tasks based on probability matrix for each task. The probabilities of each task at a particular time instance are dependent on task size, task deadline, and number of robots in the task at that instance.

Di Paola et al. (2015) propose a decentralized model for task allocation for heterogeneous robots. The subset of robots that meet skill requirements to execute a task is formed. Task assignment is carried out by a consensus among the subset of

robots. Selection criteria is modelled based on distance from the task and urgency (termed as *fieriness*).

Mottola et al. (2014) presented a team-level programming model system called VOLTRON which uses a distributed hash-table to perform dynamic task allocation for drones. Space is divided into grids and actions(tasks) are specified in form of spatial variables for each location in space.

Dantu et al. (2011) demonstrate 'Karma', a system to program a swarm of micro-aerial vehicles (MAVs). These MAVs update spatial information in form of tuple space at a centralized host which is the central scheduler for the system. Importance of spatial information in swarm system is highlighted despite the centralized nature of the work.

2.5 Drawbacks

This section is devoted to analysing the drawbacks and shortcomings of the models and methods mentioned in Section 2.3 and Section 2.4 respectively.

Challenges in mTSP:

Solving the mTSP using traditional methods such as Kuhn (1955) 's algorithm and its extension results in computation time of $\mathcal{O}(n^3)$ (where n is the number of cities). This limits scalability unless response time for the solution is relaxed considerably in impractical for hundreds of robots and task.Thus as the size of the problem increases, exact methods consume become too slow and approximate methods are the only option. Ant Colony Optimization is an approximate method with no guarantees on optimality of the solution. The quality of the solution provided by ACO improves with the number of iterations performed. Additionally, according to Zhang et al. (2006) 's method, mTSP has to be decomposed as a TSP and different decompositions are provided to each ant. Such a solution puts a higher demand on the number of iterations required.

Bottlenecks in Auction/Market-based Algorithms:

Complexity analysis for auction based methods by Kalra et al. (2006) shows that computation complexity of auction based algorithms depends upon the processes of

bid valuation, winner determination, and *number of auctions* while the communication complexity depends on the phases of *auction call, bid submission,* and *decision announce phase.* The complexity of *winner determination* is worst at $\mathcal{O}(r.n^2)$ where n is the number of tasks and r is the number of robots that bid for the item. Additionally, after each auction, m robots are assigned to tasks and therefore *the total number of auctions* are $\frac{n}{m}$. Bottlenecks in communication complexity of auction methods is *auction call* phase with $\mathcal{O}(r.n)$ and *bid submission* phase with $\mathcal{O}(r.n)$.

Decentralized Approaches that do not use Spatial Task Information

Among decentralized approaches, Khaluf and Rammig (2013) 's work focuses on biasing weights based on task size, task deadlines, and currently present robots in the task but do not consider the spatial distribution of tasks. Ducatelle et al. (2009) 's work in multi-robot task assignment depends on perception of light intensity and a gossip based algorithm, however robots do not use spatial information of task to improve on task selection.

Shortcomings of work that use Spatial Task Information

Di Paola et al. (2015) and Claes et al. (2015) make use of spatial information of tasks to improve on task selection. However in both methods, additional information of other robots' locations, decisions (true or predicted), and ability to execute a task is taken into account. This invokes the need of communication of additional information and increased local interaction with other robots. The work in this paper enables robots to make decisions independently without any knowledge of other robots in the system.

Ak and Akn (2016) and Dantu et al. (2011) 's work include spatial information of tasks in assigning robots to task. In both methods, assignment to tasks is performed by a centralized system that is made aware of task locations. This is also the case with Mottola et al. (2014) 's work with VOLTRON programming system, where drones are connected to a centralized system using Wifi..

Novelty

The work in this papers presents a decentralized framework that allows robots to independently make decisions and perform self allocation of tasks. This aids

in removing the bottleneck of scalability as each robot makes decisions by itself. Additionally, the robots do not depend on selection performed by other robots and thus bottlenecks present in communication for task assignment are removed. Thus the response of the system for task assignment is instantaneous. Due to the lack of information about the decisions taken by other robots, the resultant assignment is sub-optimal. We make use of the spatial information to improve on the sub-optimality of the solution and explore how to bias tasks selection with task size and task locations. Additionally, we conduct an extensive simulations in various spatial topologies and study the behaviour of the system in detail.

Chapter 3

Approach

In Chapter 2, the complexity of the forming coalitions is discussed. Additionally, drawbacks of existing systems are noted in Section **??**. In this chapter, a framework for allocating spatially distributed tasks and a design for task allocation algorithm is discussed. However, the framework realizes a complex emergent behaviour which makes it hard to study the task allocation strategy. Therefore, simplifications (Section 3.1.6) are made to the system in order to study spatial task allocation in detail. The design of spatial task allocation is studied in Section 3.2.

3.1 Framework

Figure 3.1: Framework

The framework in Figure 3.1 describes a decentralized system in which a robotic swarm discovers task in the environment (see Section 3.1.1), communicates task information (see Section 3.1.2), performs task allocation (see Section 3.1.3), navigation (see Section 3.1.4) and keeps a count of robots (see Section 3.1.5) performing each task. Each part of the framework is important for the swarm to function independently.

3.1.1 Spatial Task Information Management

In Chapter 1, numerous examples of spatially distributed tasks are mentioned. Knowledge about task information such as task location, task size, task deadlines, required sensing and actuation, and scheduling dependencies are essential to perform all tasks. The focus of this research is on spatially distributed tasks for a homogeneous swarm with no deadlines or inter-task scheduling dependencies. Therefore this subsection is focused on inferring spatial and task size information.

Discovering Spatially Distributed Tasks

The aim of this subsection is to understand the concept of discovering spatially distributed tasks. Each task is characterised by task quota and task location. Ants and other social animals infer task information from the environment. Information about the presence of food is validated with smell, taste, touch, and sight. Along with the location, the size of food source needs to be communicated. Ants communicate food location by laying a pheromone trail from the food source to the nest. Mailleux et al. (2000) showed that ants have a higher probability of laying a trail if the volume of a food source exceeds a threshold. This results in a global emergent behaviour where the size of a food source corresponds to the recruitment of ants.

Similarly, robots are equipped with various sensors such light sensors, proximity sensors, camera, audio receivers, and GPS to deduce existence, location and task size from environment. Such a wide array of sensing abilities allow for multiple ways to detect task locations.

Some examples in swarm literature include:

1. The Swarmanoid Project (Dorigo et al., 2013) used camera mounted on indoor flying robots (eye-bot) to search a desired object in the environment. The eye-bots then communicate location to via connected swarm network to foot-bots (see Section 4.2.7) and hand-bot(robots capable of climbing and manipulating small objects) which cooperate to retrieve the object.

2. Ducatelle et al. (2009) used different coloured lights to create attractive and repulsive potentials. The foot-bots use camera to detect and calculate an attraction potential to the position of a yellow light, and a repulsive potential from the position of green light. The required recruitment of robots is controlled by modifying the number of yellow and green lights at a task position.

3. Habibi et al. (2016) demonstrate two distributed methods by using proximity sensors and local communications where robots cooperate to estimate 2D geometry of objects placed on the ground.

4. Li et al. (2017) detected task by projecting different lights on robots. The robotic swarm then attempted to camouflage according to incident light.

In this experiment, robots are required to move in the arena. The arena floor is equipped with lights and change in grey-level value from floor colour indicate a task location. The robots detect task quotas by sensing the grey level values on the floor and task location is naturally set at the location where the robot discovered the task. However, such a detection of task locations and task quotas to invoke the need to recruit robots to execute tasks.

3.1.2 Recruitment

Once tasks are discovered and requirement of agents is estimated, it is vital to spread this information across the swarm. The spread of task information aids in recruiting robots to perform tasks. The roles are divided as *performer*, *messenger*, and *worker*. The main goal of the *forager* robots is to discover spatially distributed tasks and recruit robots to perform tasks. The role of the *messenger* robots is to spread task information using local communication across the swarm in order to recruit robots. The *performer* robots execute the tasks (see Figure3.2).

Forager

Each robot starts off as a *forager* robot and switches role to either a *messenger* or a *performer* robot. Initially, the robot has no task information and moves randomly in search of tasks. While exploring discovering tasks (see Section 3.1.1), the robot also listens to any local broadcast messages from other other robots. If a *forager* robot receives task information from a *messenger* robot, the *forager* robot then takes role of a *messenger* robot. If it discovers a new task and does not encounter a *messenger* robot, the forager robot takes the role of *performer* at the discovered task.

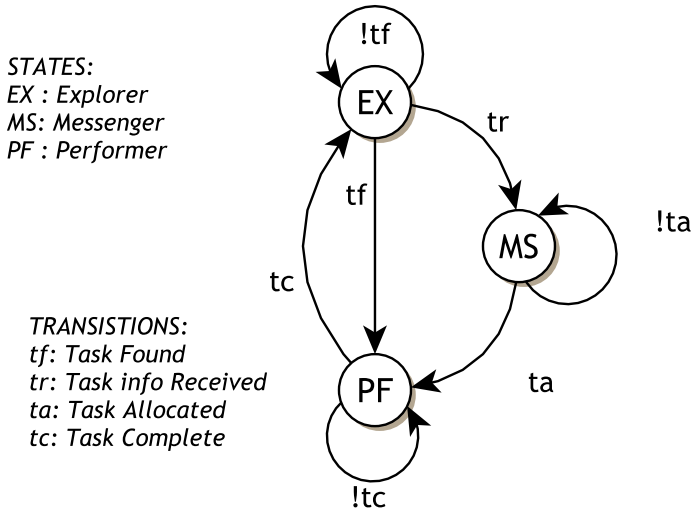

Figure 3.2: Recruitment

Messenger

As a robot switches the role to a *messenger* it does not discover a task successfully but receives task information from a nearby broadcasting robot. As a *messenger*, the robot continues to accumulates new task information received from other *messenger* and *performer* robots. Additionally, the *messenger* robot keeps broadcasting all known task information within the local communication range for a certain amount of time. Once a certain amount of time is passed, the robot allocates itself a task to perform based on the task selection strategy and proceeds to the task location.

Performer

In this experiment, the only task executed by a *performer* robot is to wait inside a radius around task location and to keep broadcasting all known task information. Additionally, the robot continues to accumulate information about new tasks from passing *messenger* robots or other nearby *performer* robots.

Virtual Stigmergy:

Though the concept of *virtual stigmergy* is not explored in this work, the concept refers to modifications made to the environment to transfer information. The inspiration behind *virtual stigmergy* arises from social insects (eg. ants laying a *phermone* trail). Pinciroli et al. (2015) explored the concept of *virtual stigmergy* by allowing robots to share (*key,value*) pairs using tuples. Previously described works of Dantu et al. (2011) and Mottola et al. (2014) use the concept of tuple spaces to model variables distributed in space. The concept of *virtual stigmergy* is important as it is an active area of research in robotic swarm and an effective means for information transfer.

3.1.3 Spatial Bias Strategy for Task Selection

If a robot in a messenger state receives information about multiple tasks, then after a threshold time has passed, it must make a decision to choose a task from the available list of tasks. The strategy is a decentralized strategy since every robot selects a task independently.

Task Selection

Each task is characterized by task size and spatial location. A robot selects a number randomly between 0 and 1. Each task is weighted based on the number of required robots and these weights are then added together and normalized. This strategy assigns intervals proportional to task size for each robot. When number of robots and number of tasks are high, such a strategy ensures that roughly the required number of robots are assigned to tasks. The the task selection strategy is scalable and facilitates quick decision making.

Spatial Bias

The task selection method does not take into account task locations and therefore robots can end up choosing far away tasks. It is therefore necessary to take into account the relative location of these tasks with respect to a robot's locations.

Spatial bias is used to modify weights locally on a robot utilizing task location information. Even though weights generated from task size are propagated equally within the swarm; the weights assigned by a robot to each task depend on its relative distance to tasks. This strategy results in different robots having different weights for the same set of tasks. The resultant effect is that tasks that are closer, and require high number of robots have increased weights and tasks that are far away, and require low number of robots have reduced weights.

Figure 3.3 shows an example of 3 robots adjusting weights according to relative distance to each task. Column 1 named 'common' shows that all tasks require equal number of robots and thus have the same weights. Columns 1, 2, and 3 show weights biased using spatial information by individual robots 1, 2, and 3 respectively.

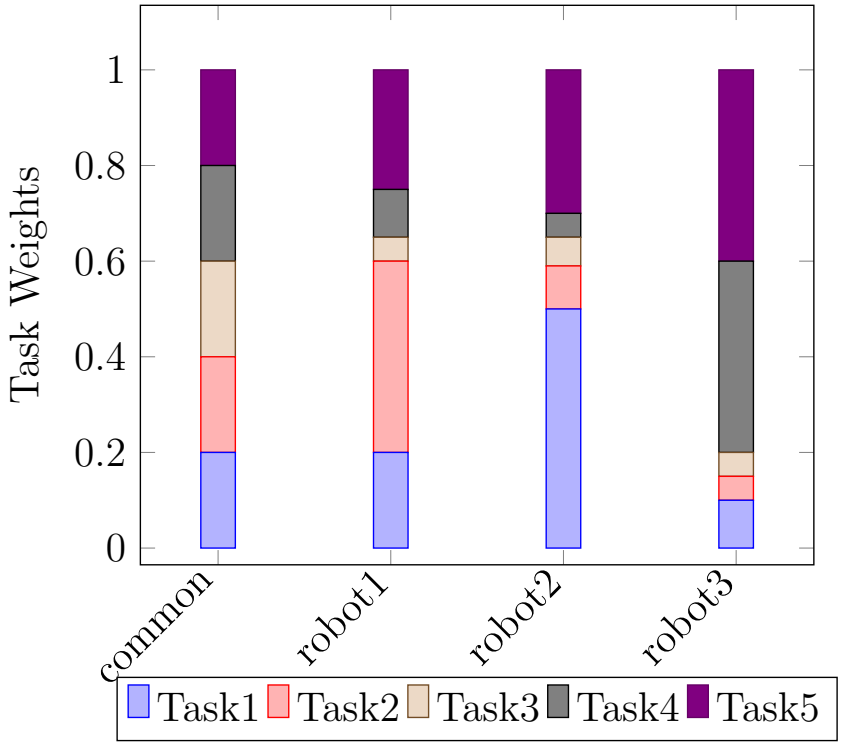

Figure 3.3: Spatial Bias

3.1.4 Navigation

Robots have to move in the environment for searching tasks, communicating task information with other robots, and travelling to task location. Robots diffuse in the arena when performing the role of *forager* or *messenger*. When a robot allocates itself to a task, the robot must go to the task location. A potential field approach is used to attract robots to task locations. This approach results in optimal paths in an arena with no obstacles.

Collision Avoidance

Collision avoidance is vital for any robot. Foot-bots 4.2.7 have 24 proximity sensors arranged uniformly on the peripheral circle. The proximity sensor informs about the range and bearing of obstacles up to 10 cm away from the robot. Readings from all sensors are added together to get the resultant vector of collision. For each reading from the proximity sensors, an opposite direction vector is calculated. The magnitude of vector is more sensitive to obstacles that are closer as compared to those far away. The magnitude is calculated with the aid of Lennard-Jones potential. The magnitude of repulsion from Lennard-Jones potential for a target distance x_t is given as:

$$f(x) = -\frac{g}{x}\left(\left(\frac{x_t}{x}\right)^{2n} - \left(\frac{x_t}{x}\right)^{n}\right)$$

where x_t = Target distance
x = Distance of collision
g = Gain factor
n = Exponent

Target distance is the range of proximity sensor, therefore any reading from the proximity sensors for $x > x_t$ are not possible and potential for noisy readings is truncated to 0 (shown by the red dot in Figure 3.4). Gain factor (g) is set to 100 and Exponent(n) is set to 2.

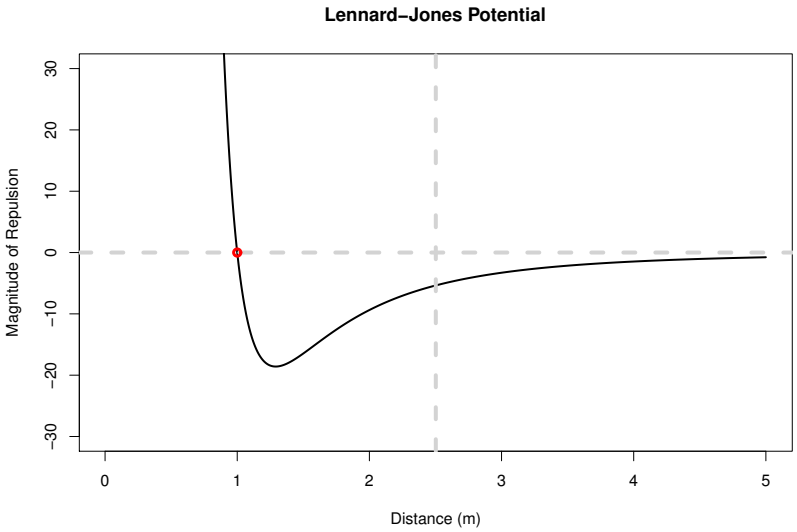

Figure 3.4: Lennard-Jones Potential

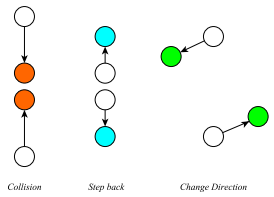

Figure 3.5: Collision Avoidance

In order to avoid collision, the foot-bot is made to move for short amount of time T_a in the opposite direction to collision. After which the foot-bot is made to turn and move in another direction for another short duration T_b and then foot-bot continues either diffusing if the robot state is *forager* or *messenger*, or moving to task location if the robot is a *performer*.

When two robots are in imminent collision, both of them perform collision avoidance. This results in both robots moving away from each other followed by both robots moving in different directions.T_a and T_b are tuned to ensure that both robots move sufficiently away from each other and do not encounter repeated collisions.

3.1.5 Counting Robots

Local counting of robots is essential to decide whether to make redundant robots leave a task or ask for more robots to execute a task. A simple algorithm to count robots is to maintain a list of robot IDs on each robot. However, maintaining a list is an expensive operation in a swarm as each robot must broadcast its ID to all robots, IDs must be shared and propagated throughout the swarm. Additionally, the ID list must be checked for each received message and updated in the case of new IDs. Brambilla (2009) work shows three decentralized algorithms to estimate group size.

A good decentralized counting algorithm returns the count quickly, does not require ether high number or large size communication messages and requires less computations and memory storage. Spanning tree serves as a viable solution as it satisfies listed requirements.

Spanning Trees

Robots performing a task are assumed to have a connected network among them. Such a connected network ensures that each robot can communicate directly or indirectly with every other robot. Thus by randomly selecting a robot to perform as a root node, we can build a spanning tree in the network which is a connected graph.

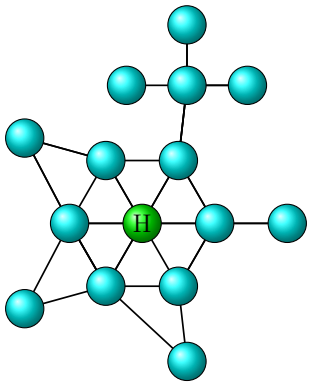

Figure 3.6: Connected Network of Robots

Broadcast Phase

The spanning tree is built in the broadcast phase. An initiator robot begins the count as the root (referred to as *Host*) robot and connects to nearby neighbouring robots. From the point of each robot other than the Host robot, there is a parent robot which initiates connection. Once a connection with a parent robot is established, the robot then acts a parent and connects to other robots in its vicinity as its children. The newly added children now broadcast and establish connection with further robots.

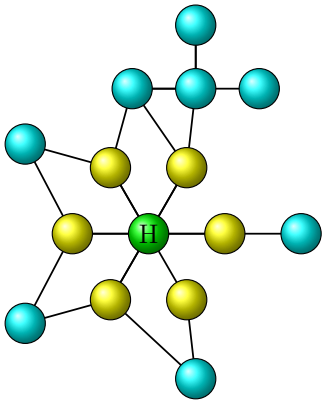

Figure 3.7: Broadcast Initiated by Host Robot

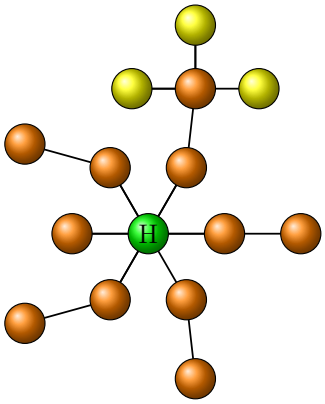

Figure 3.8: Connected Spanning Tree

Convergecast Phase

If a robot has no children, it begins the convergecast phase and sends a count value of *1* to the parent. The parent robot waits until the count value is received from all children. Once the count value is received, the parent robot sums up the count from all children, adds itself to the count and sends the new count value to its parent. This continues until the root robot receives count value from all its children.

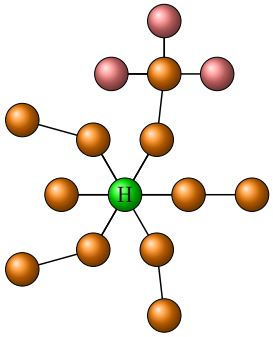

Figure 3.9: Convergecast Initiated by Nodes

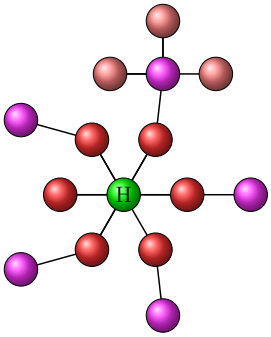

Figure 3.10: Convergecast last step

Count Propagation Phase

Once the root robot has received a count from all children, the total value of count is the sum received from all children plus the root robot itself. The root robot computes and propagates this count throughout the connected network and thus each robot in the network has information on the total count.

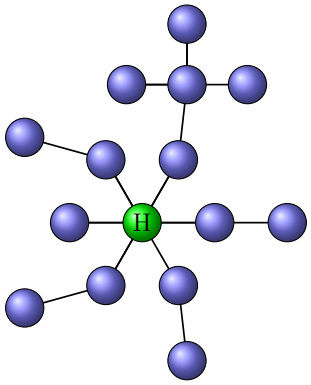

Figure 3.11: Count Propagated to all Nodes

Discussion

1. The advantages of using spanning trees in this application are:

   - The number of messages required for counting are $3n$ where n is the number of robots in a connected graph.
   - Each robot needs to only store IDs of Root, Parent and Children.
   - Time complexity depends on the topology of robots and in the worst case is$3nt$ where t is the time required to transmit and process each message. In the average case, it depends on the diameter of spatial distribution of robots.

2. Limitations when spanning trees are used for counting. They are as follows:

   - Breakdown of a robot during the three phases of count leads to disruption of count.
   - Since substantial time is required for constructing spanning tree, the robots are assumed to be remain static in space. Subsequent movement results in change of connected graph and disrupts the count.
   - If the range of communication is small, multiple spanning trees will be formed and invokes the need of merging count from different hosts.

3.1.6 Simplification

Difficulties in isolating task allocation

1. **Effect of Task Discovery:** Task discovery in space depends on the distribution of robots with respect to distribution of tasks and delay in task discovery results in dynamic addition of tasks to the list of existing tasks. This creates a temporal aspect to task allocation problem mimicking dynamic task requests.

2. **Effect of Robot Counting:** Counting the number of robots has the above mentioned limitations(3.1.5) . Despite ignoring any robot failure in simulation, the time required to count robots is non-deterministic and introduces delays.

3. **Effect of Recruitment:** Recruitment procedure induces an effect similar to task discovery, where task information is spread via local communication. This results in robots making decisions for allocation based on partially gathered task information.

Modifications

The focus of this work is on allocation of spatially distributed tasks, and therefore the modifications written below are made to the mentioned framework. These modifications enable isolation of the spatial allocation aspect and simplify some of the elements of a distributed system.

1. Spatial locations and robot requirement of each task is made readily available to the swarm via a Shared Table (3.2.1).
2. Counting of robots performing a particular task is performed via Loop Functions (Section 4.2.7).
3. Real-time update about the current requirement of robots for each task is also updated via a Shared Table (Section 3.2.1).

3.2 Design

3.2.1 Algorithm Structure

Shared Table

A Shared Table includes information about all tasks that is communicated to robots periodically. Task information is in the form of task location and task selection weights. Task selection weights are initially determined by the number of robots required by each task. The weights depends on two factors, the number of robots required by all tasks and the number of robots required by the task itself. A task that is under allocated is assigned positive weight while a task allocated with the exact number of robots has weight set to *0* and a task with more than necessary robots is assigned negative weight. Robots only select tasks with weights greater than 0.

Table 3.1: Shared Table

Task ID	Task 1	Task 2	Task 3	...	Task m	
Task Location	x_1, y_1	x_2, y_2	x_3, y_3	...	x_m, y_m	
Task Quota	C_1	C_2	C_3	...	C_m	

Thus robots allocate themselves to a task from the set of tasks with positive thresholds and perform task switching (see Section 3.2.2) if they are joining or performing a task with negative threshold. Table 3.1 shows an example of shared table. The quota values are updated when robots join or leave a task.

State Machine

We modelled the swarm behaviour with a state machine (see Fig 3.12. At any give time, a robot can execute only one state.

- ***NOTASK:*** Robot is neither performing any task nor joining any task. Every robot starts the experiment in a *NOTASK* state and if it allocates a task to itself, robot changes state to *JOININGTASK*.

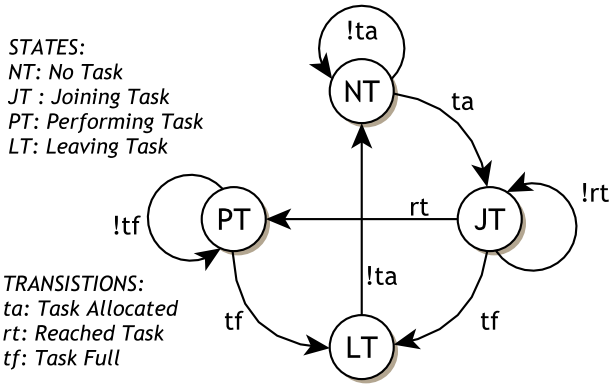

Figure 3.12: Robot State Machine

- ***JOININGTASK:*** A robot in *NOSTATE* allocates itself a task and changes its Execution State to *JOININGTASK*. The robot then proceeds to move to the task location.

- ***PERFORMINGTASK:*** A robot in *JOININGSTATE* changes its Execution State to *PERFORMINGTASK* upon reaching the task location. For simulation, the switch to *PERFORMINGSTATE* takes place when the distance of robot from task location is less than task radius.

- ***LEAVETASK:*** When the number of robots performing a task is equal or more than the required robots in task, a robot joining the task or performing the task can enter *LEAVETASK* state. In *LEAVETASK* state, the robot executes task switching where robots assign themselves to a new task. All robots in *JOININGTASK* state immediately switch tasks while the robots performing the task have a predefined probability to leave the task. This ensures that all robots performing a task do not switch tasks simultaneously.

3.2.2 Task Selection and Spatial Bias Formulation

At the initial stage, the number of robots required in each task is known and weights for each task are derived from them. When a robot selects a task (see Section 3.1.3), the probability of a task being chosen depends on the weights assigned to each task.

However, in order to improve the efficiency of the system, it is necessary to bias these weights with spatial information from task locations. The following subsection is dedicated to different formulations of biasing the weights.

The following terms are used in various spatial bias formulations.
$w_{i_{init}}$: Initial weight for $task_i$
$w_{i_{new}}$: Biased weight for $task_i$
d_{ij} : Distance of $task_i$ from $robot_j$
s_{ij} : Distance inverse($1/d_{ij}$) of $task_i$ from $robot_j$
μ_j : Average of distances to all tasks from $robot_j$
Σd_j : Sum of distances of all tasks from $robot_j$
$w_{i_{init}}$: Initial weight for $task_i$
β : Spatial Bias Factor
α : Task Size Factor

Each formulation is explained and compared with ten tasks. To understand the effect of the formulation on spatial information, the task size is kept equal for all tasks.

Table 3.2: Sample Comparison Set

Task ID	Task1	Task2	Task3	Task4	Task5	Task6	Task7	Task8	Task9	
Task Size	1	1	1	1	1	1	1	1	1	
Dist to Task	1	2	3	4	5	6	7	8	9	

It is important to note that the above sample set is for one robot with equally sized tasks. In scenarios with more robots, more tasks and various distribution of tasks in space, analysis of formulations do not remain trivial. For this reason extensive simulation experiments have been carried out.

Linear Formulation

This is the proposed primary formulation for spatial bias. The biasing function includes a tuning factor β which decides the sensitivity of the system. Increasing the sensitivity results in increased weights for nearby tasks and tasks further away get neglected once the weights for the tasks go below zero. Increasing sensitivity

also results in reduced impact from task size factor which is instrumental in deciding the initial weight.

$$w_{i_{new}} = w_{i_{init}} - \frac{\beta(d_{ij} - \mu_j)}{\Sigma d_j}$$

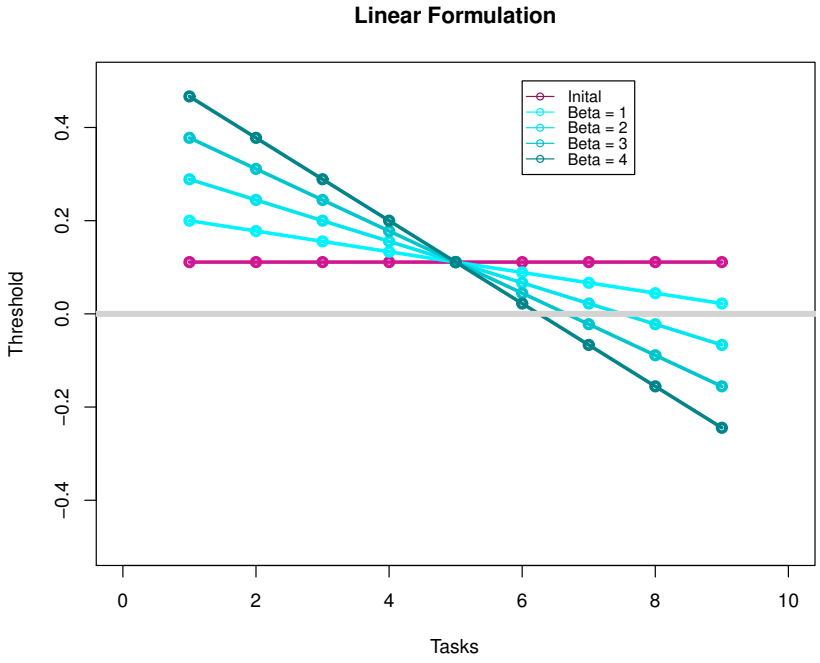

Figure 3.13: Linear Formulation

Exponent Formulation

Jevtić et al. (2012) proposed this formulation in the *Distributed Bees Algorithm*. The formulation takes two parameters into account, namely *quality* and *cost* of the task which is controlled by two factors,α and β. The two factors enable control over *qualities* and *cost* independently.

To show the behaviour of Jevtic's function in this case, *quality* of a task is assigned to task size and definition of cost is kept similar to the original algorithm($1/DistToTask_i$).

$$w_{i_{new}} = \frac{w_{i_{init}}^{\alpha} s_{ij}^{\beta}}{\sum_{i=1}^{n} w_{i_{init}}^{\alpha} s_{ij}^{\beta}}$$

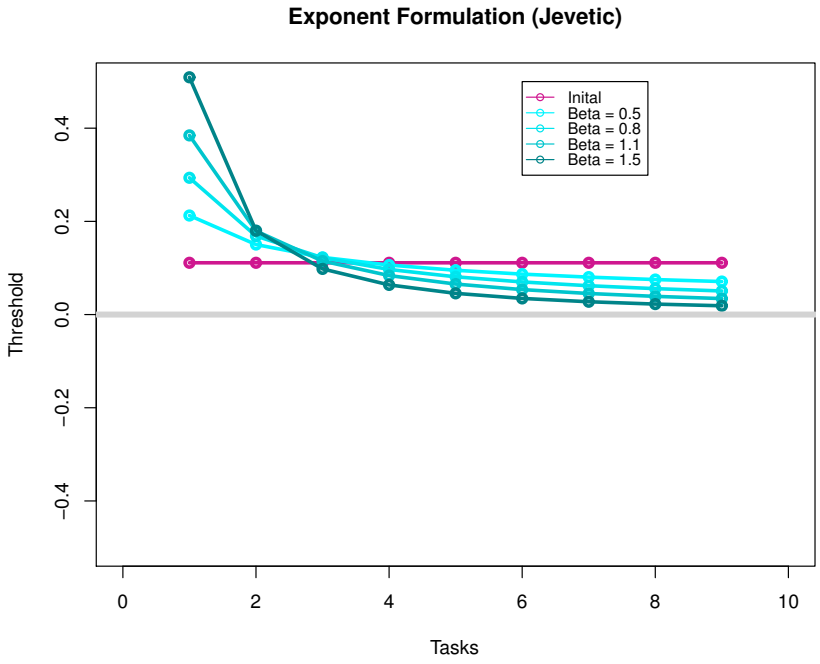

Figure 3.14: Exponent Formulation

Task Switching

Task switching is an integral part of a decentralized approach to task allocation when more than the required number of robots attempt to perform a task. Excessive robots at one or more tasks results in under-allocation of other tasks, waste of travelled distance and time taken to reach the task, and crowding effect. Thus it is beneficial for the system when excess robots, performing or joining a task, to allocate themselves to another task. It is more efficient for robots in *JOININGTASK* state to switch tasks than the robots in *PERFORMINGTASK* state.

Chapter 4

Evaluation

Chapter 3 discussed the framework and design of Spatial Bias Strategy. This chapter discusses the criteria for evaluation of the performance of biasing strategy (see Section 4.1). Section 4.2 lists the various changes made to experiment setup that are necessary to observe the behaviour of the system. Section 4.3 shows the compiled results from selected sets of experiments. Since a vast number of simulations were carried out, it is not possible to show result from every set of simulations. Therefore, results that aid in understanding the nature of the system under various parameters are taken into consideration.

4.1 Criteria for Evaluation

4.1.1 Robot Metrics

Distance Travelled

Reducing total distance travelled by the entire swarm is the primary criteria in robot metrics. In a physical world, reducing the total distance means using less energy to travel. This is critical as inexpensive robots usually have low battery life.

It is important to note that reduction of total travel distance does not imply that time required to allocate tasks also reduces. Consider a situation where a high number of robots travel very low distance and small number of robots travel large distances due to excessive task switching yet results in reduced total travel distance. However, the small number of robots take more time to get allocated. Therefore, distance travelled is mapped as box plots for each set of experiments.

Total Allocation Time

Reducing total task allocation time is equally important criteria as compared to reducing the distance travelled by the swarm. This criteria refers to the time required to completely allocate all tasks. In simulations, the experiment is said to be complete when all tasks are allocated. As the number of robots are set to be equal or greater than the total number of robots required, each experiment converges to total allocation state. In practical situations, priority is given to total allocation time or total distance travelled depending on the application.

Task Switches

This is a secondary criteria as compared to total distance travelled and total time required. When a robot is forced to switch allocation from one task to another, the energy spent in travelling to initial task is wasted. Thus limiting the number of task switches is important. However in situations where the number of robots required is more than required, this criteria can be relaxed and focus can be given to either or both of the above criteria.

4.2 Experiment Design

Section 4.1 focuses on an evaluation metric to judge task allocation strategy. It is necessary to carry out extensive simulation to understand the workings of this system with respect to different parameters. These parameters include spatial task topology, ratio of number of robots to number of tasks, and number of tasks in an arena. Additionally, varying these values over different bias factors for a large number of random seeds aids in understanding the system. Studying the behaviour of swarm system requires simulations involving large number of robots such as hundreds or even thousands. Section 4.2.7 describes the use of ARGoS simulator and the foot-bots used for simulations.

4.2.1 Spatial Task Distribution

Lattice

Spatial distribution of tasks follow different topologies depending on the application. Application in structured environments such as warehouses, construction, surveillance grids and agriculture fields have task locations placed in grid-like topology.

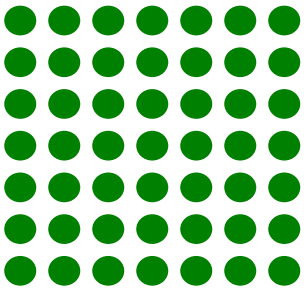

Figure 4.1: Lattice Distribution of Tasks

Uniform

Applications where task locations do not have relationship with other task locations tend to have uniform distribution. Applications such as pick-up and delivery requests in cities, show uniform distribution. Uniform distribution means that the probability of task being present at a position in space is equal for all positions.

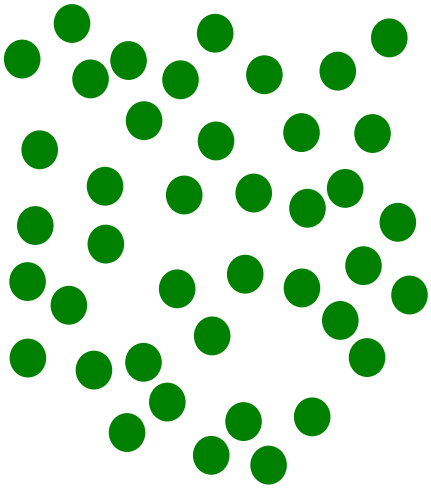

Figure 4.2: Uniform Distribution of Tasks

Scale-free

Scale-free networks show interesting behaviour which is seen in many applications. Scale-free distribution means that new task requests have higher probability of occurrence in the vicinity of existing tasks. This results in high density clusters of task and is seen in applications such as search and rescue, agriculture, and mining.

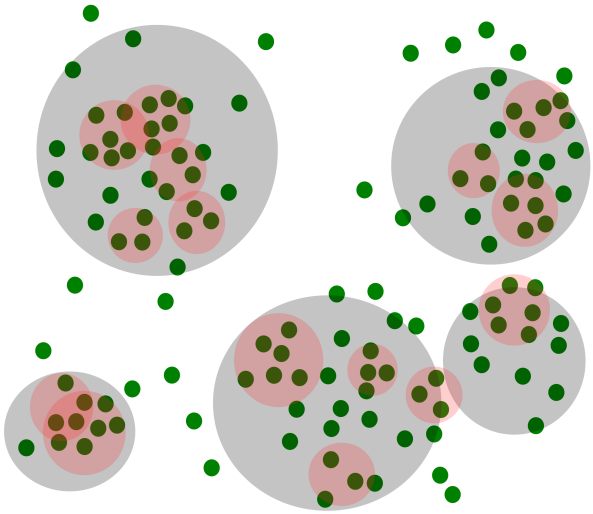

Figure 4.3: Scale-free Distribution of tasks

4.2.2 Task Size Distribution

Constant task size

In order to study the effect of task locations, task size is kept constant. This enables all tasks to have equal initial weights. The initial probabilities are altered by each robot depending on its relative position to each task. It is expected for the system to travel smaller total distances for increasing values of bias factor. Figure 4.4 shows an example when each task requires five robots.

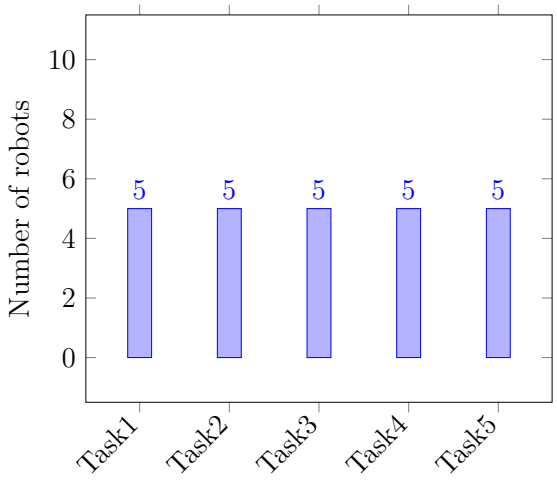

Figure 4.4: Constant Task Size

Uniformly distributed task size

Selecting task based only on distance results in ignoring the number of robots required by a task. The problem of efficiently distributing tasks is studied along with varying task sizes. Such a setting is important to study how sensitivity of the system and the effect on time required for complete allocation. The range for each experiment conducted with varying task size (0 - 2.5*mean$_{robots/task}$) i.e., if mean$_{robots/task}$ is set to 10, the range for task size is set between (0-25).

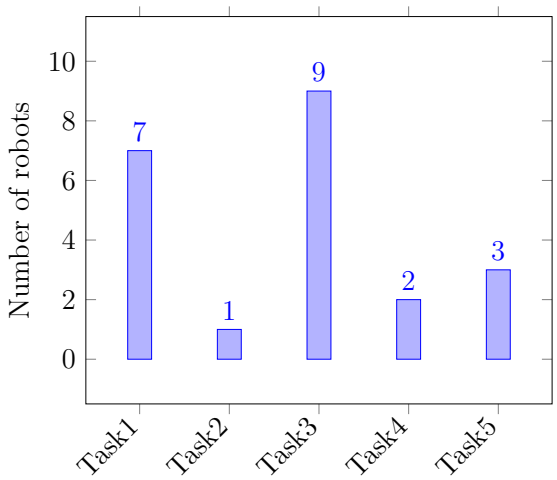

Figure 4.5: Varying Task Size

4.2.3 Arena Size

To confirm the scalability of the system, it is important to test the system by increasing the number of tasks. Arena size is increased in proportion to the number of tasks such that ratio of tasks to area is kept constant. Thus increasing arena sizes have increasing number of tasks. Increase in number of tasks directly corresponds to increase in density of clusters in a scale-free topology; a situation that can possibly lead to crowding of robots.

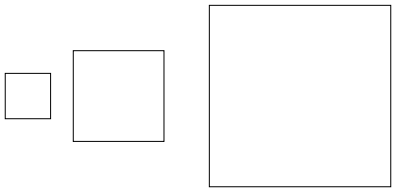

Figure 4.6: Increasing Arena Size: (a) 16m x 16m; 9 Tasks (b) 32m x 32m; 50 Tasks (c) 56m x 56m; 170 Tasks

4.2.4 Mean Robots per Task

Robots per task is an equally important parameter that is varied. This enables to study the scalability of system in terms of number of robots, along with the effect on time required to allocate tasks when number of robots per task increase. The number of robots per tasks is varied from 1,5,10,15,20. Figure 4.7 shows the three cases with 1, 10, 20 robots per task. The design of the system is such that it can handle both ST-SR-IA and ST-MR-IA cases of Gerkey's taxonomy (see Section 2.1.1) without any special reservations for either categories.

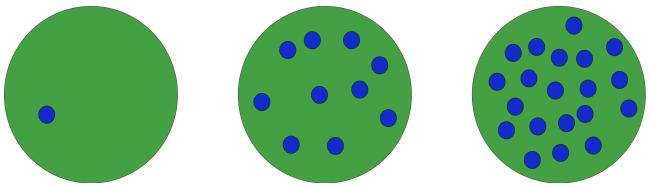

Figure 4.7: Robots per Task. From right: 1, 10, 20 robots per task

4.2.5 Numerical Parameters

Three numerical parameters are varied for various sets of experiments.

1. **Bias Factor β :** The value of Bias Factor is varied from 0 to 5.0 with steps of 0.25. Increasing the value of Bias Factor makes the robots more greedy to tasks that are closer and over power the task quota factor.

2. **Random seed :** Since the strategy depends on random number selection for task allocation decisions, each experiment setting is carried out for 50 different values of random seed. This is important because task size and task locations in uniform and scale-free settings are also randomized.

4.2.6 Special Cases

Robot Redundancy Factor

Performance of the system is measured when redundant robots are added to the system. Redundancy factor of 1.2 implies that if the system needed 100 total robots, 120 robots were given to the system. Redundancy factor of 1.2 and 1.5 is used in simulations.

Jevtic's Formulation

A set of experiments is conducted by using the formulation provided by Jevtic (see Section 3.2.2). Although the intention of Jevtic in (Jevtić et al., 2012) is to measure the accuracy of coalition formed; we study the the performance of the formulation with respect to the mentioned evaluation metrics (see Section 4.1).

PointMass3D Physics Engine

A large of experiments have been perform using the 'Dynamics2D' physics engine. The Dynamics2D physics engine takes into account collisions with physical geometries of robots while the PointMass3D engine treats robots as point-masses and allows robots to pass through one another without collision. Thus, the PointMass3D physics engine aids in studying the crowding effect cause when the density of robots and tasks in an area increases. Such a crowding effect makes it hard for other robots to find a way through robots engaged in other tasks.

Circle topology

This scenario is used as a corner case scenario and used as a means to validate the working of the proposed strategy. Tasks are spread evenly on the circumference of a circle and robots are densely clustered near the center of the circle (see Fig 4.8). Such a situation is expected to negate the distance factor as deviation of distance of all tasks from any robot is very less.

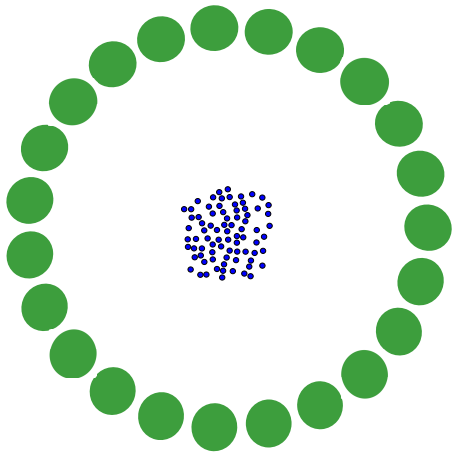

Figure 4.8: Circle topology with robots clustered at the centre.

4.2.7 ARGoS - Autonomous Robots Go Swarming

ARGoS (Pinciroli et al., 2012) is a multi-robot simulator and was built for the Swarmanoid Project (Dorigo et al., 2013) at IRIDIA Lab at the Universit Libre de Bruxelles, Belgium. ARGoS is highly flexible and efficient as it offers modularity in terms of design, parallelism in execution, and composability of objects. ARGoS is the most efficient tool to simulate the physics of thousands or even tens of thousands of robots.

Simulator

ARGoS facilitates use of multiple physics engines. ARGoS can divide the arena in regions and use a different dedicated physics engine for each region. This improves the flexibility and efficiency of the simulator and also allows robots to switch physics engines based on the position in the simulation space. ARGoS supports 2D kinematics and dynamics engines, a 3D particle engine and a 3D-dynamics engine.

The ARGoS simulator is highly modular. It facilitates the addition of different robots, physics engines,entities, visualizations, and communication media. This modularity makes ARGoS highly flexible. ARGoS simulator supports visualizations using a combination of Qt5 and OpenGL, ray tracing using POV-Ray, and text-based visualization.

ARGoS is a multi-threaded simulator that maximises performance and improves speed of execution on modern CPUs. ARGoS executes an experiment in multiple simulation steps. Each simulation step is divided into three main phases: *sense+control*, *act*, and *physics*. All three phases share resources such as sensors, actuators, and entity information. ARGoS prevents race conditions between the phases by executing *sense+control* first, followed by *act* and finally *physics* and also prevents multiple components linking to a same resource. For example, an actuator is linked to a component on a specific robot and cannot be shared by two robots.

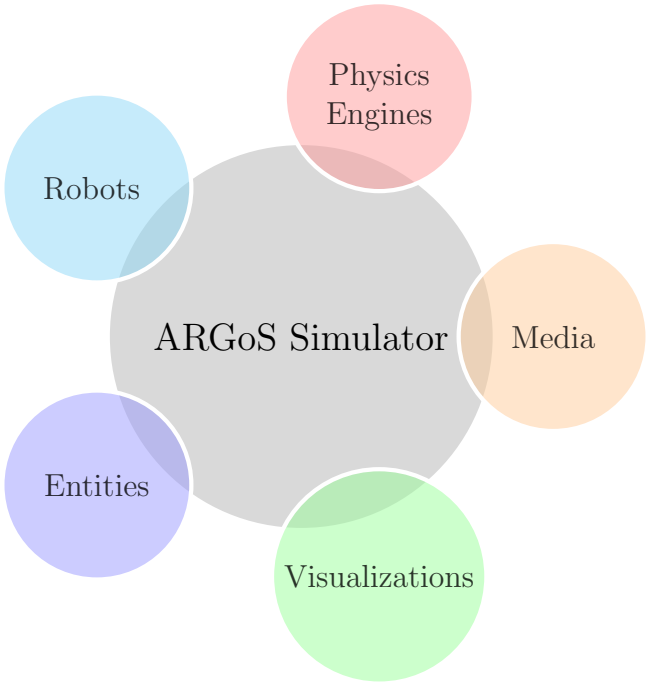

Figure 4.9: ARGoS Simulator

Foot-bot

The foot-bot robot was built at cole Polytechnique Fdrale de Lausanne, Switzerland , for the Swarmanoid Project (Dorigo et al., 2013) . The foot-bot is a differential drive robot (maximum speed 30 cm/s) with a circular chassis of diameter 13cm and 28cm high. The foot-bot has mechanical modularity and can attach itself to other foot-bots or hand-bots with the aid of a docking ring and gripper. The foot-bot contains 24 IR sensors along the circular chassis which act as proximity sensors and additional 8 IR sensors to the base of the foot-bot which perceive reflected shades of grey. The foot-bot is also equipped with 13 LEDs, 12 on the circular ring and one on top. The foot-bot is also equipped with two cameras; one top-front camera and an omnidirectional camera. It contains two distance sensors; one for near range(40-300 mm) and a long range sensor (200 - 1500 mm). Communication between foot-bots is achieved using the Range and Bearing system.

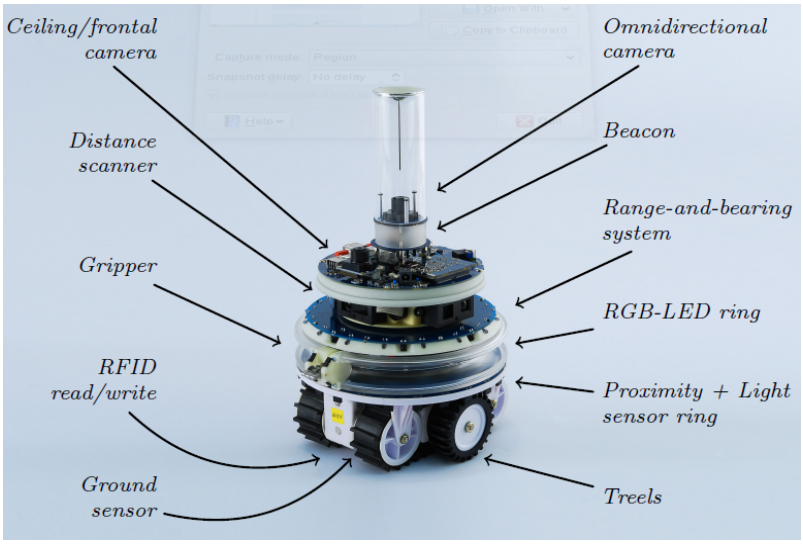

Figure 4.10: Foot-bot *(Source: IRIDIA)*

The foot-bot is an ideal robot for swarm applications due the amount of sensing capability, mechanical modularity, interaction with other robots and environment, small size, and ability to hot-swap the battery. Thus the foot-bot can be used to mimic recharging robots, self-assembling robots, autonomous and decentralized swarm experiments. In this work proximity, range and bearing, positioning, LED, and ground sensors/ actuators are used.

Controller

A controller is a plug-in that controls the behaviour of robots. ARGoS aids in developing user code, which is directly usable on real robots by making the controller access a *Control Interface*. *Control Interface* enables users to access sensors and actuators in a manner similar to that of real robots. Controllers for robots are written in C++, Lua, and Buzz (Pinciroli et al., 2015) .

Inside the ARGoS configuration file (*.argos*), the controller section allows addition of multiple controllers to an experiment.

Loop Functions

Loop functions allow users to modify the simulation. Loop functions act as *hooks* in an experiment and aid in initialization as well as determining the end of an experiment. Loop functions enable users to capture robot and environment data after every time-step and offers hooks for analysis after the experiment. Loop functions also provide functions for adding, removing, altering parameters or modifying states of entities. Entities in this case are robots, and objects in the arena.

In this work, loop functions are used to initialize the experiment by placing tasks in various topologies. Once the experiment begins, the Shared Table (see Section 3.2.1) is communicated to all robots at the start of each control loop. The shared table is updated by counting the robots that perform and join different tasks or are idling. This provides robots with any updates in the requirement of robots by a task and enables the robots to take task switching and task selection decisions. Loop functions are also used to monitor and record task allocation data after every 100 time-steps(10 secs) and distance travelled by the robots. Data analysed from various experiments in seen in Section 4.3

4.3 Results

This section shows results for various set of experiments described in Section 4.2. Each experiment setting result is made up of two graphs. The first graph contains information on the distance travelled by robots in an experiment and average task switches for all robots. Each experiment is setting is run for 50 different random seeds. Thus the box plot data of distance travelled by robots consists of number of robots in the experiment setting times 50. As we want to observe emergent behaviour, it is important to understand the performance of the majority of the robots and hence we show box plot data of the travel data of robots. The majority of robots allocate tasks in the first task selection and do no switch tasks, and therefore average task switches are considered over median task switches. The second graph contains data about the time required to completely allocate all tasks.

4.3.1 Spatial Task Distribution

Lattice Task Topology

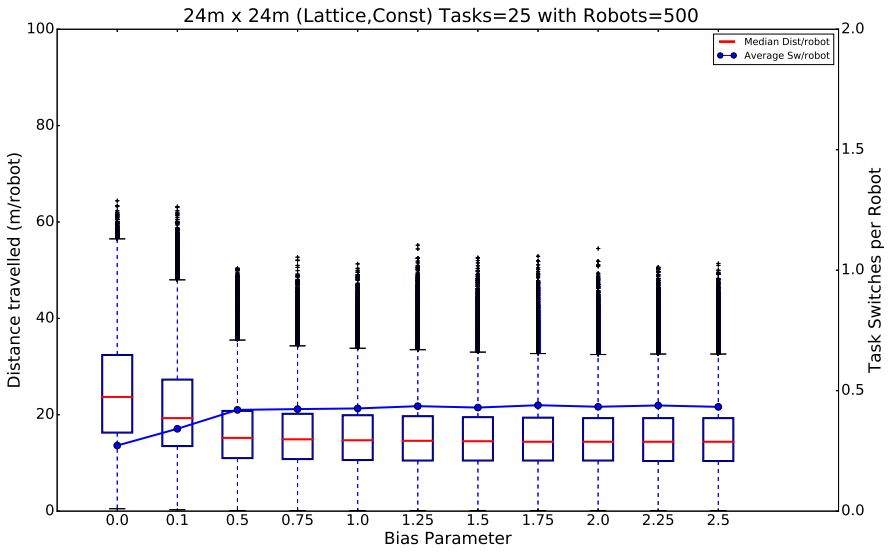

Figure 4.11: Lattice Task Topology (a) Distance and Task Switches

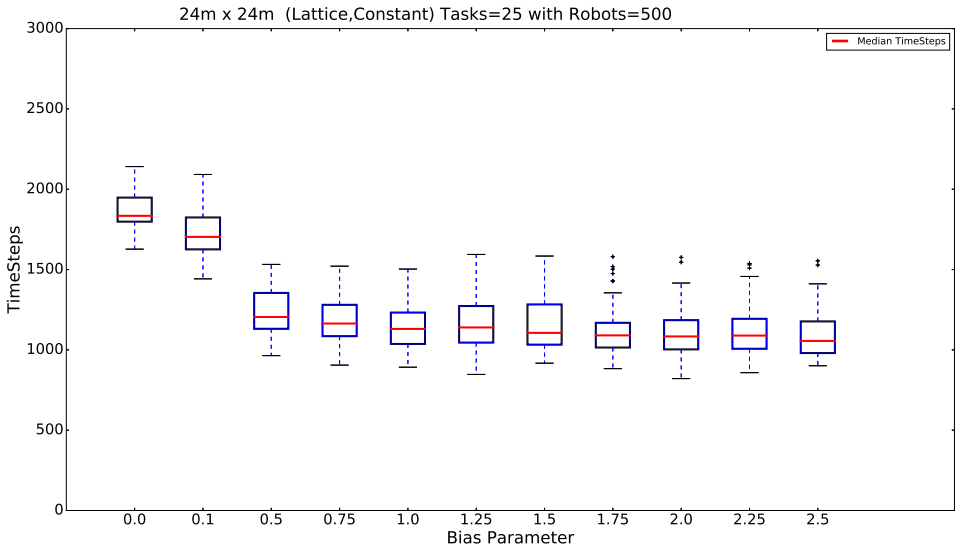

Figure 4.12: Lattice Task Topology (b) Allocation Time

Uniform Task Topology

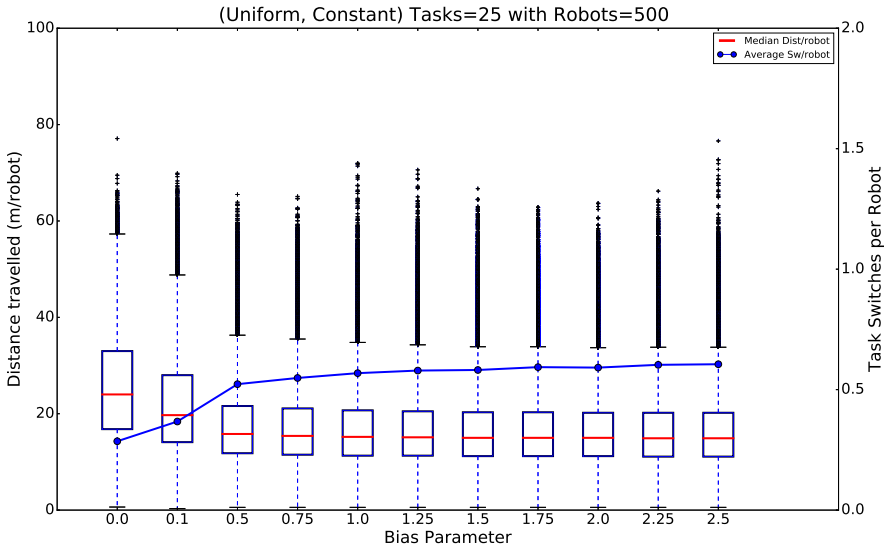

Figure 4.13: Uniform Task Topology (a) Distance and Task Switches

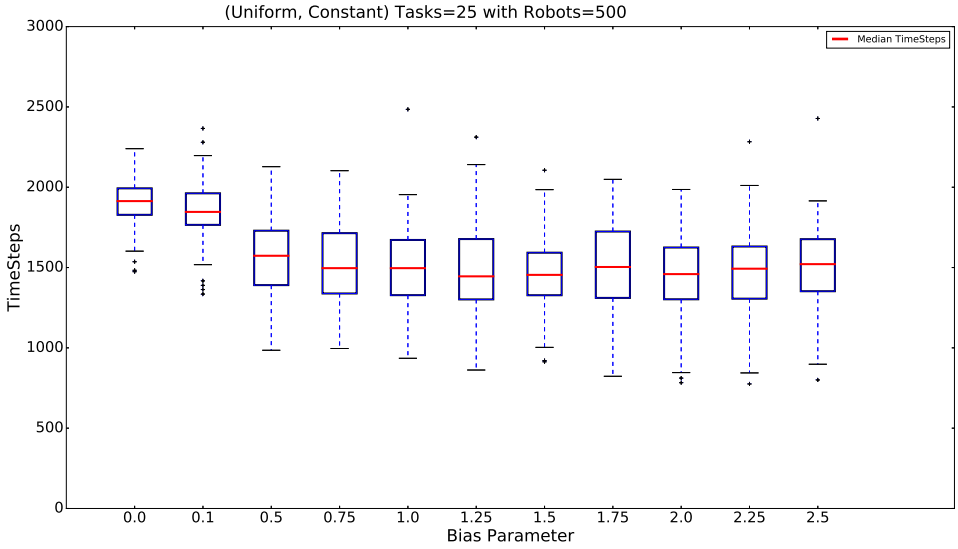

Figure 4.14: Uniform Task Topology (b) Allocation Time

Scale-free Task Topology

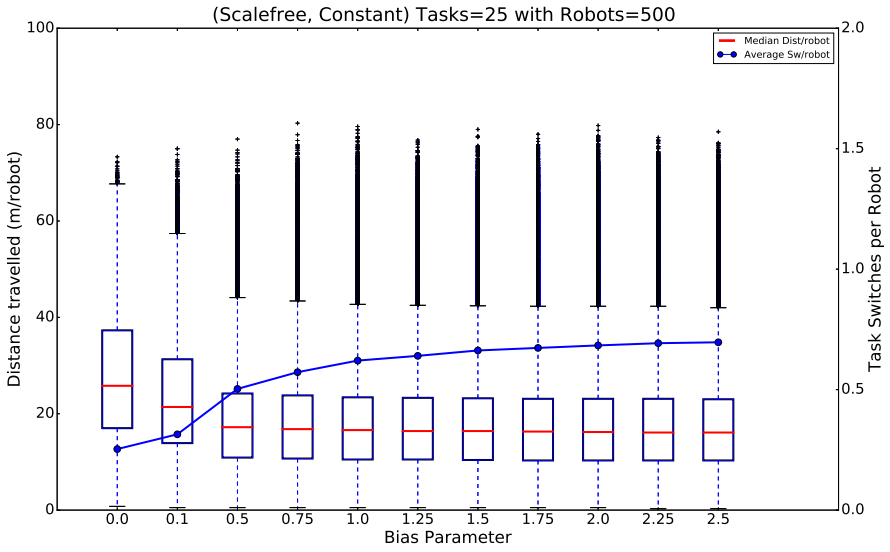

Figure 4.15: Scale-free Task Topology (a) Distance and Task Switches

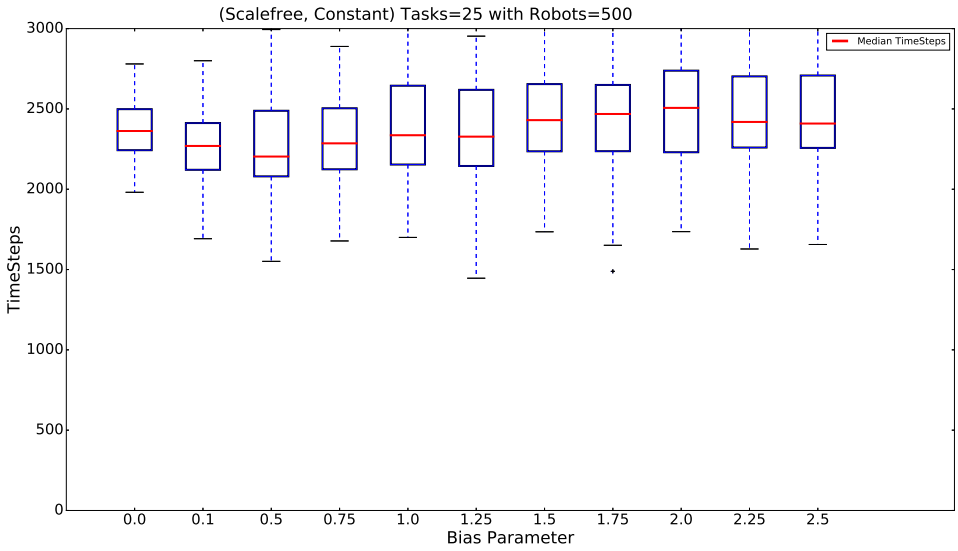

Figure 4.16: Scale-free Task Topology (b) Allocation Time

4.3.2 Task Size Distribution

A similar set of experiments with different topologies is carried out by randomising task size. However, since the trends and behaviour of the system is similar to that of the graphs in Section 4.3.1, the results have not been included in this report in interest of space. As a verification check, all the experiments in the following subsections include randomised task size distribution.

4.3.3 Arena Size

Small Arena (9 Tasks)

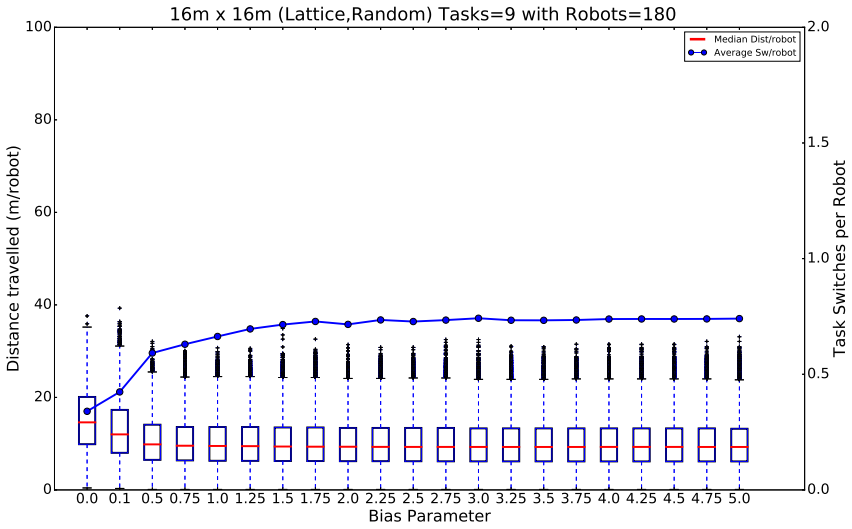

Figure 4.17: Small Arena 16m x 16m (a) Distance and Task Switches

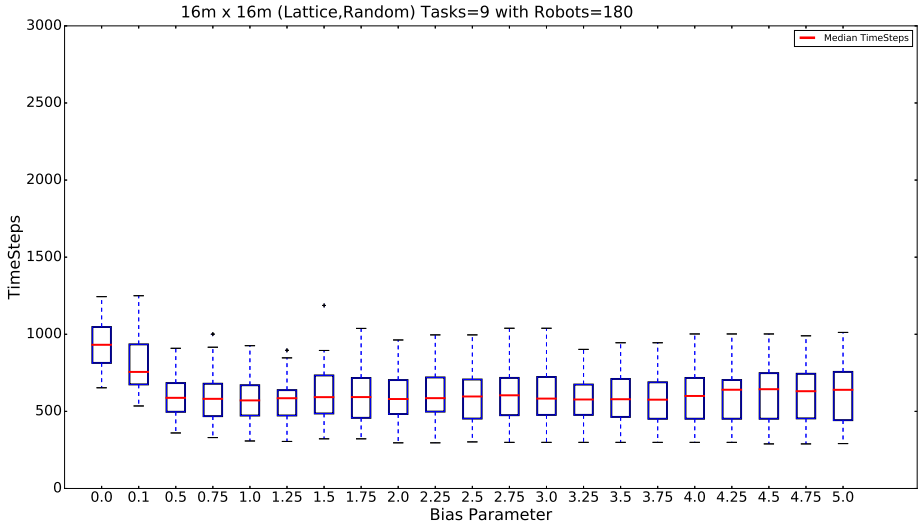

Figure 4.18: Small Arena 16m x 16m (b) Allocation Time

Medium Arena (50 Tasks)

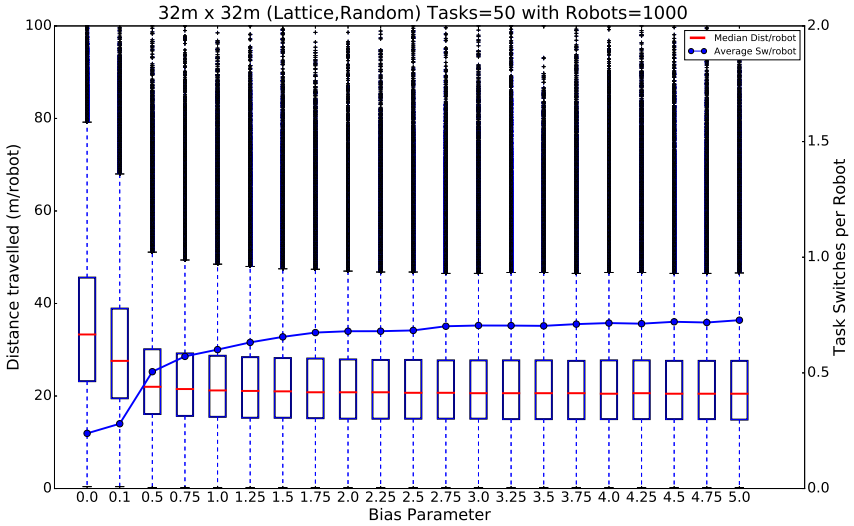

Figure 4.19: Medium Arena 32m x 32m (a) Distance and Task Switches

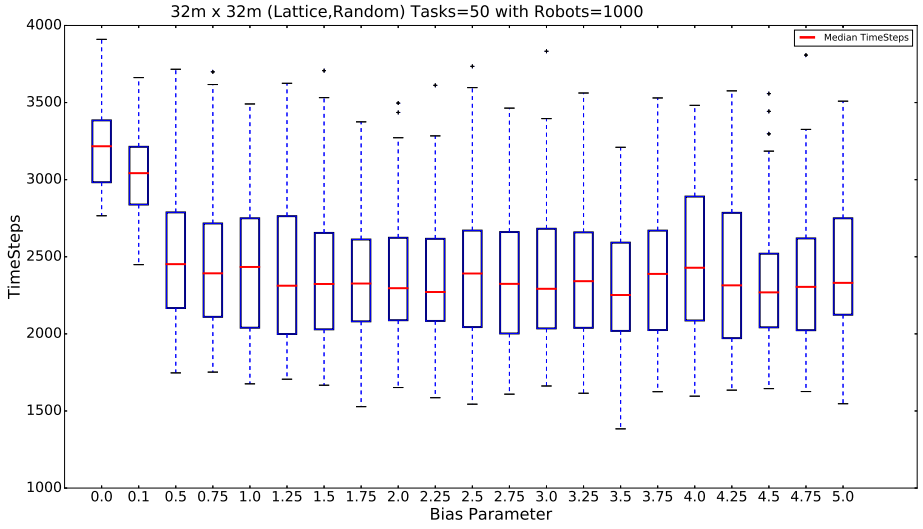

Figure 4.20: Medium Arena 32m x 32m (b) Allocation Time

Large Arena (170 Tasks)

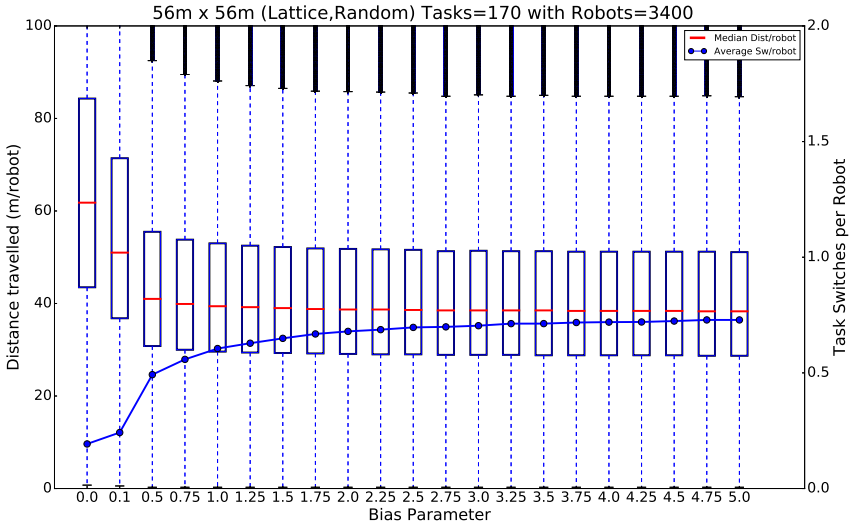

Figure 4.21: Large Arena 56m x 56m (a) Distance and Time

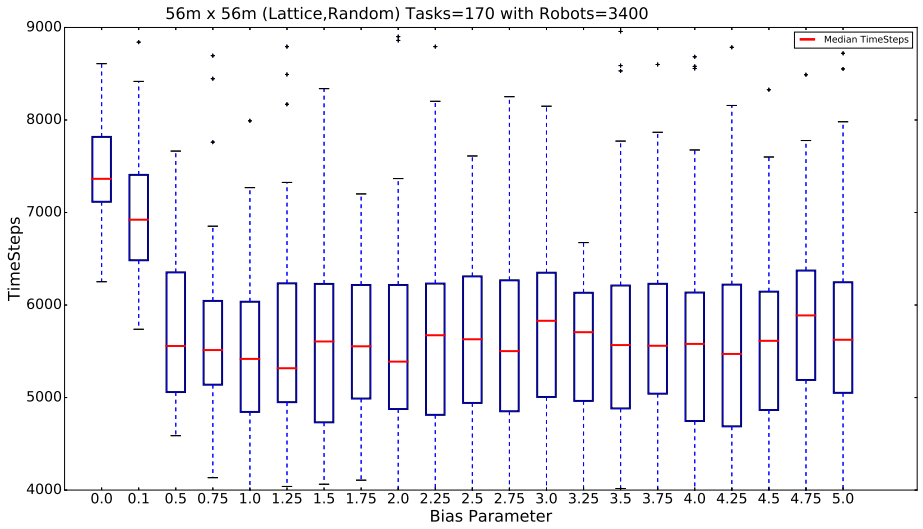

Figure 4.22: Large Arena 56m x 56m (b) Allocation Time

4.3.4 Mean robots per task

Mean Robot/Task = 1

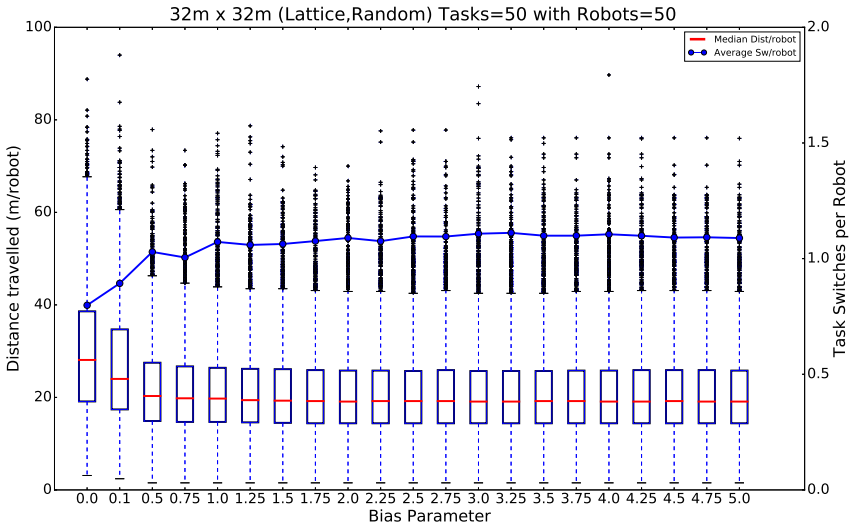

Figure 4.23: Mean Robots per Task = 1 (a) Distance and Task Switches

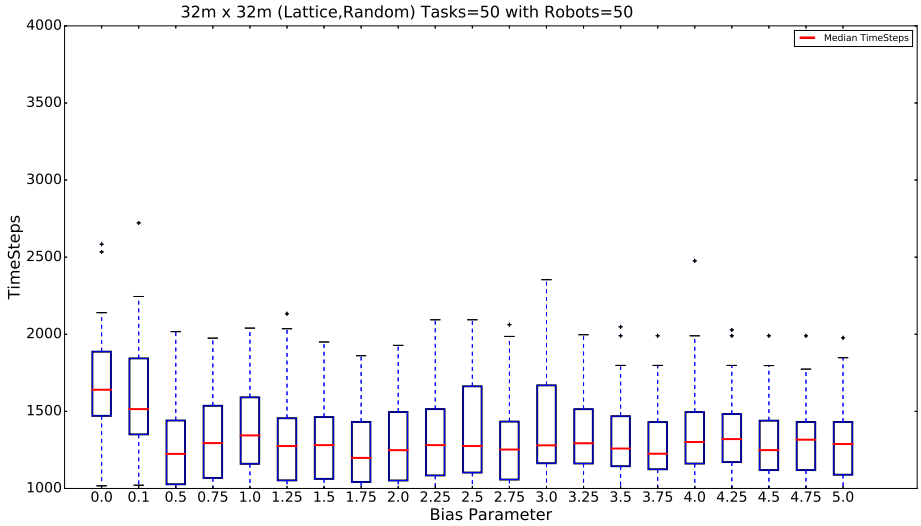

Figure 4.24: Mean Robots per Task = 1 (b) Allocation Time

Mean Robot/Task = 10

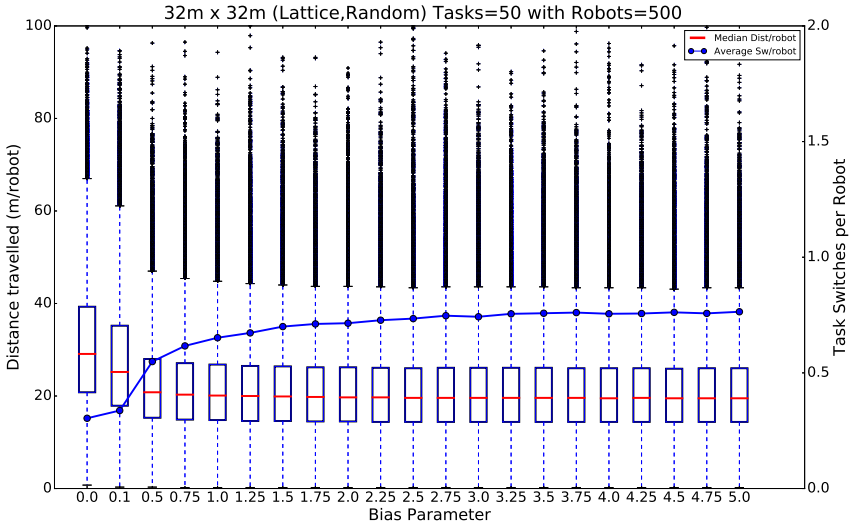

Figure 4.25: Mean Robots per Task = 10 (a) Distance and Task Switches

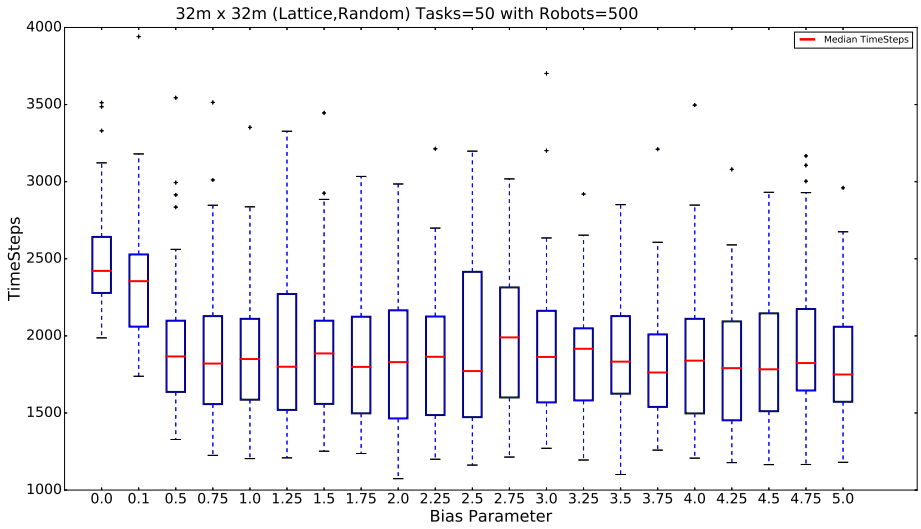

Figure 4.26: Mean Robots per Task = 10 (b) Allocation Time

Mean Robot/Task = 20

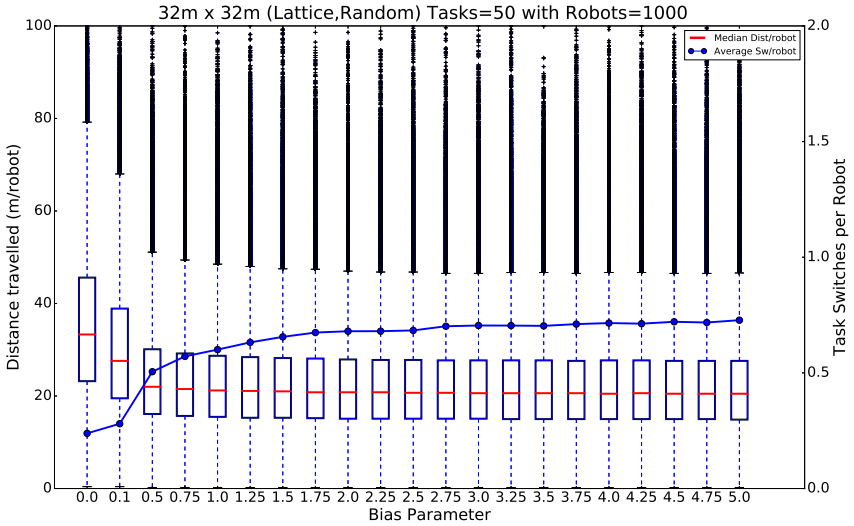

Figure 4.27: Mean Robots per Task = 20 (a) Distance and Task Switches

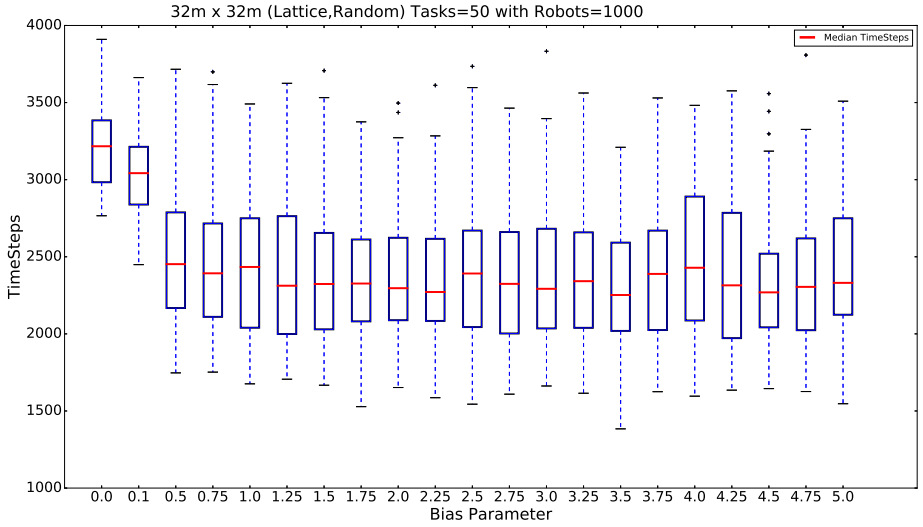

Figure 4.28: Mean Robots per Task = 20 (b) Allocation Time

4.3.5 Pointmass3d Engine

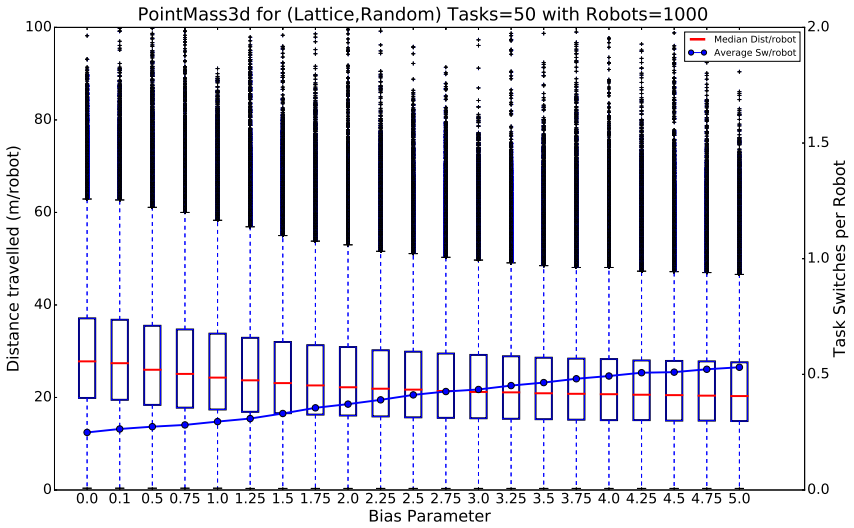

Figure 4.29: Pointmass3d Engine Mean Robots per Task = 20 (a) Distance and Task Switches

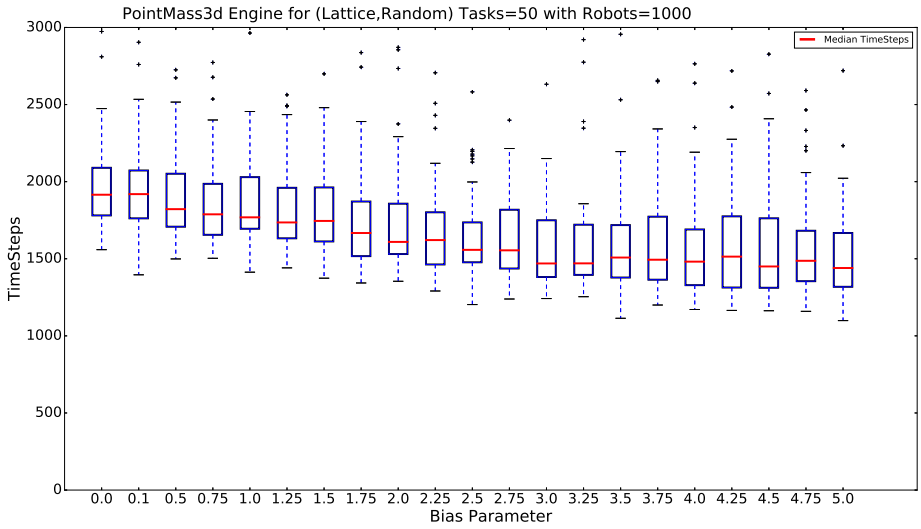

Figure 4.30: Pointmass3d Engine Mean Robots per Task = 20 (b) Allocation Time

4.3.6 Redundant Robots

Robot Redundancy = 1.2

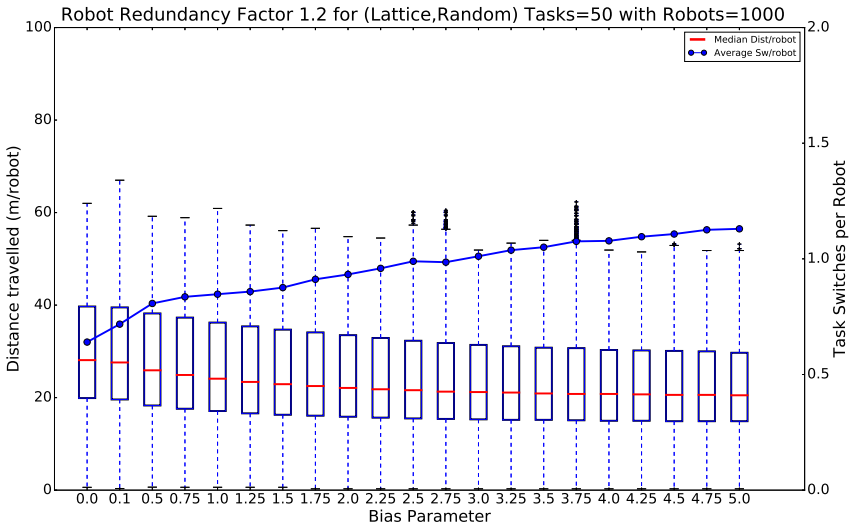

Figure 4.31: Redundancy Factor 1.2 (a) Distance and Task Switches

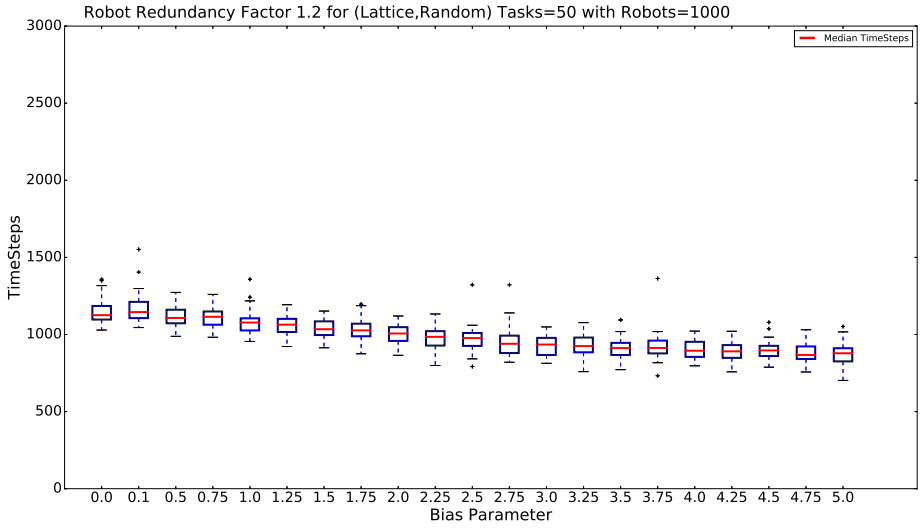

Figure 4.32: Redundancy Factor 1.2 (b) Allocation Time

Robot Redundancy = 1.5

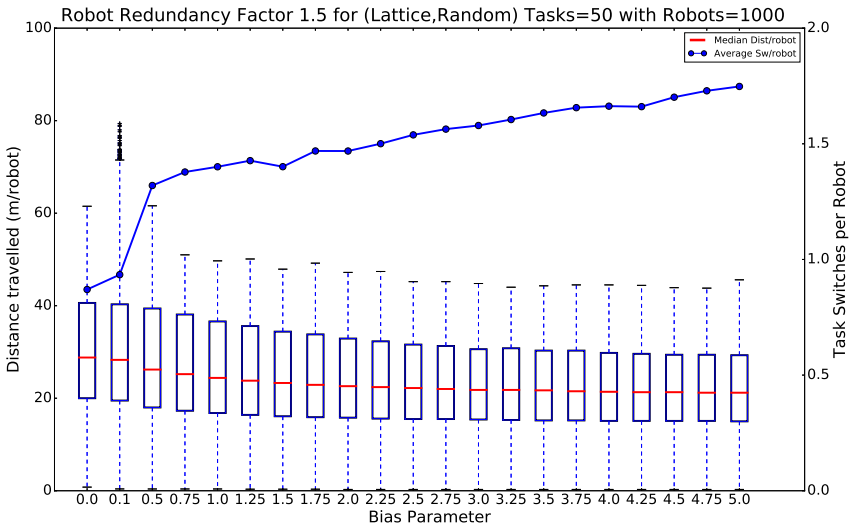

Figure 4.33: Redundancy Factor 1.5 (a) Distance and Task Switches

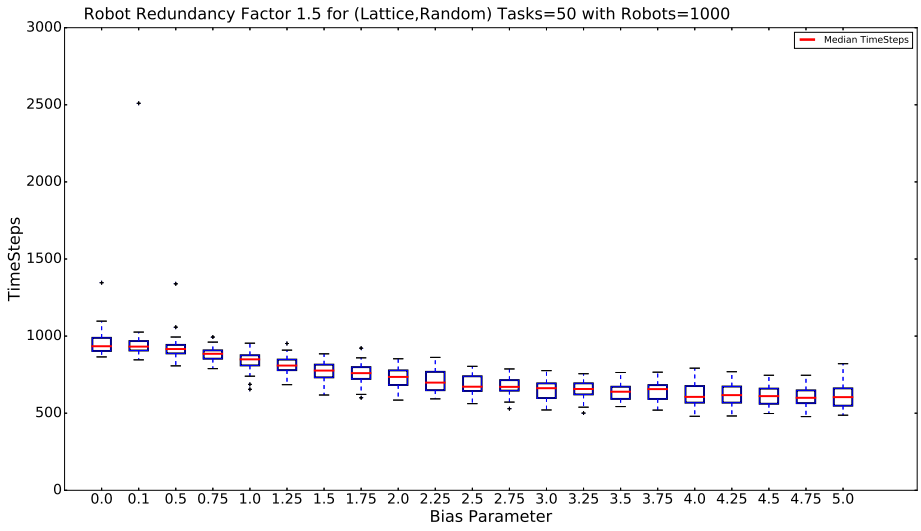

Figure 4.34: Redundancy Factor 1.5 (b) Allocation Time

4.3.7 Jevtic's Formulation

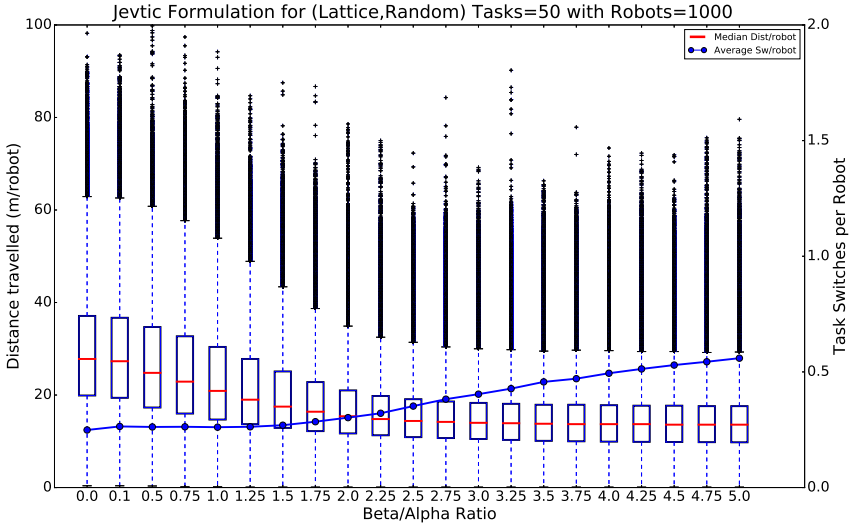

Figure 4.35: Jevtic Formulation (a) Distance and Switches

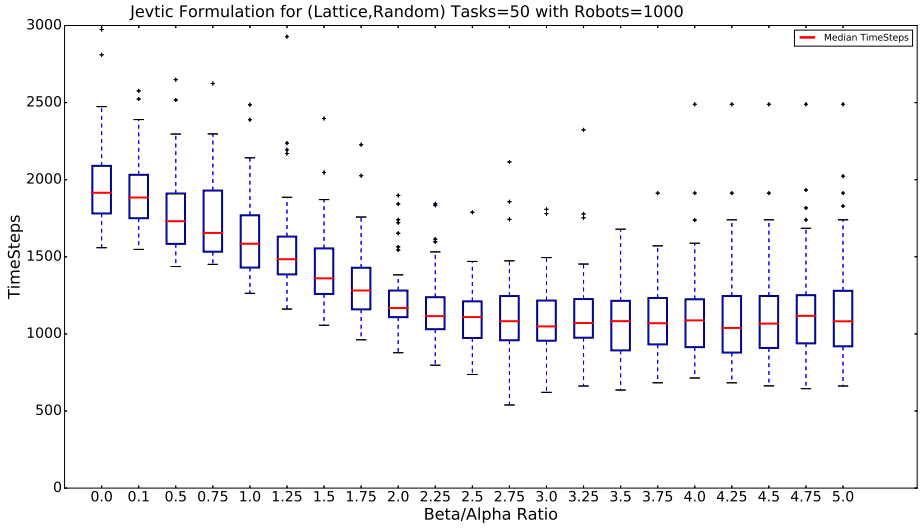

Figure 4.36: Jevtic Formulation (b) Allocation Time

4.3.8 Circle Topology

Circle Topology for Linear Formulation

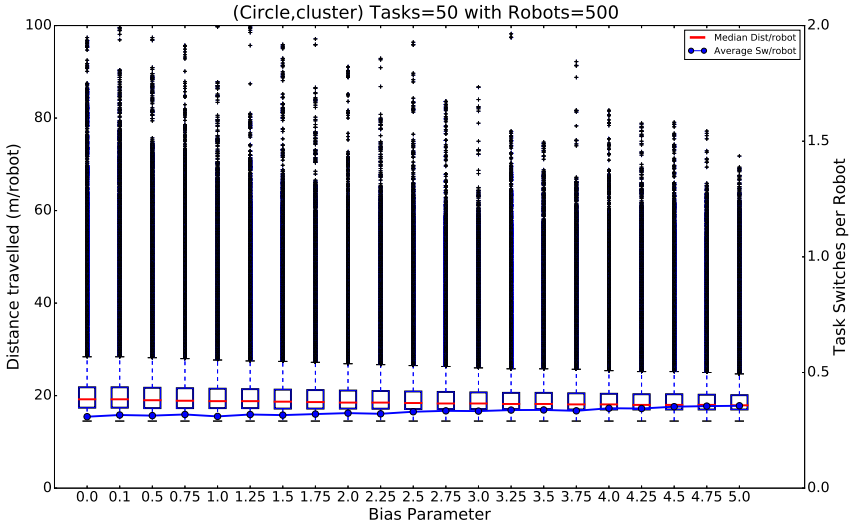

Figure 4.37: Circle Topology for Linear Formulation (a) Distance and Task Switches

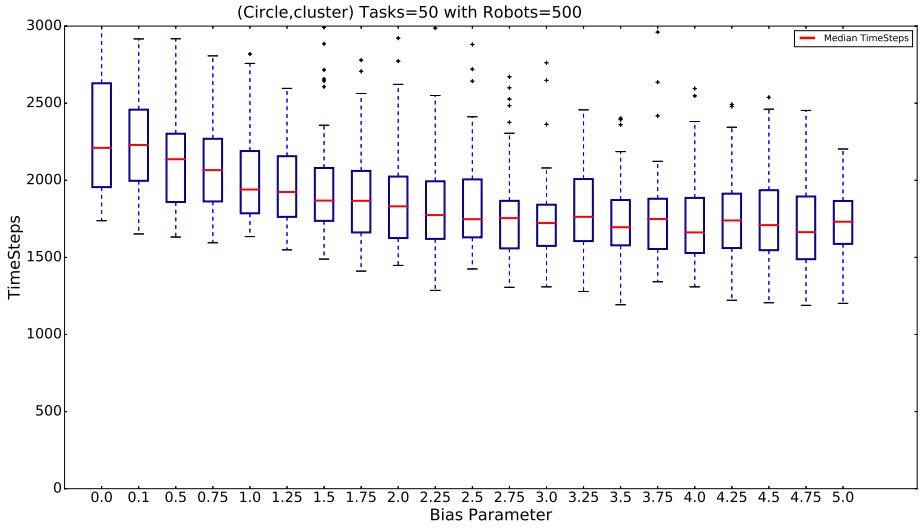

Figure 4.38: Circle Topology for Linear Formulation (b) Allocation Time

Circle Topology for Jevtic's Formulation

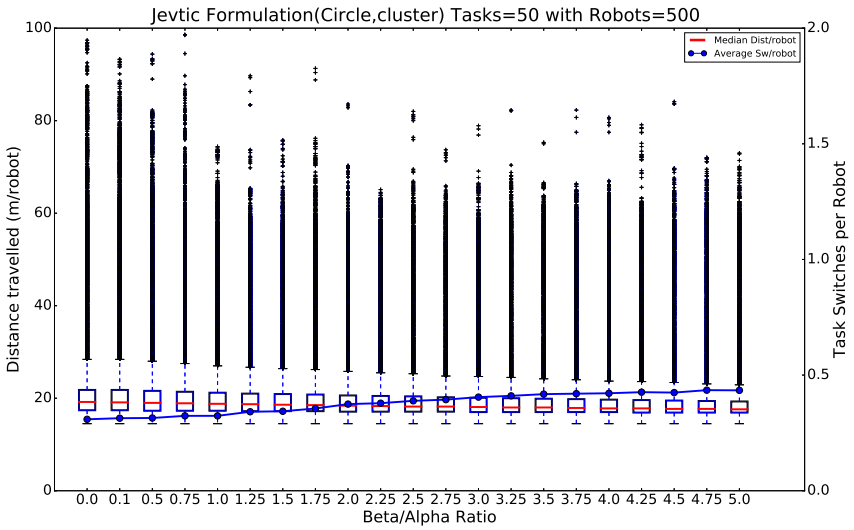

Figure 4.39: Circle Topology for Jevtic's Formulation (a) Distance and Task Switches

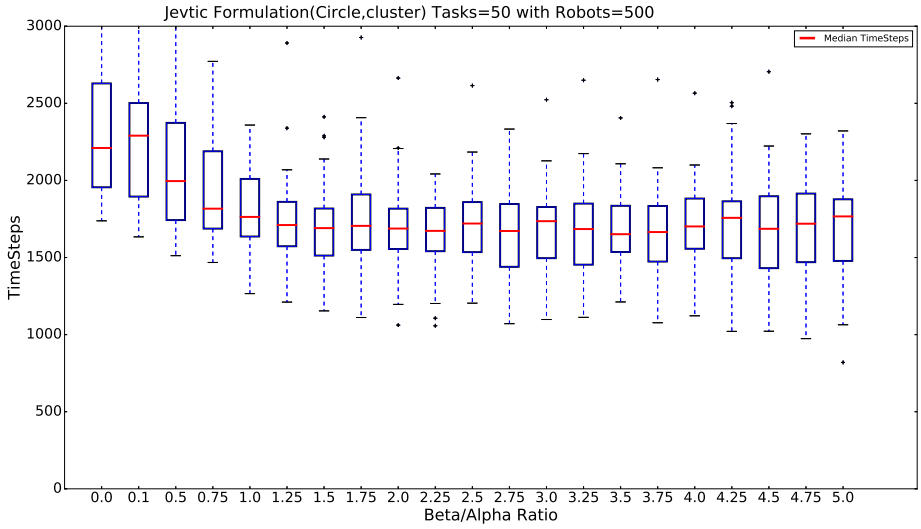

Figure 4.40: Circle Topology for Jevtic's Formulation (b) Allocation Time

4.4 Analysis of Results

Results for various sets of simulated experiments are given in Section 4.3. This section includes a summary of all the results in tabular form that aids in understanding various trends present in different sets of experiments

4.4.1 Task Topology

Lattice topology has tasks in a grid-like formation and the tasks are spread apart evenly. This does not result in crowding of tasks in the arena. However, the same is not true in the case of uniform and scale-free topologies.The time required for allocation for lattice topology converges to approximately 1100 timesteps while it increases for uniform distribution to 1500 timesteps and around 2300 timesteps. Task switching increased marginally from 0.45 for lattice to 0.6 for uniform and 0.65 for scale-free topology. The distance travelled by the robots remains approximately the same for all the topologies.

Table 4.1: Task Topology Results

Topology	Lattice	Uniform	Scale-free	
Distance	-	-	-	
Time	-	↑	↑↑	
Switching	-	↑	↑↑	

4.4.2 Arena Size Results

The increase in arena size also corresponds to an increase in the number of tasks as the tasks to area ratio is kept constant. This naturally results in robots travelling larger distances, however, an interesting observation is made that the median distance travelled is 1/2 *times* diagonal length of the arena at higher values of Bias Factor. Analogously, an increase in number of tasks, arena size and robots meant that time required increased.

Table 4.2: Arena Size Results

Arena Size(Tasks)	16x16(1 0)	32x32(50)	56x56(170)	
Distance	-	↑	↑↑	
Time	-	↑	↑↑	
Switching	-	-	-	

4.4.3 Mean Robots per Task Results

In the set of experiments with randomized task quotas, we impose the constraint that both the total number of robots and the ratio between total number of robots and total number of tasks are kept constant. When the mean number of robots of robots required by tasks is low, a high amount of task switching is observed as tasks quickly get fully allocated and more robots end up switching. When the mean number of robots is set to 10 or 20, mean task switching is approximately 0.7 unlike the case when the mean robots per task is 1 which shows mean task switching to be near 1.2.

Table 4.3: Mean Robots per Task Results

Mean Robots per Task	1	10	20	
Distance	-	-	-	
Time	-	↑	↑↑	
Switching	↑↑	↑	-	

4.4.4 Pointmass3d Results

Table 4.4: PointMass3D Physics Engine Results

Comparison with PointMass3D and dynamics2d engine shows the difference between robots as point masses with no collisions and robots as physical entities with collision. Using the PointMass3D engine, collision check and obstacle avoidance is not performed. The difference between the performance is noticed when the number of robots is increased as collision and crowding increases. PointMass3D engine helps validate the crowding effect.

Mean Robots per Task	Dynamics2D	PointMass3D	
Distance	-	-	
Time	-	↓	
Switching	-	↓	

4.4.5 Robot Redundancy Results

Increasing the number of robots by a factor of 1.2 and 1.5 has a profound effect on the time required for allocation. For the lattice setup with 50 tasks and 1000 robots, the average completion time with no redundancy is approximately 2400 timesteps. Redundancy factors 1.2 and 1.5 reduces this time to 1200 and 900 timesteps respectively. Additionally redundant robots have an adverse effect on task switching as redundancy factors of 1.2 and 1.5 show average task switching to be 1.1 and 1.6 when bias factor is kept at the maximum of 5.0.

Table 4.5: Robot Redundancy Results

Redundancy Factor	1	1.2	1.5	
Distance	-	-	-	
Time	-	↓	↓↓	
Switching	-	↑	↑↑	

4.4.6 Jevtic Formulation Comparison

Jevtic's Formulation shows an improved performance as compared to linear formulation. Both time and distance travelled show marginal improvement using Jevtic's formulation. This suggests that having high sensitivity is beneficial. The set of experiments are conducted using the PointMass3D physics engine for Jevtic's formulation. The comparison is performed with PointMass3D physics engine for linear formulation.

Table 4.6: Jevtic Formulation Comparison

Formulation	Jevtic	Linear	
Distance	-	↑	
Time	-	↑	
Switching	-	-	

4.4.7 Circle Topology Results: Sanity Check

This experiment setting is a corner case situation where the effect of relative distances is eliminated as the robots are placed at the centre of the circle and the tasks are on the circumference. For such a situation, the system is expected to show minimal change in behaviour for different values of Bias Factor. Such a result is seen for both Linear and Jevtic's formulation, where the values for distance travelled, time required for allocation and task switching remain almost same and show least deviation when compared to all other sets of experiments.

Table 4.7: Circle Topology

Formulation	Linear	Jevtic	
Distance	-	-	
Time	-	-	
Switching	-	-	

Chapter 5

Concluding Remarks

Chapter 4 concentrated on evaluation of the Spatial Bias Allocation Strategy. Large set of simulated experiments were analysed to observe trends in behaviour of the system with respect to change in one or more parameters of experiment setup. This chapter derives conclusions based on results and also suggests future scope of this work.

5.1 Conclusions

In this work, we propose a decentralized framework and study spatial bias strategy for multi-robot task allocation. The framework shows the importance of the robots' ability to discover tasks, recruit other robots and perform internal counting for a decentralized approach. We discuss strategies for task discovery and robot recruitment and implement a spanning tree approach to perform internal counting. The work further focuses on spatial bias formulation, task selection and biasing weights for task selection. In order to study the task allocation approach in detail, task discovery, recruitment and counting is performed using a shared table.

Results show that distance travelled and time required for allocation reduces as the bias factor is increased. However this has an adverse effect on mean task switching. Scale-free topologies suffer from crowding effect when number of robots per task and number of tasks increase. Use of Pointmass3D engine for scale-free topology shows improved performance as collisions are turned off and validates the crowding effect. Robot switching is high when number of robots per task is low. Redundant robots aid in task allocation and significantly improve allocation time

at the cost of increased task switching. Jevtic's formulation shows the significance of increased sensitivity of linear formulation that concentrates on eliminating tasks as valid selection options.

The system shows a scalable solution with fast response time. Experiments with 170 tasks and 3400 robots completed in 6000 timesteps (equivalent to 10 mins) in an arena size of 56m x56m where robots travelled at the speed of 0.3m/sec. Although task assignment is sub-optimal, a simple strategy shows potential for a scalable solution capable of quick response time. Additionally it is worth noting that redundant robots improve the performance of the system in terms of allocation time and also make the system robust to robot failures. The rigorous set of experiments carried out show the necessary evaluation required for robotics swarms. The large number of experiments are vital to understand emergent behaviour of the system. Thus the work presents a robust, scalable, simple, and an efficient strategy in terms of response time to manage multi-robot task allocation along with a supportive framework for a completely decentralized approach.

5.2 Future work

The future work can be expanded upon improving the task allocation strategy and evaluating the performance of allocation strategy under various aspects of task allocation.Additionally future work also includes performing experiments with real robots and running simulations for the entire framework.

- **Multi-vote Selection:** Currently selection of task is performed with a single draw of random number. In order to ensure that tasks with higher weights are given more preference, a multi-vote strategy can be used where multiple random numbers are sampled and tasks are selected on multiple occasion. Final selection of task depends on the task that is selected most.

- **Iterative Biasing:** Tasks in this work had two parameters, location and quota. Iterative biasing is useful in situations where tasks have multiple parameters such as locations, quotas, deadlines, priorities, urgencies, and skill requirements. The linear bias method has the capacity to eliminate tasks as selection options and a similar iterative biasing on multiple parameters can be performed to further reduce the subset of tasks available for selection. Such a biasing method may further improve the performance.

- **Dynamic Biasing:** The aim of this method is to reduce task switches by making the robot dynamically increase bias factor in proportion to the number of task switched performed by the robot.

- **Future Scope in Task Allocation:** The temporal nature of tasks in not considered in this work. The performance of the system with dynamic addition and deletion of task requests along with task allocation deadlines is an important aspect of task allocation. Additionally the system can be explored with heterogeneous robots where the biasing strategy can include provisions for robot's ability to perform tasks.

Bibliography

Ak, O. and Akn, H. L. (2016). Effective Multi-Robot Spatial Task Allocation using Model Approximations. *ArXiv preprint*.

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, 34(3):209–219.

Brambilla, M. (2009). Group Size Estimation in Swarm Robotics. *Milano, Politecnico D I*.

Choi, H. L., Brunet, L., and How, J. P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926.

Claes, D., Robbel, P., Oliehoek, F. A., Tuyls, K., Hennes, D., and van der Hoek, W. (2015). Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. *Aamas*, pages 881–890.

Dantu, K., Kate, B., Waterman, J., Bailis, P., and Welsh, M. (2011). Programming micro-aerial vehicle swarms with karma. *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*, pages 121–134.

Di Paola, D., Gasparri, A., Naso, D., and Lewis, F. L. (2015). Decentralized dynamic task planning for heterogeneous robotic networks. *Autonomous Robots*, 38(1):31–48.

Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Förster, A., Gonzales, J. M., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes De Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stützle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2013). Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20(4):60–71.

Dorigo, M. and Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling\nsalesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Ducatelle, F., Caro, G. A. D., and Gambardella, L. M. (2009). Task allocation in robotic swarms : new methods and comparisons Task allocation in robotic swarms : new methods and comparisons. *Technical Report No. IDSIA-01-09*.

Gerkey, B. P. (2003). ON MULTI-ROBOT TASK ALLOCATION. *CRES Technical Report, CRES-03-012*, (April).

Gerkey, B. P. and Matarić, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768.

Habibi, G., Fekete, P., Kingston, Z., and Mclurkin, J. (2016). Distributed Object Characterization with Local Sensing by a Multi-Robot System. *DARS*, pages 1–14.

Hoffman, K. and Padberg, M. (2001). Set Covering, Packing and Partitioning Problems. *Encyclopedia of Optimization*, pages 174–178.

Jevtić, A., Gutierrez, Á., Andina, D., and Jamshidi, M. (2012). Distributed bees algorithm for task allocation in swarm of robots. *IEEE Systems Journal*, 6(2):296–304.

Kalra, N., Zlot, R., Dias, M. B., and Stentz, A. (2006). Market-Based Multirobot Coordination: A Comprehensive Survey and Analysis. *Robotics Institute Carnegie Mellon University Tech Rep CMURITR0516*, 94(April):48.

Khaluf, Y. and Rammig, F. (2013). Task Allocation Strategy for Time-Constrained Tasks in Robots Swarms. *European Conference on Artificial Life*, (2009):737–744.

Korsah, G. A., Stentz, A., and Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512.

Kuhn, H. W. (1955). The Hungarian Method for the ASsignment Problem.

Li, Y., Klingner, J., and Correll, N. (2017). Distributed Camouflage for Swarm Robotics and Smart Materials. *NA*.

Mailleux, A., Deneubourg, J.-l., and Detrain, C. (2000). How do ants assess food volume? *Animal behaviour*, 59(5):1061–1069.

McLurkin, J. and Yamins, D. (2005). Dynamic task assignment in robot swarms. *Proceedings of Robotics: Science and Systems*, pages 2007–214838.

Mottola, L., Moretta, M., Whitehouse, K., and Ghezzi, C. (2014). Team-level Programming of Drone Sensor Networks. *SenSys '14 Proceedings of the 12th ACM Conference on Embedded Network Sensor SystemsSystems*, pages 177–190.

Nunes, E., Manner, M., Mitiche, H., and Gini, M. (2016). A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, pages –.

Padberg, E. B. M. (1972). On the Set-Covering Problem Author (s): Egon Balas and Manfred W . Padberg Published by : INFORMS Stable URL : http://www.jstor.org/stable/169305 REFERENCES Linked references are available on JSTOR for this article : You may need to log in to JSTOR to a. 20(6):1152–1161.

Parker, J., Nunes, E., Godoy, J., and Gini, M. (2016). Exploiting Spatial Locality and Heterogeneity of Agents for Search and Rescue Teamwork*. *Journal of Field Robotics*, 33(7):877–900.

Pinciroli, C., Lee-Brown, A., and Beltrame, G. (2015). Buzz: An Extensible Programming Language for Self-Organizing Heterogeneous Robot Swarms. *arXiv:1507.05946*, page 12.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., and Birattari, M. (2011). Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, 5(3-4):283–304.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1):209–238.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1):165–200.

Vig, L. and Adams, J. A. (2006). Market-Based Multi-robot Coalition Formation. *Distributed Autonomous Robotic Systems 7*, 22(4):227–236.

Zhang, J., He, K., Zheng, X., and Zhou, J. (2006). An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem. *2010 International Conference on Artificial Intelligence and Computational Intelligence*, 1(Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC'06)).

Zlot, R. and Stentz, A. (2006). Market-Based Complex Task Allocation for Multirobot Teams. *Proceedings of the 24th US Army Science Conference - Transformational Science and Technology for the Current and Future Force*, pages 169–176.