ABSTRACT

Title of thesis: DEVELOPMENT OF DECENTRALIZED TASK ALLOCATION ALGORITHMS FOR MULTI-AGENT SYSTEMS WITH VERY LOW COMMUNICATION

Akshay V. Bapat
Master of Science, 2020

Thesis directed by: Professor Jeffrey W. Herrmann,
Department of Mechanical Engineering
and
Dr. Michael W. Otte,
Department of Aerospace Engineering

Existing decentralized task allocation algorithms perform poorly under very low communication. Although previous work has considered task allocation algorithms in the presence of imperfect communication, the case of very low communication has not yet been addressed.

In this thesis, we present two new algorithms: the Spatial Division Playbook Algorithm and the Travelling Salesman Playbook Algorithm, which cater to the cases when the instantaneous probability (p) of a successful message between agents satisfies $p \ll 0.01$. These algorithms work by assuming that communications may not happen, but then derive advantages whenever communications are successful.

We compare these algorithms experimentally with three state-of-the-art algorithms — ACBBA, DHBA and PIA — across multiple communication levels and multiple numbers of targets, based on three communication models: Bernoulli

model, Gilbert-Elliot model and Rayleigh Fading model. Our results show that the algorithms perform better than the other algorithms and reduce the time required to ensure all targets are visited.

DEVELOPMENT OF DECENTRALIZED MULTI-AGENT TASK ALLOCATION ALGORITHMS FOR MULTI-AGENT SYSTEMS WITH VERY LOW COMMUNICATION

by

Akshay V. Bapat

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2020

Advisory Committee:
Professor Jeffrey W. Herrmann, Chair
Dr. Michael W. Otte, Advisor
Dr. Nikhil Chopra

Dedication

I would like to dedicate this work to my mother, Varsha Bapat, who continues to shower her love and blessings, and to whom I owe everything in my life.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisors, Dr. Michael W. Otte and Dr. Jeffrey W. Herrmann, for providing me with this extremely interesting research topic and supporting me throughout the duration of my work. They have been extremely helpful and have continually provided me with useful suggestions, without which this thesis would not have been possible. It has been a genuine pleasure to work as a part of the research group advised by Dr. Herrmann, Dr. Otte, Dr. Azarm and Dr. Xu. I would also like to thank Dr. Nikhil Chopra for agreeing to serve as a member of my thesis committee.

I would also like to thank my research group colleagues — Estefany Carrillo, Sharan Nayak and Suyash Yeotikar — whose previous work on developing the simulation framework and communication models for running experiments on multi-agent task allocation algorithms has provided us with a strong foundation for conducting this research.

My family — my father, mother and sister — deserve a special acknowledgement for their constant support and faith in me, and for making me the person I am today. I would specially like to thank Aishwarya Patwardhan for her unmoving support and motivation, and for providing me with excellent schematic diagrams for the algorithms in my thesis.

I would also like to acknowledge the huge role that all my friends played during

my work. It is due to their friendship and support that I have been able to maintain balance in my life.

Lastly, I would like to acknowledge financial support from the Air Force Research Laboratory (AFRL), USA. This work was supported by the grant FA8750-18-2-0114.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Autonomous Multi-Agent Systems

Autonomous multi-agent systems have been proposed for use in a variety of applications, including: search and rescue [1], agriculture [2], surveillance [3] and firefighting [4]. The agents collaborate with each other to complete the required mission. *Distributive task allocation* involves agents communicating with each other and assigning tasks among themselves. Task allocation algorithms can be broadly divided into two categories, centralized and decentralized algorithms [5][6]. Centralized algorithms require the existence of a central authority that dictates actions to the agents, while decentralized algorithms require the agents to think and act by themselves, without any centralized control. Centralized algorithms are vulnerable to single points of failure, since the central authority has to communicate tasks to all of the agents. Decentralized algorithms are less susceptible to single points of failure, but may require duplication of computational effort by multiple agents.

In real world scenarios, communication is not necessarily perfect, and both centralized and decentralized algorithms can fail due to a lack of communication. For example: (1) when signal jamming prevents agents from communicating, (2) when the environment is so large that there is considerable loss in communication

signal, (3) when the mission requires the agents to operate silently, as a message can give away the location of an agent to an adversary, or (4) when hardware malfunction results in agents being unable to send or receive messages. Taking these cases into account, we attempt to solve the task allocation problem for very low communication scenarios.

1.2 Solution Approach

The main contribution of this thesis is a set of two novel "Playbook Algorithms" tailored to work better than existing decentralized task allocation algorithms when communication availability is very low. We also present experimental evidence showing that the proposed algorithms work better than three existing algorithms (ACBBA [10][11], DHBA [12] and PIA [21]) in a variety of scenarios.

Assuming n agents and m targets, the Playbook Algorithms leverage the idea that a deterministic procedure will produce the same result when run independently by each agent in the team. Therefore, we design two deterministic procedures, one for each algorithm.

The Spatial Division Playbook Algorithm (SDPbA) divides the environment into n different regions — one region per agent – such that each region contains either $\lfloor m/n \rfloor$ or $\lceil m/n \rceil$ tasks. The i-th agent then starts by doing the tasks in the i-th region.

The Traveling Salesman Playbook Algorithm (TSPbA) generates a traveling salesman approximate solution which is the same for each agent and exhaustively

divides it into n distinct pieces — one piece per agent — such that the variance of the piece lengths is minimum. In an ideal scenario, the variance would be *zero*, implying that each piece has the same length.

In either algorithm, to accommodate for cases when some agents may not complete their tasks (for example, if they may be destroyed) each agent continues to move, in order, through the remaining regions/pieces, to ensure that tasks in all regions are eventually completed. Agents periodically broadcast all tasks they know to have been completed. Thus, even a single successful communication can drastically reduce the work of the receiving agent.

We evaluate the Playbook Algorithms on scenarios of decentralized task allocation for target visits. Experiments are performed in simulation to compare the performance of the Playbook Algorithms to other state of the art methods across a variety of communication quality levels and target numbers m (while keeping agent number n constant). Communication is modelled in three different ways: Bernoulli model, Gilbert-Elliot model and Rayleigh Fading model. We focus on the cases when the instantaneous probability of a message being successfully delivered from one agent to another, given by p, satisfies $p \ll 0.01$.

Figure 1.1 illustrates the approach used by SDPbA. Here, the autonomous agents are not able to communicate with each other, but divide tasks (denoted by T_i) by region in space and allocate regions amongst themselves. Agents A_1, A_2 and A_3 allocate unique regions (R_1, R_2 and R_3) to themselves. In scenarios where agents may not finish tasks, then after an agent completes its own tasks, it continues through the other regions to ensure all tasks are completed. This behavior acts as

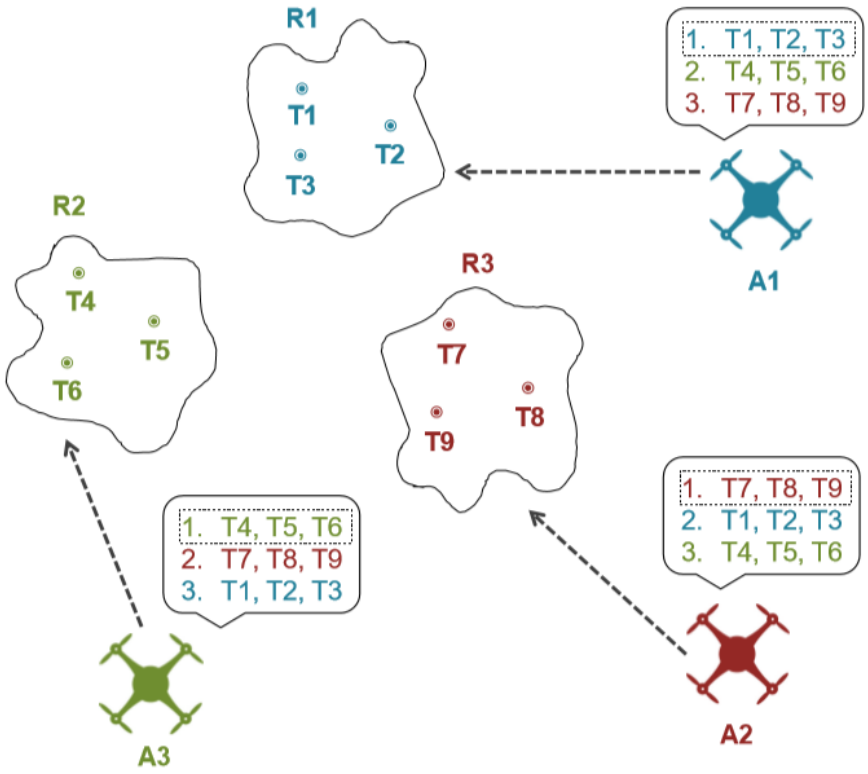

Figure 1.1: The approach used by Spatial Division Playbook Algorithm

a redundancy check in scenarios when agents may not be able to complete their assigned tasks. Successful communications (containing lists of completed tasks) improve performance, since agents can forgo visits to tasks known to be completed.

Figure 1.2 illustrates the approach used by TSPbA. Here, the autonomous agents compute an approximate solution to the travelling salesman problem (TSP) such that each agent has the exact same path. This solution is a Hamiltonian path through all the targets. The agents (A_1, A_2 and A_3) divide this computed path through all targets into pieces (P_1, P_2 and P_3) and exhaustively allocate these pieces to themselves such that each agent is allocated a unique piece. A dotted gray line indicates that edge of the TSP solution that is not assigned to any agent because targets that define that edge are assigned to one agent each. A dashed

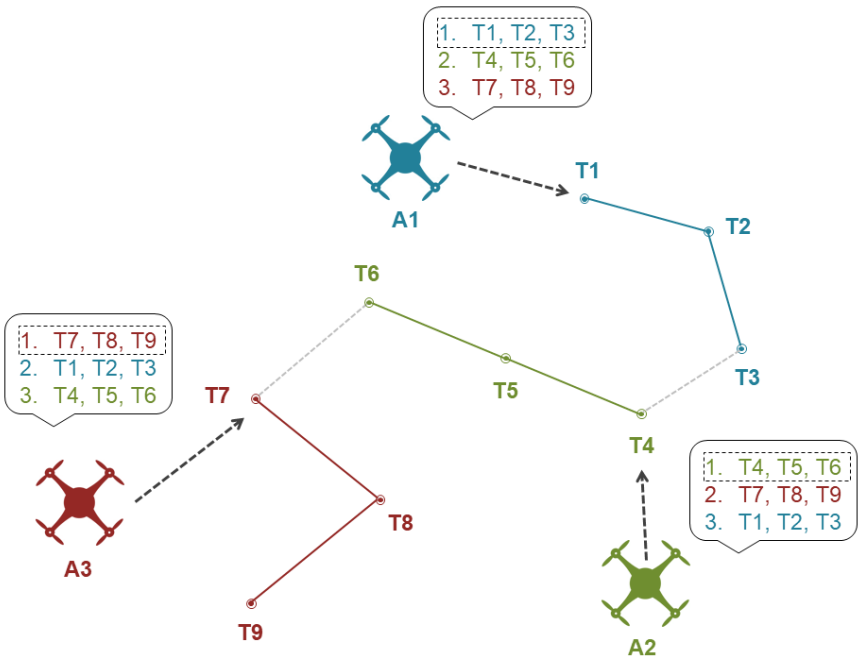

Figure 1.2: The approach used by Travelling Salesman Playbook Algorithm

black line indicates the path that an agent has to traverse in order to reach the first target of its assigned piece from its starting location. This path is not a part of the TSP solution and may be different for each agent, depending on the agent's starting location. Similar to SDPbA, an agent running TSPbA completes its own piece and then continues to other pieces to ensure all tasks are completed.

1.3 Outline of Thesis

Chapter 2 of this thesis outlines relevant work in this area of research. Chapter 3 defines relevant nomenclature, states the assumptions and formulates the problem statement that is addressed in this thesis. It then discusses the communication models used for simulating the experiments.

Chapter 4 proposes and describes the new playbook algorithms and Chapter

5 outlines the three existing task allocation algorithms (ACBBA, DHBA and PIA) that are used for comparison with the playbook algorithms.

Chapter 6 explains the experimental setup used to run simulations for comparing the five algorithms. This includes an overview of the simulation framework and a detailed explanation of the design of experiments.

Chapter 7 presents the results of the set of experiments run. Chapter 8 presents a mathematical proof that aids us in the analysis of the results and discusses the communication bandwidth requirements for all the algorithms. Chapter 9 discusses the implications of results obtained from the simulations and Chapter 10 derives conclusions based on these results, and also discusses possible future directions of work in this area.

Chapter 2: Related Work

Solutions to the multi-agent task allocation problem have been proposed in several papers. One commonly used approach is to conduct an auction in which each member of the team submits a competing bid, and the task is awarded to the agent with the best bid. The Consensus-Based Auction Algorithm (CBAA) and the Consensus-Based Bundling Algorithm (CBBA) [9] are widely used algorithms based on the auction approach. The Asynchronous Consensus-Based Bundling Algorithm (ACBBA) [10][11], Performance Impact Algorithm (PIA) [21] and the Hybrid Information and Plan Consensus algorithm (HIPC) [22] are improved algorithms based on the CBBA.

Another approach to the task allocation problem involves the use of deterministic or stochastic optimization techniques. The Decentralized Hungarian Based Algorithm (DHBA) [12] employs deterministic optimization using the Hungarian method, while the ant-colony optimization algorithm uses stochastic optimization techniques. Herrmann [25] discusses the use of experimental data about collaborative algorithms to develop a metareasoning policy that changes the algorithm that an agent is running based on the expected performance in that scenario. Nayak *et al.* [8] provide a detailed comparison of five decentralized task allocation algorithms

for a wide range of communication quality simulated using three different communication models, namely Bernoulli, Gilbert-Elliot and Rayleigh Fading models. This thesis builds on the work done by Nayak *et al.* by focusing on cases of very low quality of communication using the three aforementioned communication models.

The problem of imperfect communication in multi-agent task allocation has also been addressed in various papers in the literature. Otte *et al.* [13] evaluates three distributed auction-based algorithms in lossy networks and the effect of imperfect communication on task allocation in centralized multi-agent systems with a single auctioneer. Rantanen *et al.* [19] discuss the effect of a realistic communication model on the performance of ACBBA using an open-source network simulator. The results of [19] indicate that even when the communication is near-perfect, the algorithm becomes very inefficient due to redundant task assignments. Alighanbari *et al.* [23] propose a different approach to task allocation that introduces a second phase in which agents communicate with each other, which improves the algorithm performance in sparse networks. Sujit *et al.* [20] propose a team theory to overcome inefficient performance of greedy algorithms when sensor range is limited.

Previous work has investigated how centralized [13] and decentralized [8] multi-robot task allocation algorithms degrade as a function of decreasing communication quality. However, this previous work has taken the approach of starting with an algorithm designed for perfect communication, and then observing how well that algorithm performs as communication degrades. In contrast, we design multi-agent task algorithms that work when there is *no* communication between agents in the field, and then study how its performance improves with increasing communication

quality.

Each agent in a team has a different perception of the world and, as a result, possesses unique information regarding the status of tasks. Agents share this information with each other to collectively complete the mission more effectively. When communication availability is very low, *i.e.* a large majority of the messages get dropped. The task allocation algorithms studied in previous work, rely on communication for coordination and may fail if agents are unable to share information. In contrast, the algorithms that we present in this thesis are designed to work as well as possible even in the case of no communication. As communication improves, the performance of the algorithms also improves.

Chapter 3: Preliminaries

3.1 Nomenclature and Assumptions

The set of autonomous agents, denoted by A, consists of n distinct agents $A = \{a_1, \ldots, a_n\}$ and the set of tasks, denoted by T, consists of m distinct tasks $T = \{t_1, \ldots, t_m\}$. The sequence of tasks assigned to agent a_i is denoted by S_i, where $S_i \subseteq T$. In this thesis, we consider the collaborative visit scenario where the agents are supposed to visit the set of stationary targets (T) spread across a map of fixed dimensions. A target is considered to be visited if an agent comes within a threshold distance δ of the target. We compare the algorithms based on a metric, the Mission Completion Time (MCT), defined as the time when all targets have been visited at least once. This is an absolute measure of the time taken to complete the mission from an observer's point of view. It is assumed that the locations of the targets are not known before the agents are deployed. After the agents start at their respective locations at time $t = 0$, they receive details regarding the target locations. This is assumed to be a one-way message (for example, a satellite that can send a single message to the agents but they cannot send a message back, or the communication channel is expected to be jammed after a single message is sent). The multi-agent system is "decentralized", implying that there is no central computing element that

assists the agents and the agents are capable of independent computing.

3.2 Problem Definitions

We now formally define the general multi-agent task allocation problem, as well as the specific target visit problem that we study.

Problem 1, Multi-Agent Task Allocation:

Given a set of agents $A = \{a_1, \ldots, a_n\}$ *and a set of tasks* $T = \{t_1, \ldots, t_m\}$*, find sequence of tasks* S_i *for agent* a_i *such that* $\cup_{i=1}^{n} S_i = T$*.*

Problem 2, Assured Multi-Agent Target Visit With Very Limited On-The-Fly Communication:

Given an environment with very limited communication, a set of agents, given by $A = \{a_1, \ldots, a_n\}$ *known a priori and a set of targets* $T = \{t_1, \ldots, t_m\}$ *known to all agents but determined at run-time after agents are already in the field, determine agent movement in order to minimize the mission completion time (MCT), where MCT is the duration between mission start and the time when all targets* $t \in T$ *have been visited at least once.*

For the purposes of improving the state of the art, a satisfactory solution to this problem would yield a MCT that is less than the MCT of solutions generated by existing task allocation algorithms (ACBBA, DHBA and PIA).

3.3 Communication Models

Real-world communication can be modelled in a number of ways. In this thesis, we run simulations using three different communication models, that have been modelled by Nayak *et al.* [8]:

3.3.1 Bernoulli Model

The Bernoulli model uses a parameter (the Bernoulli Parameter), $p \in [0, 1]$ which is the probability that a message sent from an agent is received by another agent. The Bernoulli parameter is assumed to remain constant throughout a single simulation run. The communication attempts are assumed to be independent and identically distributed.

3.3.2 Gilbert-Elliot Model

This model assumes that the communication channel between two agents is a Markov chain, which has two possible states, good (messages can be successfully sent) and bad (messages will be dropped). Figure 3.1 illustrates the Markov chain and its states. The channel can transition between these two states after every time step t. This behavior is governed by transition probabilities, defined as follows:

(a) Good-Good Transition Probability (p_{gg}) is the probability that the channel is in the good state in the next time step, given that it is already in the good state. The complement of this probability is the Good-Bad Transition Probability

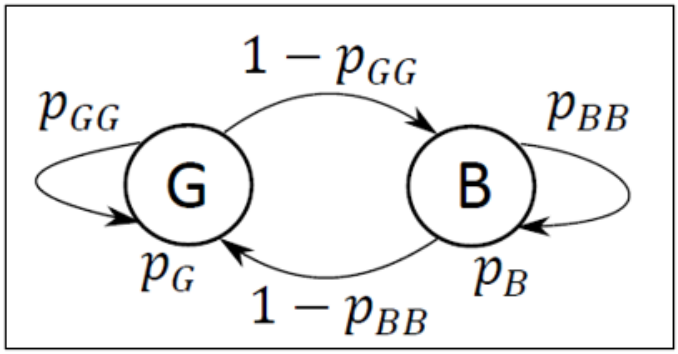

Figure 3.1: Communication channel in the Gilbert-Elliot modelled as a Markov chain

(p_{gb}), which is the probability that the channel is in the bad state in the next time step, given that it is already in the good state. Thus, $p_{gg} = 1 - p_{gb}$.

(b) The Bad-Bad Transition Probability (p_{bb}) is the probability that the channel is in the bad state in the next time step, given that it is already in the bad state. The complement of this probability is the Bad-Good Transition Probability (p_{bg}), which is the probability that the channel is in the good state in the next time step, given that it is already in the bad state. Thus, $p_{bb} = 1 - p_{bg}$.

In this research, we use a fixed value of t, which is 0.5 seconds.

3.3.3 Rayleigh Fading Model

This model considers two effects that attenuate the signal strength: fading and path loss. Fading is the attenuation in signal strength due to interference from objects in the environment. Signals from the transmitting agent take several paths to the receiving agent due to these objects, resulting in interference at the receiver's end, which may be constructive or destructive in nature. The envelope

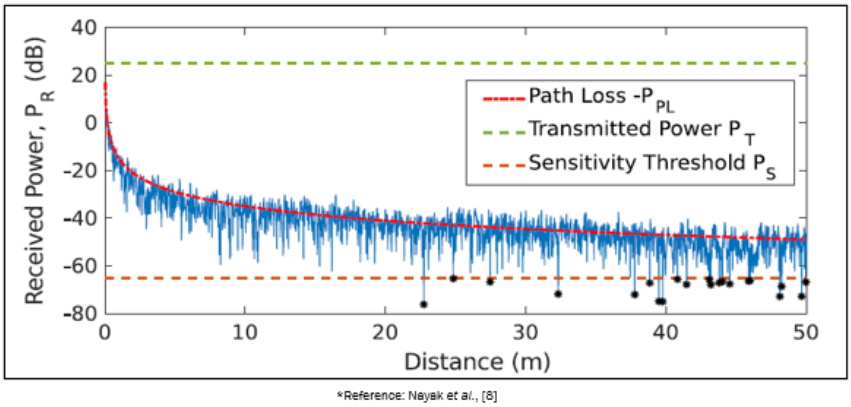

Figure 3.2: Received signal strength using the Rayleigh fading model

of the channel response varies according to the Rayleigh distribution. Nayak *et al.* [8] use Inverse Discrete Fourier Transform (IDFT) to generate the Rayleigh random variate sequence. The power loss due to fading is denoted by P_F.

Path loss is the attenuation in the signal as the distance between the transmitting and receiving agents increases. This path loss, given by P_{PL}, is given by the equation:

$$P_{PL} = P_{L_0} + 10\gamma log_{10}(\frac{d}{d_0})$$

where, d is the distance between the transmitting and receiving agents, P_{L_0} is the path loss at the reference distance d_0 and γ is the path loss exponent.

If the transmitted signal power is P_T, the total power received is given by $P_R = P_T - P_F - P_{PL}$. We define a sensitivity threshold P_S so that if P_R is less than P_S, the message packet is dropped. This drop in received signal strength due to path loss and fading is illustrated in Figure 3.2.

Chapter 4: The Playbook Algorithms

Here we propose two new task allocation algorithms that are tailored to work well under very low (or no) communication. These algorithms are modeled like a sports team's playbook, which contains rules and strategies for each player, so that each player plays their own part and the team as a whole performs as intended. With this method, the agents work independently (in the absence of communication) and yet come up with a strategy that is mutually compatible. Following this "playbook" policy, we develop two algorithms using two different approaches as follows:

4.1 Spatial Division Playbook Algorithm (SDPbA)

The Spatial Division Playbook Algorithm programs the agents to distribute tasks amongst themselves such that each agent gets a sequence S_i of tasks and satisfies $\cap_{i=1}^{n} S_i = \phi$. These sequences of tasks assigned to agents are mutually exclusive and collectively exhaustive. As shown in Figure 4.1, for a scenario with 25 targets (blue) and 5 agents, each agent running SDPbA divides the map into 5 regions (red lines mark the region boundaries). Each region contains 5 targets that are assigned to the agent corresponding to that region.

Each agent running SDPbA (Algorithm 1) takes the target set (T), number

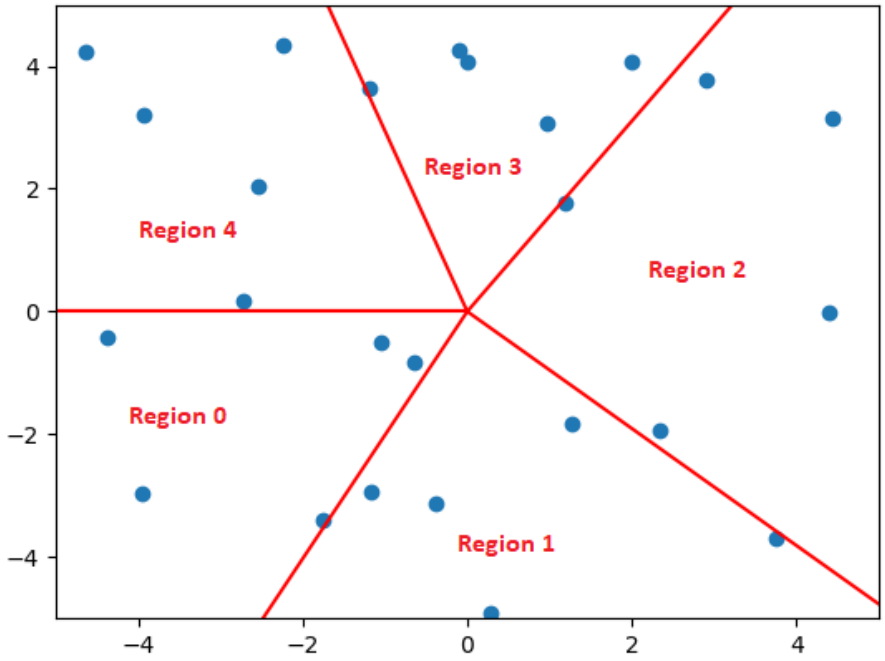

Figure 4.1: Map Slicing as done by the Spatial Division Playbook Algorithm.

of agents (n_a), increment resolution ($resolution$) and agent ID (ID) as inputs and slices the map into a number of regions equal to the number of agents (n_a) such that each region has an equal number of targets, denoted by TPR (line 5). A region is a slice of the map defined by its boundaries, which are straight lines from the centre of the map to the edges. The algorithm starts defining region boundaries at an angle of $-\pi/2$ and sweeps counter-clockwise till the next target is found and adds them to the $region$ list (line 8). The number of targets per region (TPR) is calculated as the ceiling of the ratio of remaining targets (targets not assigned to any region yet) to the number of agents. The initial value of TPR is the length of the target list T, given by the function call $len(T)$. The function `nextTarg`() sorts the targets according to their angles (polar coordinate θ) and returns (if any) targets that lie in the slice of the map bounded by $boundary$ and $angle$. $boundary$ is defined (line 6) as the angle at which the current region begins. Thus, for region 0, the boundary is

Algorithm 1: Spatial Division Playbook Algorithm

```
1  Function SDPbA(T, n_a, resolution, ID)
2      region, path←None
3      i←0
4      targsLeft←len(T)
5      TPR←⌈targsLeft/n_a⌉
6      angle, boundary←−π/2
7      while angle < π/2 do
8          region[i].insert(nextTarg(angle, T))
9          if len(region[i]) = TPR then
10             i←i + 1
11             targsLeft←targsLeft − TPR
12             n_a←n_a − 1
13             TPR←⌈targsLeft/n_a⌉
14             boundary←angle
15         angle←angle + resolution
16     path←TSPPath(region[ID])
17     appendRemainingTasks(path, region, ID)
18     while missionNotComplete do
19         sendCompletedTasksList()
20         removeCompletedTasks(path)
```

$-\pi/2$. This value of *boundary* is updated every time a new region is being considered (line 14). TPR is re-calculated each time a new region is being considered (line 13), using updated values of targets left and agents left for assignment (line 11, 12). The ceiling function ensures that if the number of agents does not perfectly divide the number of targets, agents get assigned unequal numbers of targets. For example, if there are 9 targets and 2 agents, the targets must be assigned unequally. Using the ceiling function, the value of TPR for the first agent will be 5 and 4 for the second agent.

The region being considered is changed when TPR targets are assigned to that region (line 9). Using this approach, each agent computes the exact same *region* list.

In the special case where all targets are in a single line on one side of the origin, the algorithm will allocate all targets to a single region, making the allocation inefficient. This special case has not been addressed since the probability it is realised over a uniform randomized generation of targets is zero. However, if ties are a concern, then any deterministic tie-breaking algorithm can be used. The algorithm returns an approximate Travelling Salesman Problem (TSP) solution corresponding to the region that has the same ID as the agent (line 16), in the function `TSPPath`(). This approximate solution is computed using Christofides' algorithm [24], which guarantees a solution that is no worse than 1.5 times the optimal TSP solution in $O(n^3)$ time. Following this, the TSP-approximate solutions for the remaining regions are appended to the path (line 17) to ensure that agents move on to complete other tasks when done with their assigned regions. After assigning pieces among themselves, the agents periodically broadcast their completed tasks lists to other agents (line 19). If they receive such a message from other agents, they remove the tasks that the transmitting agents have completed from their own task list (line 20). These actions are repeated till the mission is complete, which is checked by the flag $missionNotComplete$ (line 18).

4.2 Travelling Salesman Playbook Algorithm (TSPbA)

The Travelling Salesman Playbook Algorithm programs the agents to compute identical approximate solutions to the Travelling Salesman Problem (TSP) where each target in the map is a node of the TSP. Each of the n agents computes an

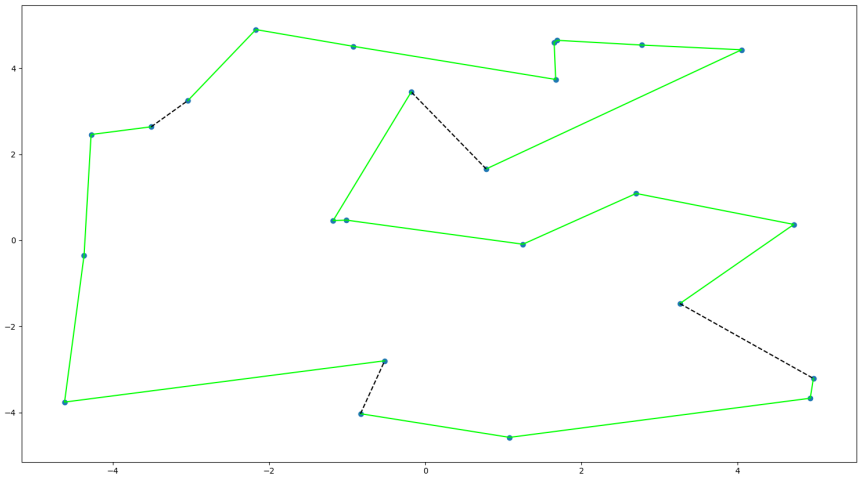

Figure 4.2: Division of TSP solution as done by the Travelling Salesman Playbook Algorithm.

approximately optimal path P through all targets using Christofides' algorithm and then divides that path into n pieces $(p_1, \ldots, p_n)$, each piece having length l_i, such that each piece p_i satisfies $p_i \subseteq P$. The pieces assigned to all agents are sequences of targets and these sequences are mutually exclusive and collectively exhaustive. To help equalize the load distribution among the agents, we divide the path into pieces such that the maximum of the elements of L is minimized, where L is the set $\{l_1, \ldots, l_n\}$.

An example is illustrated in Figure 4.2, where the algorithm computes a TSP-approximate path through 25 targets (blue). Assuming 5 agents, this path is divided into 5 pieces (green). Black dashed lines indicate edges of the path that are part of the TSP solution but are not assigned to any agent, since the vertices that define the edge are both assigned to one agent each. The number of targets per piece of TSP path is optimized such that each agent gets a near-equal amount of load in

terms of length of the piece assigned to it.

Algorithm 2 details the working of TSPbA. The algorithm accepts the target set T, number of agents n_a and the agent ID, denoted by ID. The algorithm starts by solving the TSP using Christofides' algorithm (line 3), which returns the path and edges. Edges is a list containing lengths of all the edges of the TSP path. Partitions is list of indices that separate two pieces of the path, to be assigned to two different agents. In line 4, the function `getNaiveParts`() generates a naïve list of partitions that divide the path into pieces such that each piece has approximately the same number of targets. This naïve partitions list is iteratively changed (lines 5 and 6) such that the variance of the set L is minimized. This is done by the function `minMax`() by looping through each partition index and making increments and decrements to it in order to minimize the maximum length in L. This process is iterated *iter* times. In each iteration, the algorithm considers each piece in order and adds or removes edges from that piece till the maximum element of L is minimized.

The end product of this loop is a partitions list that divides the path among agents in the most equal manner. The function `dividePath`() (line 9) is responsible for assigning the piece corresponding to the agent ID to the agent. The rest of the path is also appended to *pieces* by this function to ensure that after completing its own part, each agent moves on to complete the rest of the path, in the case when other agents are unable to complete their assigned tasks. After assigning pieces among themselves, the agents periodically broadcast their completed tasks lists to other agents (line 11). If they receive such a message from other agents, they remove the tasks that the transmitting agents have completed from their own task list (line

Algorithm 2: Travelling Salesman Playbook Algorithm

```
1  Function TSPbA(T, n_a, ID, iter)
2      pieces, partitions, edges, path←None
3      path, edges←TSPPath(T)
4      partitions←getNaiveParts(edges, n_a)
5      while iter do
6          for i in partitions do
7              partitions←minMax(partitions, i)
8          iter = iter − 1
9      pieces←dividePath(path, partitions, ID)
10     while missionNotComplete do
11         sendCompletedTasksList()
12         removeCompletedTasks(path)
```

12). These actions are repeated till the mission is complete, which is checked by the flag *missionNotComplete* (line 10).

4.3 Caveats of the Playbook Approach

The general idea of the playbook approach used in the two proposed algorithms is to make all the agents come up with identical solutions and divide the solution between all agents such that there is no overlap. To keep the solutions identical, the starting positions of agents do not play a part in either coming up with a solution or in dividing the solution into parts. As the starting location of agents is random, an agent has to travel from its starting location to the first target assigned to it. We term this travel as the 'first haul' and the distance travelled in the first haul as the 'startup cost' of that agent, since no targets are visited till this distance is traversed by the agent. As a result of this first haul, the MCT is affected with a magnitude that is proportional to the startup cost of agents, which may be small or

large, depending on the starting locations of agents.

Chapter 5: Existing Collaborative Task Allocation Algorithms

The following three algorithms are used for comparison with the Playbook Algorithms:

5.1 Asynchronous Consensus-Based Bundling Algorithm (ACBBA)

This algorithm (Algorithm 3) is an improved version of CBBA. The input arguments are distance travelled (d_i), winning bids list (W_i), task list (T), iteration count (I) and bundle size upper bound (B). It operates in two phases, assignment and consensus. In the assignment phase each agent greedily determines an ordered task set, called a bundle (b_i) and updates its winning bid list with the bids of the task list. In the consensus phase, the agents broadcast messages that contain their bids list. If agents receive messages from other agents that have better bids, they update their own winning bids lists, thus achieving consensus. Both the phases are repeated by each agent I times.

5.2 Performance Impact Algorithm (PIA)

The input arguments to this algorithm (Algorithm 4) are distance travelled (d_i), significance list (S_i), task list (T), iteration count (I) and bundle size upper

Algorithm 3: ACBBA

```
1 Function ACBBA(d_i, W_i, T, I, B)
2     b_i ← None
3     for k ← 1 to I do
4         (b_i, W_i) ← Assignment(b_i, d_i, W_i, T, B)
5         SendBids(W_i)
6         (b_i, W_i) ← Consensus(b_i, W_i, B)
7     return b_i
```

bound (B). PIA modifies CBBA by having two phases, namely the task inclusion phase and the consensus and task removal phase. In the task inclusion phase, each agent calculates the 'significance' of tasks not included in its bundle (b_i) and updates the task bundle and significance list. The significance of a task is the impact that the task has on the cost of the bundle. In the consensus and task removal phase, agents exchange their significance lists and achieve consensus by updating their bundles for tasks that have been outbid by other agents. Both the phases of this algorithm are repeated I times.

Algorithm 4: PIA

```
1 Function PIA(d_i, S_i, T, I, B)
2     b_i ← None
3     for k ← 1 to I do
4         (b_i, S_i) ← Assignment(b_i, d_i, S_i, T, B)
5         SendSignificanceList(S_i)
6         (b_i, S_i) ← ConsensusAndRemoval(b_i, S_i, B)
7     return b_i
```

5.3 Decentralized Hungarian-Based Algorithm (DHBA)

In DHBA, as shown in 5, the input arguments are distance travelled (d_i), task list (T) and iteration count (I). Each agent first initializes a cost matrix (C_i) corresponding to the cost to complete all unfinished tasks and then proceeds to two phases, the assignment phase and the update phase. In the assignment phase, each agent runs the Hungarian Assignment algorithm to get an unfinished task (t_i) and broadcasts its cost matrix to all other agents. In the update phase, each agent receives the matrix from other agents and updates its own cost matrix. Similar to the other algorithms, this process is repeated by each agent I times.

Algorithm 5: DHBA

```
1 Function DHBA(d_i, T, I)
2     t_i ← None
3     C_i ← None
4     for k ← 1 to I do
5         t_i ← Assignment(C_i)
6         SendCostMatrix(C_i)
7         C_i ← Update(C_i)
8     return C_i
```

Chapter 6: Experimental Setup

6.1 Simulation Framework

Experiments were carried out using a simulator developed in ROS [16][18] with Python and C++ as programming languages in a Linux environment. The simulation framework consists of two module types: the agent module and the central environment simulator module.

Each agent module is an independent processing unit in the CPU and simulates a unique autonomous agent that runs the task allocation algorithm. Agents exchange messages with each other based on the three communication models: the Bernoulli model, Gilbert-Elliot model and Rayleigh fading model. The decisions made by agents, total number of messages sent and received by the agents, time taken and distance traveled to visit targets are all recorded in log files when running the simulation.

6.2 Design of Experiments

An experiment instance is defined by the number of targets, the random target locations, the agent starting locations, target cluster locations and a communication

level. In this thesis we use a constant value of 5 agents, and other parameters are generated within constraints as given in Table 6.1. Since this set of experiments is aimed at testing the performance of task allocation algorithms in conditions with very low communication availability, the scenarios are generated for five levels of very low communication as shown in Tables 6.2, 6.3 and 6.4. Level C0 corresponds to the least communication and this increases till level C5. For each communication model, the rationale behind defining parameters for each of the communication levels is as follows: the lowest communication level (C0) corresponds to no communication at all. At this level, the playbook algorithms perform better in terms of mean value of MCT as compared to the existing task allocation algorithms. We increase the communication availability till we observe the point where the existing algorithms start performing better than at least one of the playbook algorithms.

When using the Gilbert Elliot model, algorithms exhibit a variation in performance across a wider range of transition probabilities. Hence, as an exception, this model has six communication levels. Of these six levels, C0 corresponds to zero communication, and hence is identical to level C0 of the Bernoulli model. Thus, results with communication level C0 for the Gilbert-Elliot model have not been included in this study.

The map for the simulation is of dimensions $M \times M$ where M is 100 units and the origin is defined to be the bottom left corner of the map. For the purpose of generating scenarios, it is assumed that targets exist in clusters. The targets are modeled in clusters because this represents real-world scenarios in which targets need not be concentrated in a single region. A target cluster is defined as a circular

Table 6.1: Parameters for Design of Experiments

Parameter	**Value**
Number of Targets m	10, 20, 30, 40, 50, 60
Communication Levels	C0, C1, C2, C3, C4
Number of Target Clusters	Random Integer in [1, 4]
Radius of Target Clusters	Random Floating Point Value in [5, 50]
x, y Coordinates of Clusters, Agents and Targets	Random Floating Point Value in [1,100]

Table 6.2: Definition of Communication Levels: Bernoulli Model

Communication Level	**Bernoulli Parameter**
C0	0.0
C1	0.00005
C2	0.0001
C3	0.0002
C4	0.0005

region of radius given by cluster radius, centered at the cluster centre. A number of target clusters is chosen, followed by the cluster centre locations and the cluster radii, all chosen randomly with the constraints provided in Table 6.1. Using these parameters, targets are then placed within the clusters based on a uniform distribution. Figure 6.1 illustrates an instance of random scenario generation, where 80 targets are generated in 3 clusters of radii 10 units each.

Lastly, the agents are placed at random locations on the map using a uniform distribution. A target is said to be "visited" if an agent moves within a threshold distance (δ) of 0.25 units of the target. The agents in the simulation are assumed to have on-board computers. At the start of the experiment, the dimensions of the map and the locations of all the targets are assumed to be known to the agent.

Table 6.3: Definition of Communication Levels: Gilbert-Elliot Model. *C0 is not included in the results due to similarity with C0 of Bernoulli model.

Communication Level	**Good-Good Transition Probability (p_{gg})**	**Bad-Bad Transition Probability (p_{bb})**
C0*	0.0	1.0
C1	0.0001	0.9999
C2	0.002	0.998
C3	0.005	0.995
C4	0.01	0.99
C5	0.02	0.98

Table 6.4: Definition of Communication Levels: Rayleigh Fading Model

Communication Level	**Sensitivity Threshold (dB)**
C0	30
C1	25
C2	20
C3	10
C4	0

Also, it is assumed that the agent knows its own location at all times, but has no information about other agents except for the total number of agents.

For a single task allocation algorithm, there are 90 possible combinations of communication model, communication level and number of targets. For each combination, 40 random scenarios are generated by varying the number of target clusters, radius of clusters and agent and target locations, making a total of 3,600 scenarios for each algorithm. Since five task allocation algorithms are being compared, a total of 18,000 experiments were conducted for this analysis.

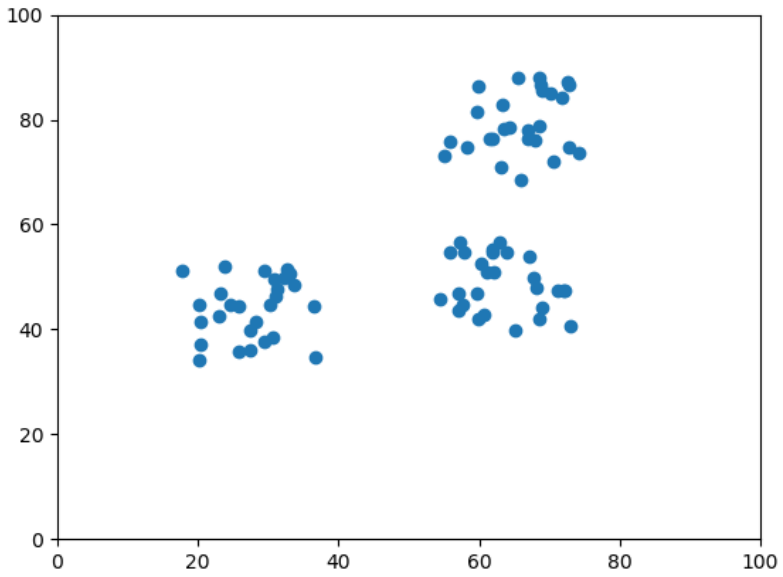

Figure 6.1: Illustration of random target generation in clusters

Chapter 7: Experimental Results

We visualize the experiment results using boxplots and line plots. A single box represents the mission completion time (in seconds) for an algorithm for 40 distinct experiments corresponding to a single combination of the parameters shown in Table 6.1. The line plots show how the mean performance of an algorithm varies as the number of targets (represented as the x-axis) and the Communication Level (varies across plots) are varied.

In the boxplots in Figures 7.1, 7.2 and 7.3, each box represents the results obtained from 40 experiments run on a single algorithm for a given communication level and number of targets. It can be observed that for all three communication models, the standard deviation in MCT of the solutions generated by all the existing task allocation algorithms increases as the number of targets increases and decreases as the communication level increases. Standard deviations of the MCT of the solutions generated by SDPbA and TSPbA are smaller as compared to those of the solutions generated by the other three algorithms. This suggests that for any given combination of simulation parameters within the scope of these experiments, SDPbA and TSPbA exhibit the least amount of variation in performance.

From the line plots in Figures 7.4, 7.5 and 7.6, we infer that for a fixed com-

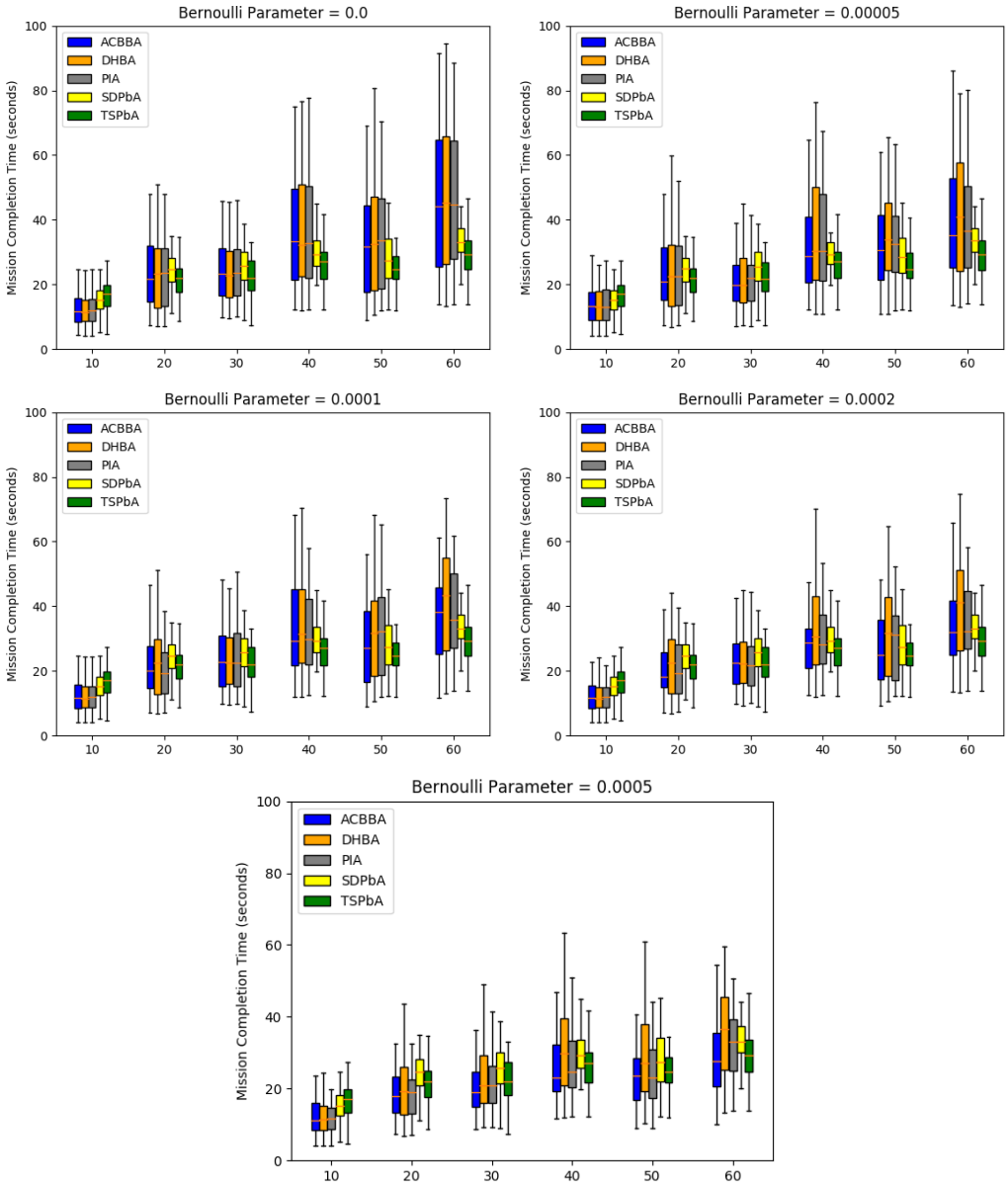

Figure 7.1: Box plots of mean Mission Completion Time (MCT) when using the Bernoulli model across all values of Bernoulli parameter (BP). A lower MCT indicates better performance.

munication level, as the number of targets increases from 10 to 60, the performance of SDPbA and TSPbA improves relative to ACBBA, DHBA and PIA. This performance refers to the mean MCT across all experiments as seen in the line plots. We observe that for a lower number of targets (typically less than 30), ACBBA performs

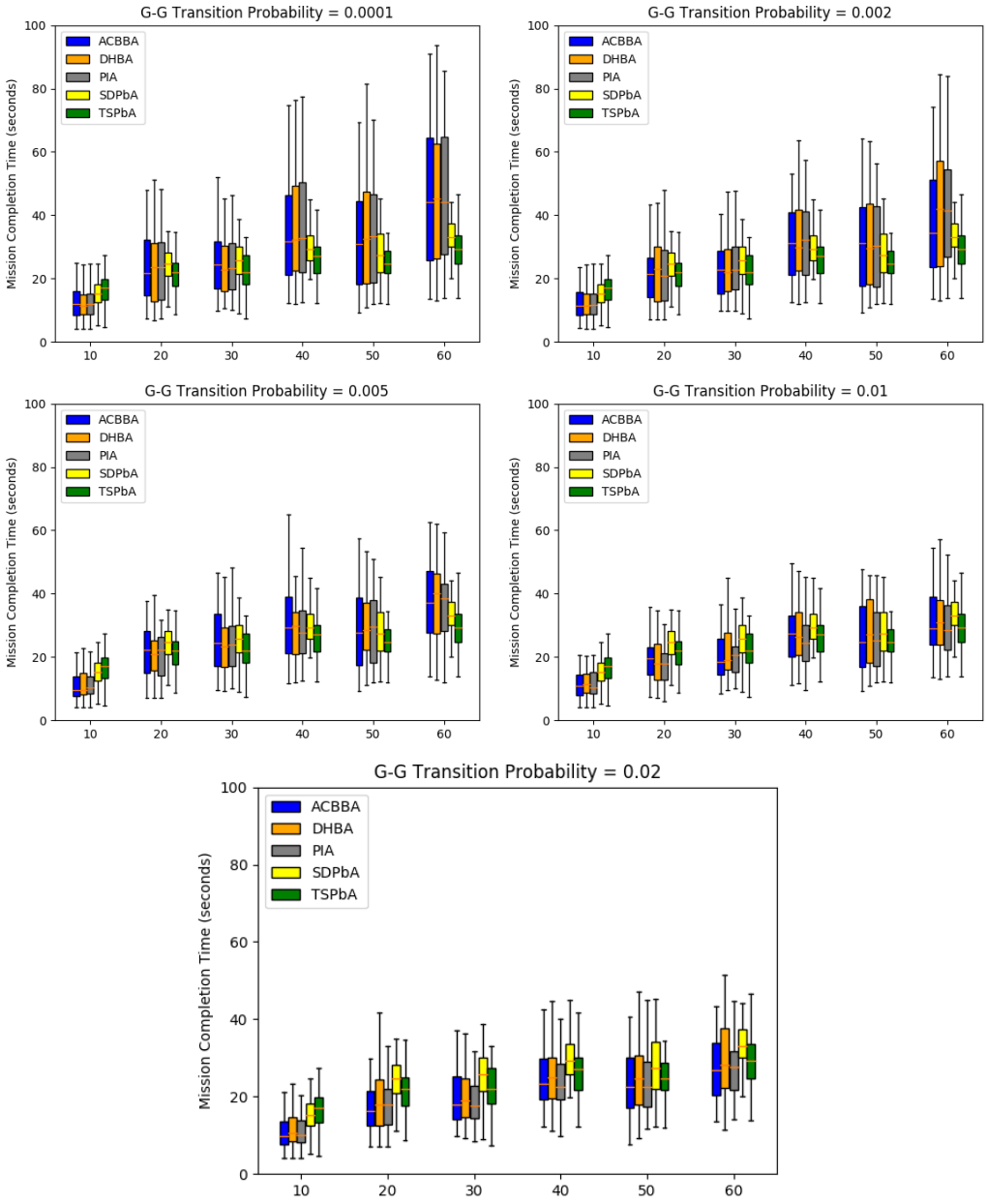

Figure 7.2: Box plots of mean Mission Completion Time (MCT) when using the Gilbert-Elliot model across all values of Good-Good Transition Probability. A lower MCT indicates better performance.

the best, and for a higher number of targets, the playbook algorithms outperform the existing algorithms, with TSPbA performing the best.

As the communication availability increases, the existing algorithms (ACBBA, DHBA and PIA) are able to communicate more often and as a result are able to

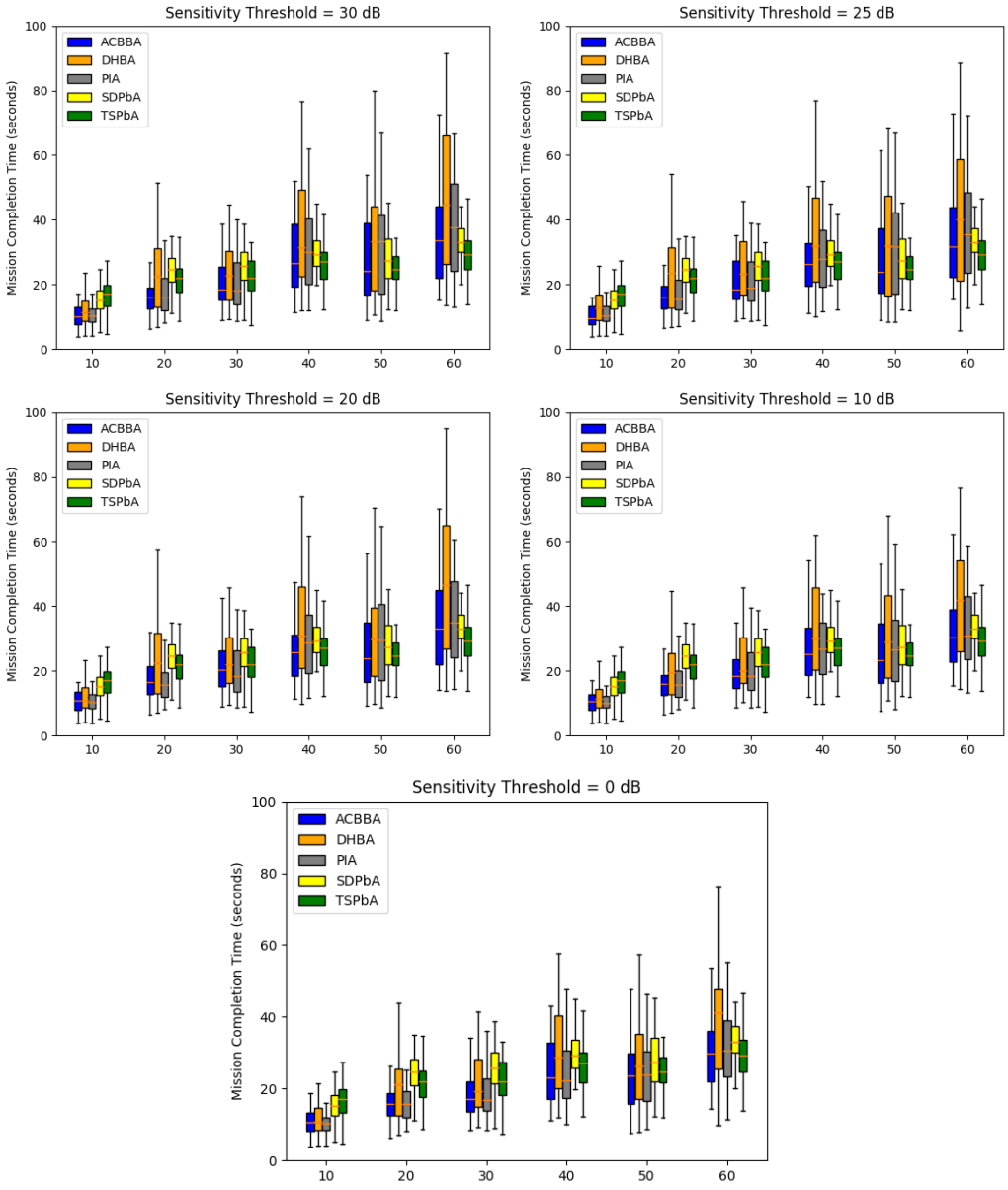

Figure 7.3: Box plots of mean Mission Completion Time (MCT) when using the Rayleigh fading model across all values of Sensitivity Threshold. A lower MCT indicates better performance.

share information with each other that improves the overall task allocation. We see from the line plots for the all models that as the communication level increases, the margin by which the playbook algorithms are better than the other three algorithms decreases. In the plots for the Bernoulli model, when the p is 0.0005, the playbook

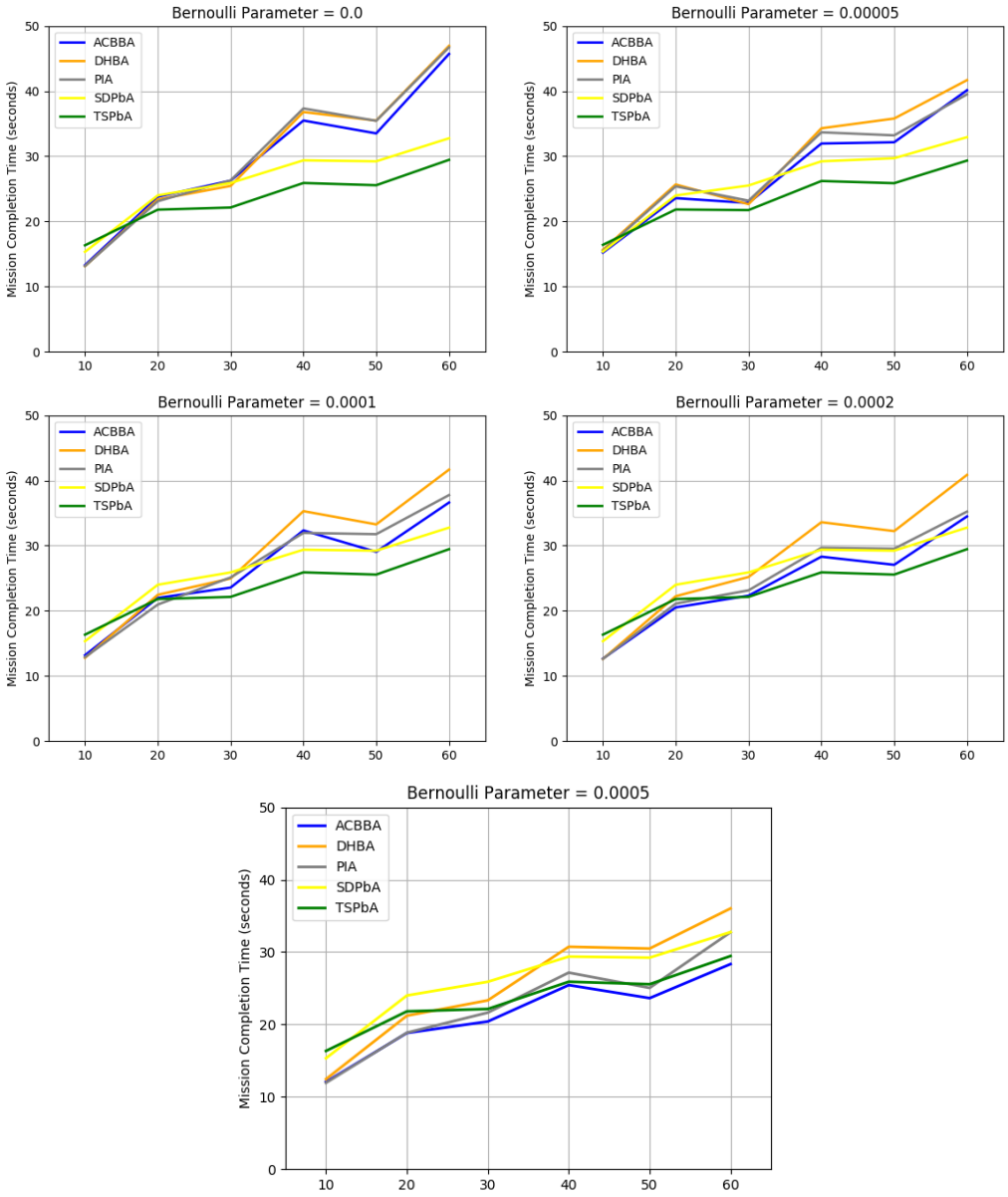

Figure 7.4: Line plots of mean Mission Completion Time (MCT) when using the Bernoulli model across all values of Bernoulli parameter (BP). A lower MCT indicates better performance.

algorithms are not the best performers. Similarly, for the Gilbert-Elliot model, we observe that the playbook algorithms cease to be the best performers when the p_{gg} increases to 0.02, and when the sensitivity threshold becomes 10 dB for the Rayleigh model.

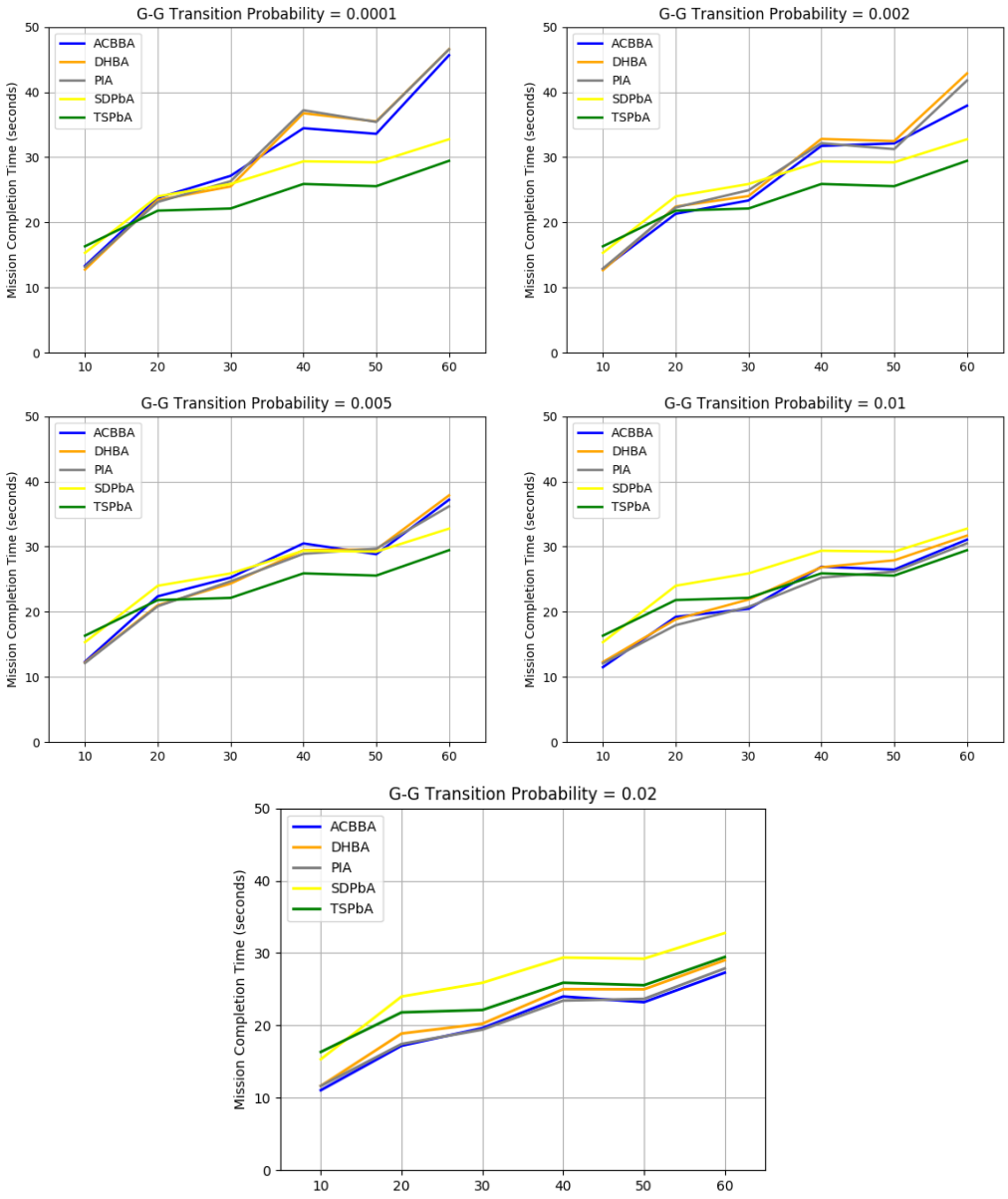

Figure 7.5: Line plots of mean Mission Completion Time (MCT) when using the Gilbert-Elliot model across all values of the Good-Good Transition Probability. A lower MCT indicates better performance.

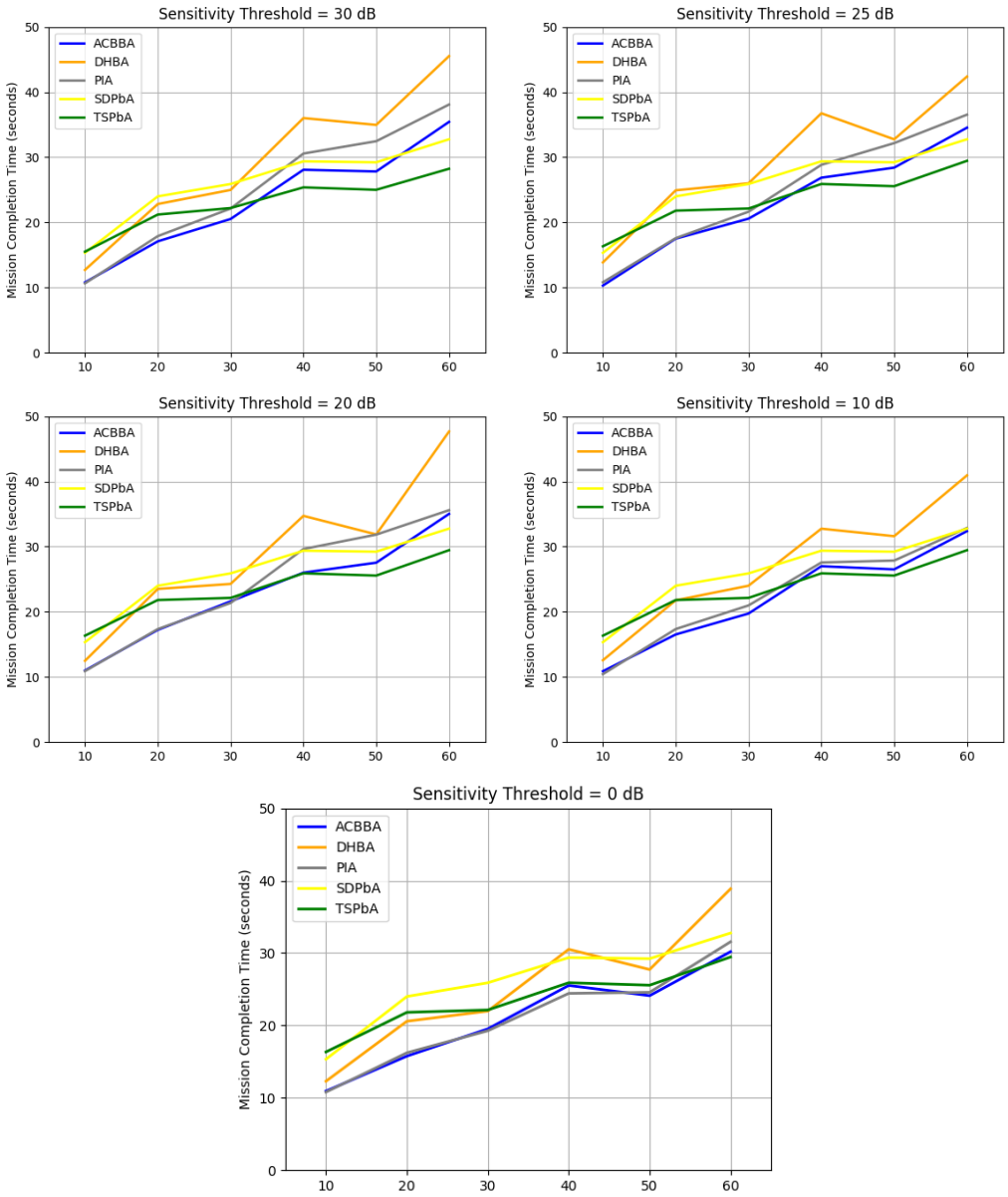

Figure 7.6: Line plots of mean Mission Completion Time (MCT) when using the Rayleigh fading model across all values of Sensitivity Threshold. A lower MCT indicates better performance.

Chapter 8: Analysis

8.1 Startup Cost of Playbook Agents

As discussed in chapter 4, the startup cost of agents results in a variability in the performance of the playbook algorithms, which depends on the starting locations of agents and the division of targets that the agents come up with. As the number of targets increases, the startup cost gets smaller in comparison to the time required to visit targets assigned to that agent. This result can be mathematically proven for TSPbA as follows:

Definition 8.1. Event Space

The event space for n agents and m targets (denoted by $\mathbf{X}_{n,m}$) is the space containing all possible starting locations of agents and targets within the given workspace.

We consider the full event space (as described above) as the Cartesian product of the event spaces for the target and the event space for the agents, denoted respectively as $\mathbf{X}_m$ and $\mathbf{X}_n$, such that:

$$\mathbf{X}_n = \mathbf{W}_1 \times \ldots \times \mathbf{W}_n = \mathbf{W}^n$$

$$\mathbf{X}_m = \mathbf{W}_1 \times \ldots \times \mathbf{W}_m = \mathbf{W}^m$$

where, $\mathbf{W}_i = \mathbf{W}$ for all i. Thus, $\mathbf{X}_{n,m} = \mathbf{X}_n \times \mathbf{X}_m$.

Definition 8.2. Event

An event for n agents and m targets (denoted by $\mathbf{x}_{n,m}$) is an element of the event space $\mathbf{X}_{n,m}$, which represents a single instance of n randomly located agents and m randomly located targets in the given workspace.

Note: An event $\mathbf{x}_{n,m}$ is a multi-variate random variable.

Also note that, similar to the event space, an event satisfies $\mathbf{x}_{n,m} = \mathbf{x}_n \oplus \mathbf{x}_m$.

Definition 8.3. TSP Distance Function

The TSP distance function, F: $\mathbf{X}_{n,m} \rightarrow \mathbb{R}$, is defined as the distance (cost) of the optimal TSP solution to the problem defined by event $\mathbf{x}_{n,m}$.

[26] states that the length of the optimal TSP solution as the number of cities in the TSP becomes very large is asymptotically proportional to the square root of the number of cities. This can be formally stated as:

Proposition 8.1. *Assuming that a target sequence of length m is drawn from a uniform distribution such that all targets are bounded by a closed region of area v, then for a constant $c \in (0, \infty)$, the length of the optimal solution (L) to the TSP corresponding to the target sequence almost surely satisfies:* $\lim_{m\to\infty} \frac{L}{\sqrt{mv}} = c$.

Lemma 8.1. *In a square map of dimensions $M \times M$, the length of the optimal solution to the travelling salesman problem, L, satisfies:* $\lim_{m\to\infty} \frac{L}{M\sqrt{m}} = c$, *for*

some $c \in (0, \infty)$.

Proof: We divide this proof in two parts: (1) the optimal solution to the TSP satisfies $\lim_{m\to\infty} \frac{L}{M\sqrt{m}} = c$, and (2) such a scenario where the solution to the TSP satisfies the condition given in (1) exists almost surely.

Part 1: As a consequence of proposition 8.1, we can say that there exists a finite and positive c such that: $\lim_{m\to\infty} \frac{L}{M\sqrt{m}} = c$.

Part 2: The event space for a 2-dimensional workspace with n agents and m targets is of the dimensions $2(n + m)$. Event $\mathbf{x}_{n,m} \in \mathbf{X}_{n,m}$ can be represented as $[t_1, \ldots, t_m, a_1, \ldots, a_n]$, where a_i is the i^{th} agent location, and t_j is the j^{th} target location; note that all a_is and t_js are random variables in $\mathbf{W}$, where $\mathbf{W}$ is the 2-dimensional workspace. From the experimental setup, we know that the probability distribution for targets and agents is non-zero at every point in the workspace.

Given this and the results of [26], it follows that, almost surely:

$$\lim_{m\to\infty} \frac{F(\mathbf{x}_{n,m})}{\sqrt{mv}} = c$$

for some $c \in [0, \infty)$.

In the scenarios under consideration, the map is a square with area M^2. Hence,

$$\lim_{m\to\infty} \frac{F(\mathbf{x}_{n,m})}{M\sqrt{m}} = c$$

□

Lemma 8.2. *The maximum possible startup cost,* L_{Smax} *is given by* $L_{Smax} = M\sqrt{2}$

Proof: The startup cost (denoted by L_S) for an agent is the distance from its starting location to the first target assigned to the agent. The worst-case startup cost corresponds to the maximum possible distance between any two points on the map, which is the diagonal length of the square map. Thus, the maximum possible startup cost, L_{Smax} is given by: $L_{Smax} = M\sqrt{2}$ □

Theorem 8.3. *The fraction of time spent by a TSPbA agent (f) in the first haul satisfies* $\lim_{m\to\infty} f = 0$.

Proof: Since the agents move at a constant speed, the time taken (MCT) is proportional to the total distance travelled (L_{total}). The total distance is the sum of the startup cost and the cost of the solution to the TSP. Thus, $L_{total} = L + L_S \leq L + L_{Smax}$. Hence,

$$L_{total} \leq L + M\sqrt{2}$$

The fraction of time spent in the first haul (f) is given by $f = \frac{L_S}{L_{total}}$. From Lemma 8.2 we know that as the number of targets (m) becomes very large, the denominator L_{total} becomes asymptotically proportional to $M\sqrt{m}$, while the numerator L_S has an upper bound L_{Smax}. Thus,

$$\lim_{m\to\infty} f = 0$$

□

This result is proven for TSPbA, but can be extended to SDPbA as well.

Corollary 8.3.1. *The fraction of time spent by a SDPbA agent in the first haul (f) satisfies $f \to 0$ as $m \to \infty$.*

Proof: In SDPbA, agents do not compute a mutually consistent TSP solution, but divide the map into regions and then compute TSP solutions independently. In this case as well, the targets assigned to an agent are bounded by a region that has an area λM^2, where $\lambda \in (0, 1]$. Lemma 8.1 implies that $\lim_{m\to\infty} \frac{L}{M\sqrt{m}} = \sqrt{\lambda}c$ for some $c \in [0, \infty)$, and the worst-case startup cost given by lemma 8.2 still holds true, making $L_{Smax} = M\sqrt{2}$. Thus, it follows from theorem 8.3 that $\lim_{m\to\infty} f = 0$. □

8.2 Bandwidth Requirements

Owing to their unique mechanisms, different task allocation algorithms have different bandwidth requirements in terms of message size and the frequency of sending messages to work effectively.

In ACBBA, each of the n agents creates a bundle containing B targets. The agent creates a winning bids list, winning agents list and a timestamp record list in a single iteration and sends these lists as a message to other agents. The agent also receives messages from other agents and updates its lists based on the messages received. This process is iterated I (iteration count) times. These I iterations are carried out in one time step and the number of time steps depends on the total duration of the simulation, which in turn depends on multiple factors such as locations of targets, communication level and starting locations of agents. Assuming perfect communication, the total number of computations involved in one time step for one agent is $O(mnI + mB^2I)$. I and B are values that are chosen by the user but are constant throughout all the experiments. Hence, for a given set of (B, I),

the complexity for a single agent is $O(mn)$. The agents send messages that contain three lists, the winning bids list, winning agents list and the timestamp record list, which are of lengths m, m and n respectively. Hence, the message size is of the order $O(m+n)$.

PIA works similar to ACBBA and hence it also has $O(mn)$ complexity for a single agent, and a message size of order $O(m+n)$.

In DHBA, each agent maintains a cost matrix $(m \times n)$ and exchanges this matrix with other agents to achieve consensus. Hence, a single agent in a single time step running DHBA has complexity $O(mn)$. Since this matrix is sent as a message as well, the message size is also of the order $O(mn)$.

The playbook algorithms have smaller requirements in terms of bandwidth, as the agents running SDPbA or TSPbA only communicate completed tasks with each other. Also, the computation required to calculate paths to targets is done by agents at the beginning of the mission, after which they only visit targets and send out task completion messages to each other. The only computation involved when completing tasks is when agents receive messages from other agents and remove completed tasks from their own paths. Hence, the computational complexity as the agents are moving is $O(mn)$ and the message size is $O(m)$.

All these observations are shown in table 8.1 below:

Table 8.1: Bandwidth Requirements for Algorithms

Algorithm	**Computational Complexity**	**Message Size Complexity**
ACBBA	$O(mn)$	$O(m+n)$
PIA	$O(mn)$	$O(m+n)$
DHBA	$O(mn)$	$O(mn)$
SDPbA	$O(mn)$	$O(m)$
TSPbA	$O(mn)$	$O(m)$

Chapter 9: Discussion and Results

9.1 Overall Performance of the Playbook Algorithms

The proposed playbook algorithms perform better than existing task allocation algorithms in scenarios when the communication is very low. In terms of mean MCT, the SDPbA performs better than the three existing algorithms for communication levels C0 through C3 of the Bernoulli model, C0 through C3 of the Gilbert Elliot model, and C0 through C2 for the Rayleigh fading model. The TSPbA performs better than the SDPbA in most of the cases; it performs better than the three algorithms for communication levels C0 through C3 of the Bernoulli model, C0 through C4 of the Gilbert-Elliot model, and C0 through C4 of the Rayleigh fading model.

9.2 Smaller Standard Deviation in MCT for Playbook Algorithms

In chapter 7, we observe from the box plots in figures 7.1, 7.2 and 7.3 that the standard deviation in MCT for the playbook algorithms is smaller than the standard deviation in MCT for existing task allocation algorithms.

This behavior can be attributed to different mechanisms used by the collab-

orative algorithms. The performance of ACBBA, DHBA or PIA depends on the messages received from other agents. From the logs of our experiments, we observe that in *any* scenario, agents running these three algorithms send out an average of approximately 66, 19 and 48 (respectively for ACBBA, DHBA and PIA) messages per second. Depending on the communication level, these messages may or may not be delivered.

Although the communication model parameter for a scenario may seem very small, its effect is observable for simulations as shown in the following example: If in a Bernoulli model experiment, the Bernoulli Parameter p is 0.0001, the probability that *at least one message is delivered to exactly one agent* successfully over an ACBBA simulation that lasts t seconds is given by the expression $(1-(1-p)^{mt})$, where m is the number of messages attempted per second ($m = 66$ for ACBBA). For a simulation duration of 30 seconds, this probability is approximately 0.18. Similarly, for a Gilbert-Elliot model experiment that has a duration of 30 seconds, the channel state changes 60 times (since the time step is 0.5 seconds). In each state, an agent sends $m/2$ messages. If the state is good, all the $m/2$ messages will be delivered successfully, and the probability that the channel state is good for *at least one time step* is given by the expression $1-p_{bb}^{2t}$, where t is the simulation duration. This probability for a 30 second ACBBA simulation with $p_{bb} = 0.995$ is 0.26. For the Rayleigh model, the probability of a message getting dropped is highly scenario-dependent as the model considers path loss due to distance between agents, which changes with the scenario geometry.

The successful messages can change the output of the simulation, depending

on several randomized factors, such as the locations of the communicating agents. For example, if agents A and B start close to each other in the map, they will have similar costs to all targets. In this case a successfully delivered message from A to B will not be of great value because B does not receive any new information regarding costs to targets. On the contrary, if one of the agents starts away from the other agent, they will have different costs to targets. In which case, a successfully delivered message from A to B will provide new information to B and it will re-evaluate its own path and give lower priority to targets that are away from itself but closer to A.

In addition to the number of messages received by each agent, the overall geometry of the scenario also has a noticeable effect on the performance of all four algorithms. The geometry includes the locations of targets in clusters as well as starting locations of agents. If the targets are oriented in clusters in such a way that there are a few outlying targets, and if no agent has a starting location that is close to the outlying targets, then the outlying targets tend to end up at the bottom of the priority lists of the agents when they run ACBBA, DHBA and PIA. As a result, the agents visit all other targets first and then move to visit the outlying targets, resulting in a poor overall performance (a high MCT). SDPbA and TSPbA avoid this problem because their target assignment logic guarantees that one agent is assigned the outlying target(s).

Another case where the starting locations of agents plays an important role is when they all start close to each other. In this case, if the communication level is on the lower side of the range included in the experiments, the agents exhibit a

leader-follower behavior when running ACBBA, DHBA and PIA, because the lack of communication results in several (or in some cases, all) agents following one agent in completing several tasks. This occurs because all the agents have similar costs to targets and their inability to communicate and reach consensus results in them exhibiting this behavior. Due to this, the team of agents produces results that are no better than results produced by a single agent by itself. In this case, the playbook algorithms perform much better because the agents diverge to different sets of tasks.

9.3 Effect of Number of Targets

From the line plots in figures 7.4, 7.5 and 7.6 in chapter 7, we observed that for a fixed communication level, as the number of targets increases from 10 to 60, the performance of SDPbA and TSPbA improves relative to ACBBA, DHBA and PIA.

We hypothesize that this behavior is the result of the characteristic behavior of the playbook algorithms. When agents run either SDPbA or TSPbA, they assign distinct regions of the map to themselves. Because the agents start at random locations in the map, it is likely that the region assigned to an agent is away from its starting location, as shown in Figure 9.1. This results in a long first haul, which acts as a startup cost for the agent, subsequently increasing the MCT. The long haul is depicted by a dotted black line in Figure 9.1. This behavior is observed particularly when the number of targets is small, as the time required for an agent running SDPbA or TSPbA to complete the first haul is often much larger than

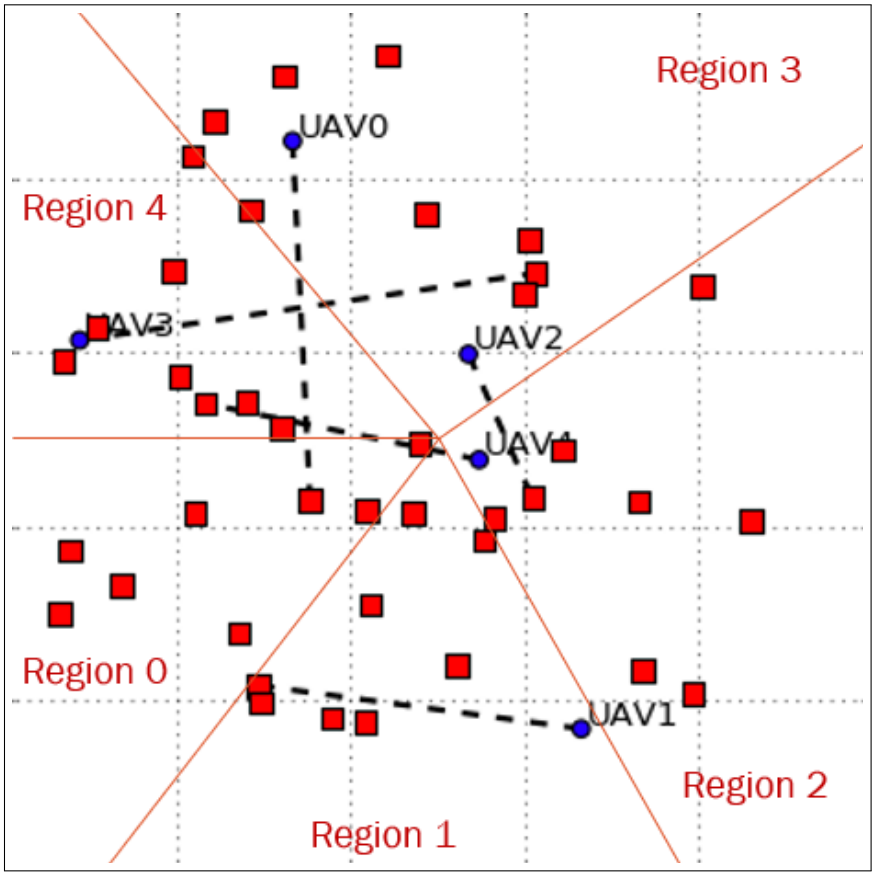

Figure 9.1: An instance of agents completing long first hauls to enter their respective regions.

the time required to visit all targets in its region. In such cases, the other three algorithms perform better than either of the playbook algorithms.

As a result of theorem 8.3 and corollary 8.3.1, as the number of targets increases, SDPbA and TSPbA become favorable algorithms in comparison to the three existing task allocation algorithms.

9.4 Effect of Communication Level and Communication Model

In chapter 7, we observe the range of communication levels and number of targets in which the playbook algorithms perform better than existing task alloca-

tion algorithms. The playbook algorithms perform worse than other task allocation algorithms as the communication level increases.

Thus, one downside of the playbook algorithms vs. existing methods is that, in cases where communication availability is high, the team is unable to use the extra communication to generate a new strategy tailored to the specific starting locations of the agents. For this reason, SDPbA and TSPbA are particularly relevant to cases where communication quality is lower than that required by existing algorithms.

In the Bernoulli model and the Gilbert-Elliot model, the success of an attempted message between agents depends only on the global probability (p) and the communication channel state respectively. On the other hand, in the Rayleigh fading model, the geometry of the scenario also plays an important role in the performance of algorithms, because this model also takes into account the change in communication quality due to path loss. Thus, if a randomly generated scenario has several targets closely packed in a cluster, the agents will be in close proximity to each other during the task completion, resulting in a smaller path loss, subsequently resulting in better MCT.

In contrast, if targets are spaced out in the scenario, the path loss will be higher, resulting in agents not being able to communicate as efficiently as in the previously mentioned case. This will result in a worse MCT. This high dependency on the geometry of the scenario results in a high variation in the performance of algorithms when using the Rayleigh fading model. We hypothesize that this is the reason behind the standard deviations for all communication levels and all algorithms being higher for the Rayleigh fading model as compared to the other two

models, as seen in Figures 7.1, 7.2 and 7.3.

Chapter 10: Conclusions

In this paper, we propose two novel algorithms — the Spatial Division Playbook algorithm (SDPbA), and the Travelling Salesman Playbook Algorithm (TSPbA) — that are designed to perform better than existing decentralized task allocation algorithms in scenarios with very low communication availability. In our experiments, we use three communication models, and vary communication quality by defining five communication levels. We also vary the number of targets and compare the performance of the proposed algorithm against three task allocation algorithms, ACBBA, DHBA and PIA on the basis of the time taken to visit all targets (MCT).

The experimental results show that SDPbA and TSPbA perform better on an average than ACBBA, DHBA and PIA at lower communication levels and at number of targets greater than 30. Among the two playbook algorithms, TSPbA performs better than SDPbA in all cases except for when the number of targets are 10. This trend reverses and SDPbA performs worse than the other three algorithms for all values of the number of targets when the Bernoulli parameter is 0.0005 (Bernoulli model) and when the G-G transition probability is 0.01 (Gilbert-Elliot model). Also, SDPbA and TSPbA consistently exhibit the least standard deviation in MCT of all five algorithms, for all communication levels in all communication

models, and all values of the number of targets except for when the Bernoulli parameter is 0.0002 and 0.0005 when the number of targets is 10. Within SDPbA and TSPbA, no algorithm has a clear advantage in terms of standard deviation. This implies that the performance of both SDPbA and TSPbA for the communication levels under consideration is much more predictable as compared to the other three algorithms. All these inferences imply that the playbook algorithms are better options than existing algorithms when the number of targets is greater than 30 and when communication signal is very weak or when the mission requires agents to refrain from communicating. TSPbA exhibits the best performance in terms of MCT in the aforementioned cases.

The communication model being chosen also has an impact on the performance of all algorithms. The Rayleigh fading model takes into account path loss, which results in more variation in the MCT for all algorithms across all communication levels.

Future work in this direction may include variations on the proposed Playbook algorithm or a detailed study on the behavior of agents running the playbook algorithms as other simulation parameters are varied. Another direction of study is the extension of the playbook idea to other multi-agent problems. A possible use of the playbook approach is to have the agents compute paths using the playbook approach, and use these paths as a contingency plan if and when communication fails.

Bibliography

[1] S. Waharte and N. Trigoni, "Supporting search and rescue operations with uavs," in 2010 International Conference on Emerging Security Technologies. IEEE, 2010, pp. 142–147.

[2] Barrientos, A., Colorado, J., Cerro, J.d., Martinez, A., Rossi, C., Sanz, D. and Valente, J. (2011), Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. J. Field Robotics, 28: 667-689.

[3] N. Nigam, S. Bieniawski, I. Kroo, and J. Vian, "Control of multiple uavs for persistent surveillance: algorithm and flight test results," IEEE Transactions on Control Systems Technology, vol. 20, no. 5, pp. 1236– 1251, 2011.

[4] J. A. Shaffer, E. Carrillo, and H. Xu, "Hierarchal application of receding horizon synthesis and dynamic allocation for uavs fighting fires," IEEE Access, vol. 6, pp. 78 868–78 880, 2018.

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

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

[7] A. A. Khuwaja, Y. Chen, N. Zhao, M.-S. Alouini, and P. Dobbins, "A survey of channel modeling for uav communications," IEEE Communications Surveys and Tutorials, vol. 20, no. 4, pp. 2804–2821, 2018.

[8] S. Nayak, S. Yeotikar, E. Carrillo, E. Rudnick-Cohen, M.K.M. Jaffar, R. Patel, S. Azarm, J.W. Herrmann, H. Xu and M. Otte, "Experimental Comparison of Decentralized Task Allocation Algorithms Under Imperfect Communication," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 572-579, April 2020.

[9] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," IEEE transactions on robotics, vol. 25, no. 4, pp. 912–926, 2009.

[10] L. Johnson, S. Ponda, H.-L. Choi, and J. P. How, "Asynchronous decentralized task allocation for dynamic environments," in Infotech@ Aerospace 2011, 2011, p. 1441.

[11] L. Johnson, S. Ponda, H.-L. Choi, and J. How, "Improving the efficiency of a decentralized tasking algorithm for uav teams with asynchronous communications," in AIAA Guidance, Navigation, and Control Conference, 2010, p. 8421.

[12] S. Ismail and L. Sun, "Decentralized hungarian-based approach for fast and scalable task allocation," in 2017 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2017, pp. 23–28.

[13] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," Autonomous Robots, pp. 1–38, 2019.

[14] J. Wang, Y. Gu, and X. Li, "Multi-robot task allocation based on ant colony algorithm," Journal of Computers, vol. 7, no. 9, pp. 2160–2167, 2012.

[15] M. Dorigo and M. Birattari, Ant colony optimization. Springer, 2010.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y Ng, *ROS: an open-source Robot Operating System*, ICRA workshop on open source software, vol. 3, no. 3.2.

[17] D. Ruppert, Statistics and data analysis for financial engineering. Springer, 2011, vol. 13.

[18] A. Koubâa, Robot Operating System (ROS). Springer, 2017.

[19] M. Rantanen, N. Mastronarde, J. Hudack and K. Dantu, "Decentralized Task Allocation in Lossy Networks: A Simulation Study," 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), Boston, MA, USA, 2019, pp. 1-9.

[20] P. B. Sujit, A. Sinha and D. Ghose, "Multi-UAV Task Allocation using Team Theory," Proceedings of the 44th IEEE Conference on Decision and Control, Seville, Spain, 2005, pp. 1497-1502.

[21] W. Zhao, Q. Meng, and P. W. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," IEEE transactions on cybernetics, vol. 46, no. 4, pp. 902–915, 2015.

[22] L. Johnson, H.-L. Choi, and J. P. How, "The hybrid information and plan consensus algorithm with imperfect situational awareness," in Distributed Autonomous Robotic Systems. Springer, 2016, pp. 221–233.

[23] Alighanbari, Mehdi and How, Jonathan. (2006). Robust Decentralized Task Assignment for Cooperative UAVs.

[24] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Report 388, Graduate School of Industrial Administration, CMU, 1976.

[25] J.W. Herrmann, *Data-driven Metareasoning for Collaborative Autonomous Systems*, Institute for Systems Research Technical Reports, University of Maryland.

[26] Beardwood, J., Halton, J., Hammersley, J. (1959). The shortest path through many points. Mathematical Proceedings of the Cambridge Philosophical Society, 55(4), 299-327.