Received 10 November 2022, accepted 18 November 2022, date of publication 23 November 2022,
date of current version 30 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3224146

RESEARCH ARTICLE

Distributed Task Allocation Algorithms for Multi-Agent Systems With Very Low Communication

AKSHAY BAPAT, BHARATH REDDY BORA, JEFFREY W. HERRMANN, SHAPOUR AZARM, HUAN XU, (Member, IEEE), AND MICHAEL W. OTTE, (Member, IEEE)
A. James Clark School of Engineering, University of Maryland, College Park, MD 20742, USA

Corresponding author: Michael W. Otte (otte@umd.edu)

This work was supported in part by the Air Force Research Laboratory under Award FA8750-18-2-0114, and in part by the U.S. Office of Naval Research under Award N00014-20-1-2712.

ABSTRACT In this paper we explore the problem of task allocation when communication is very low, e.g., when the probability of a successful message between agents is $\ll$0.01. Such situations may occur when agents choose not to communicate for reasons of stealth or when agent-to-agent communication is actively jammed by an adversary. In such cases, agents may need to divide tasks without knowing the locations of each other. Given the assumption that agents know the total number of agents in the workspace, we investigate solutions that ensure all tasks are eventually completed—even if some of the agents are destroyed. We present two task allocation algorithms that assume communication may not happen, but that benefit whenever communications are successful. (1) The Spatial Division Playbook Algorithm divides task among agents based on an area decomposition. (2) The Traveling Salesman Playbook Algorithm considers mission travel distance by leveraging Christofides' 3/2 approximation algorithm. These algorithms have task completion runtime complexity of $O(m \log m)$ and $O(m^3)$, respectively, where m refers to the total number of tasks. We compare both algorithms to four state-of-the-art task allocation algorithms — ACBBA, DHBA, PIA and GA — across multiple communication levels and multiple numbers of targets, and using three different communication models. The new algorithms perform favorably, in terms of the time required to ensure all targets are visited, in the special case when communication is very low.

INDEX TERMS Autonomous robots, distributed, low communication environment, motion planning, target search, task allocation.

I. INTRODUCTION

Distributed task allocation involves autonomous agents dividing a set of tasks among themselves. Task allocation within autonomous multi-agent systems has been proposed for use in a variety of applications, including: search and rescue [1], agriculture [2], surveillance [3] and firefighting [4].

This paper studies decentralized task allocation algorithms for the special case in which communication is very low; when the probability of a successful message transmission is $\ll$0.01. Scenarios with very low communication may occur when agents choose not to communicate for reasons of stealth (in which case we assume that task locations may be broadcast from a distant transmitter without jeopardizing agent stealth), or when communication between agents is actively jammed by an adversary (for example, after agents know task locations but before tasks have been allocated). These scenarios are depicted in Figure 1. In such scenarios agents must be able to divide tasks *without* communication and without knowledge of each other's locations.

We present two new algorithms for distributed task allocation in multi-agent systems designed for use when communication is very low—or even, possibly, nonexistent. We call our proposed algorithms "playbook" algorithms because they are similar to a sports team's playbook. A playbook contains rules for each player, such that each player knows

The associate editor coordinating the review of this manuscript and approving it for publication was Qiang Li.

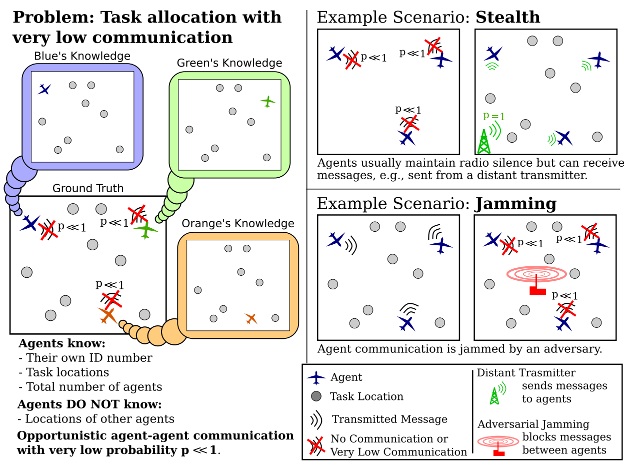
Problem: Task allocation with very low communication
Blue's Knowledge
Green's Knowledge
Ground Truth
Orange's Knowledge
p≪1
Agents know:
- Their own ID number
- Task locations
- Total number of agents
Agents DO NOT know:
- Locations of other agents
Opportunistic agent-agent communication with very low probability p ≪ 1.
Example Scenario: Stealth
p=1
Agents usually maintain radio silence but can receive messages, e.g., sent from a distant transmitter.
Example Scenario: Jamming
Agent communication is jammed by an adversary.
Agent
Task Location
Transmitted Message
No Communication or Very Low Communication
Distant Trasmitter sends messages to agents
Adversarial Jamming blocks messages between agents

FIGURE 1. Depiction of the Problem that we explore in this paper, task allocation with very low communication (left). Two example scenarios to which this problem is relevant (right) are task allocation to stealth agents already in the field (right-top) and task allocation when communication is jammed by an adversary (right-bottom).

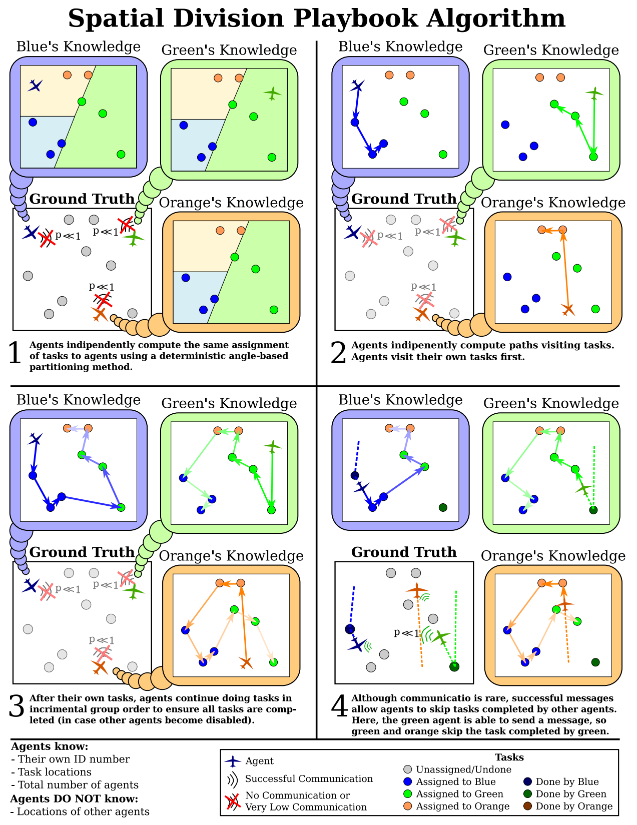
Spatial Division Playbook Algorithm
Blue's Knowledge
Green's Knowledge
Ground Truth
Orange's Knowledge
p≪1
1 Agents independently compute the same assignment of tasks to agents using a deterministic angle-based partitioning method.
2 Agents indipenently compute paths visiting tasks. Agents visit their own tasks first.
3 After their own tasks, agents continue doing tasks in incremental group order to ensure all tasks are completed (in case other agents become disabled).
4 Although communication is rare, successful messages allow agents to skip tasks completed by other agents. Here, the green agent is able to send a message, so green and orange skip the task completed by green.
Agents know:
- Their own ID number
- Task locations
- Total number of agents
Agents DO NOT know:
- Locations of other agents
Agent
Successful Communication
No Communication or Very Low Communication
Tasks
Unassigned/Undone
Assigned to Blue
Assigned to Green
Assigned to Orange
Done by Blue
Done by Green
Done by Orange

FIGURE 2. Depiction of the spatial division playbook algorithm (SDPbA). Note that in step 4 all agents have moved along their respective paths.

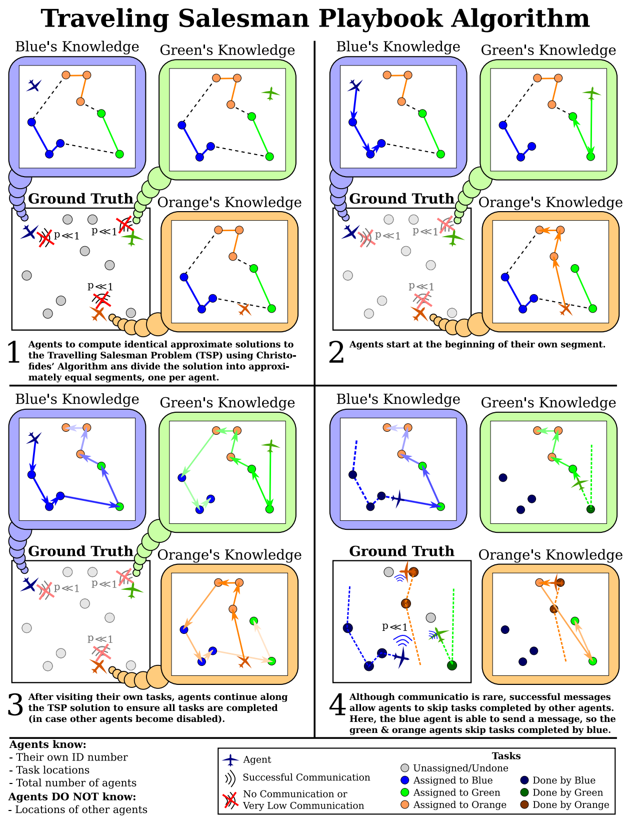
Traveling Salesman Playbook Algorithm
Blue's Knowledge
Green's Knowledge
Ground Truth
Orange's Knowledge
p≪1
1 Agents to compute identical approximate solutions to the Travelling Salesman Problem (TSP) using Christofides' Algorithm ans divide the solution into approximately equal segments, one per agent.
2 Agents start at the beginning of their own segment.
3 After visiting their own tasks, agents continue along the TSP solution to ensure all tasks are completed (in case other agents become disabled).
4 Although communicatio is rare, successful messages allow agents to skip tasks completed by other agents. Here, the blue agent is able to send a message, so the green & orange agents skip tasks completed by blue.
Agents know:
- Their own ID number
- Task locations
- Total number of agents
Agents DO NOT know:
- Locations of other agents
Agent
Successful Communication
No Communication or Very Low Communication
Tasks
Unassigned/Undone
Assigned to Blue
Assigned to Green
Assigned to Orange
Done by Blue
Done by Green
Done by Orange

FIGURE 3. Depiction of the traveling salesman playbook algorithm (TSPbA).

their own part of a coordinated team effort without the need for communication at runtime. That said, in the algorithms we present, *successful* runtime communications tend to *improve* solution quality whenever they occur.

The two playbook algorithms that we study are called: (1) The Spatial Division Playbook Algorithm (SDPbA), and (2) The Traveling Salesman Playbook Algorithm (TSPbA). The algorithms are depicted in Figure 2 and 3, respectively. Both algorithms involve a deterministic task allocation process that generates the same task assignments when run in independently and in parallel by each agent. The two algorithms differ in the deterministic mechanism that is used to allocate tasks to agents as a function of task locations and total agent number.

SDPbA uses an area-based procedure to divide the workspace into distinct regions, such that all regions contain a similar numbers of tasks (see Figure 2). Regions are allocated to agents based on agent ID number. In contrast, TSPbA calculates a single path through all the task locations using Christofides' deterministic 3/2 approximation to the Traveling Salesman Playbook Algorithm (TSP), and then then allocates contiguous segments of the TSP approximation to agents based on agent ID number (see Figure 3). TSPbA is computationally more expensive than SDPbA, but provides better load-balancing by allocating pieces of near-equal lengths to agents.

After completing their own tasks, all agents check that all tasks have been completed (this is done in case other agents have been destroyed or delayed). Any undone task is performed by the discovering agent. In both algorithms each agent i progresses through groups of tasks assigned to agents with higher and higher IDs, $i+1, \dots n, 1, \dots i-1$, with a wrap around occurring at $n \equiv 0$. Here n is the total number of agents. Successful communications, although very rare, are used to propagate knowledge of completed tasks. Agents do not need to verify tasks that are known to have been completed. Thus, successful communications tend to decrease mission time.

A. BACKGROUND

Task allocation algorithm are often categorized as being centralized or decentralized. Centralized algorithms only require communication to and from a central authority. While efficient, this also introduces a single point of failure, since the central authority has to communicate tasks to all of the agents. Decentralized algorithms are not susceptible to single points of failure because the computational and decision-making power is distributed among all agents. However, decentralized algorithms require all agents to think and act by themselves, which often requires duplication of computational effort by multiple agents.

Previous work has investigated how multi-robot decentralized task allocation algorithms that were originally designed for perfect communication perform as communication quality degrades [5], [6]. These investigations have shown that previously proposed algorithms perform well when agents are able to communicate, but their task allocation becomes inefficient when communication quality degrades. In contrast, we take the opposite approach. We present two new multi-agent task algorithms that have been designed for the special case of no communication, and then observe how their performance changes as communication quality *improves*.

B. MOTIVATION

This paper is concerned with studying distributive task allocation algorithms in scenarios in which communication is particularly limited. Examples of such communication limited scenarios include:

- When signal jamming prevents communication.
- When the environment is so large that there is considerable loss in communication signal.
- When the mission requires the agents to operate silently, e.g. when a message can give away the location of an agent to an adversary.
- When hardware malfunction results in agents being unable to send or receive messages.

SDPbA and TSPbA offer respective trade offs between using a simple and computationally efficient approach and using a more sophisticated but more expensive approach. If there are m tasks and n agents, SDPbA has a runtime complexity of $O(m \log m)$ for task allocation and $O(\frac{m^3}{n^3})$ for path-planning within the allocated tasks. Thus, the overall runtime complexity for SDPbA is $O(m \log m + \frac{m^3}{n^3})$. On the other hand, the runtime complexity of TSPbA for task allocation is $O(m^3)$ and chopping the segment into pieces takes $O(m)$. Thus, the overall runtime complexity for TSPbA is $O(m^3)$. If the number of targets is small, TSPbA is only slightly computationally more expensive than SDPbA, but can allocate tasks efficiently. On the other hand, if there are a large number of targets, TSPbA may generate a better solution but is significantly more computationally expensive than SDPbA.

C. CONTRIBUTIONS

Many real-world scenarios exist that may involve bad communication—for example, jamming, distance, stealth, or communication malfunctions. Having algorithms that can handle these conditions is useful. **The main contributions of this paper include: (1) Two new playbook algorithms designed for scenarios with very low communication between agents, (2) mathematical analysis of the playbook algorithms' transient startup phase, and (3) experimental simulations comparing the new algorithms with state-of-the-art task allocation algorithms (ACBBA [7], [8], DHBA [9], PIA [10] and GA [11]), using three communication models and two metrics across a variety of scenarios.**

We compare the Playbook Algorithms with state-of-the-art algorithms via simulations. In particular, we consider task allocation scenarios that involve visiting stationary targets, and study how performance changes across a variety of communication quality levels and target numbers m, while keeping number of agents n constant. Communication is modeled in three different ways, using, respectively: the Bernoulli model, the Gilbert-Elliot model, and the Rayleigh Fading model. We focus on the cases when the instantaneous probability (p) of a message being successfully delivered from one agent to another satisfies $p \ll 0.01$. We define two performance metrics for comparing algorithms: the Real Mission Completion Time (R-MCT) and the Agent Mission Completion Time (A-MCT). These are formally introduced in Section III.

The playbook algorithms' initial task allocation and visit sequencing are deterministic. One consequence of this is that they may start "slower" than other algorithms in terms of the rate of visiting targets per unit time. The target visit rate increases as time passes. We term this phenomenon the "startup cost" of the algorithms, and prove that the relative startup cost of both algorithms asymptotically converges to zero as the number of targets approaches infinity. We also discuss the runtime complexity and message size complexity of all six algorithms in the analysis section.

D. ORGANIZATION

The rest of the paper is organized as follows: Section II outlines existing related work. In Section III we present preliminary concepts and assumptions, and provide a formal problem statement. Section IV contains details about the Playbook Algorithm, and the other task allocation algorithms we use for comparison. Section V contains an analysis of algorithmic properties. Section VI describes the experimental setup used and presents a detailed mathematical analysis of the results. In Section VII we discus the implications of results obtained from the simulations. Section VIII derives conclusions based on these results, and also discusses possible future directions of work in this area.

II. RELATED WORK

A. RELATED WORK ON TASK ALLOCATION

Approaches for solving the multi-agent task allocation problem have appeared in several papers. A common approach is to perform 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) [12] are widely used decentralized algorithms based on the auction approach. The Asynchronous Consensus-Based Bundling Algorithm (ACBBA) [7], [8], Performance Impact Algorithm (PIA) [10] and the Hybrid Information and Plan Consensus algorithm (HIPC) [13] 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) [9] employs deterministic optimization using the Hungarian method, while the ant-colony optimization algorithm [14] uses stochastic optimization techniques. Patel et al. [15] propose an approach to task allocation based on a decentralized Genetic Algorithm [11] and demonstrate instances of when this approach works better than existing approaches.

Samiei et al. [16] propose a cluster-based Hungarian algorithm. This approach is similar to the approach proposed in this paper in that agents group tasks and solve the Hungarian assignment problem corresponding to these groups of tasks. One difference between our work and [16] is that our playbook algorithms use a deterministic procedure to assign groups of tasks to agents, while the cluster-based Hungarian algorithm in [16] relies on communication between agents for task assignment. In low communication scenarios, there is a chance that multiple agents are assigned the *same* group of tasks in [16], this cannot happen in SDPbA or TSPbA. Note that although we use SDPbA and TSPbAin scenarios requiring agents to verify tasks are completed by other agents, if the method of [16] was used in an analogous way, then two or more agents could potentially perform their tasks in the exact same order.

Best et al. [17] propose a novel approach to multi-agent planning using a variant of Monte-Carlo Tree Search (MCTS). This approach employs the MCTS to search the environment such that a global objective function is maximized subject to a constraint defined by the resource budget available to the agents. In contrast, we seek to minimize the cost (time metrics) without imposing any resource constraints on the agents.

B. RELATED WORK WITH IMPERFECT COMMUNICATION

The problem of imperfect communication in multi-agent task allocation has previously been studied in a variety of scenarios. Otte et al. [6] 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. [18] discuss the effect of a realistic communication model on the performance of ACBBA using an open-source network simulator. The results of [18] indicate that even when the communication is near-perfect, the algorithm becomes very inefficient due to redundant task assignments. Alighanbari et al. [19] 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. Radak et al. [21] compare the performance of two distributed control algorithms subject to three communication models, namely Bernoulli, Slotted Aloha and IEEE 802.11p. The main difference between our work and these previous efforts is our focus on the very low communication case. Other differences include our use of decentralized algorithms, the communication models used, and the performance metrics considered.

C. RELATIONSHIP TO AUTHORS' PREVIOUS WORK

Nayak et al. [5] 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. The current paper 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. In other words, while [5] considers p for $0.01 < p \leq 1$ the current paper considers $0 \leq p \ll 0.01$–and introduces two new algorithms designed for this particular case of very low communication.

Herrmann [22] 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. Carrillo et al. [23] explores using a formal logic based policy for metareasoning about which task allocation algorithm should be used in environments in which communication quality evolves over time. The current paper does not consider metareasoning, but provides additional tools that can potentially be used by a metareasoning task allocation system (for example, when communication degrades to such an extent that agents are rarely able to communicate).

The work in the current paper differs from our previous work [5], [22], [23] in a number of ways. Most importantly, the current paper focuses on new algorithms for the special case where communication is very low—$p \ll 0.01$ and potentially nonexistent—and does not consider metareasoning.

A preliminary version of this paper appeared in the first Author's Master's Thesis [24]. The current paper extends the Master's Thesis with a more detailed presentation of the playbook algorithms, additional experimental trials in simulation, statistical analysis of the algorithms' performance, new figures that depict the algorithms' behavior and performance, and a new discussion of results based on the new experimental trials.

III. PRELIMINARIES

In this section we define our nomenclature, discuss the metrics we use to evaluate algorithmic performance, and discuss our major assumptions. We also provide a formal problem definition for the task allocation problem that we study and discuss the communication models.

A. NOMENCLATURE

Consider a set of autonomous agents, denoted by A. A consists of n distinct agents $A = \{a_1, \ldots, a_n\}$ and a set of tasks, denoted by T, consists of m distinct tasks $T = \{t_1, \ldots, t_m\}$. The set of tasks assigned to agent a_i is denoted by S_i, where S_i is a finite set of tasks $S_i \subseteq T$. In this paper, 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.

B. PERFORMANCE METRICS

We evaluate the algorithms using two performance metrics:

- **R-MCT**: Real Mission Completion Time, defined as the time when all tasks have been performed at least once. This is an absolute measure of the time taken to complete the mission from an observer's point of view.
- **A-MCT**: Agent Mission Completion Time, defined as the time when at least one agent knows that all tasks have been completed.

The main difference between R-MCT and A-MCT is that R-MCT measures the time at which all tasks have been completed, while A-MCT measures the time at which at least one agent *knows* that all tasks have been completed. The two metrics are different in scenarios where agents have imperfect runtime knowledge of task completion. R-MCT is more relevant to scenarios where the time of task completion is more important than the delivery of this knowledge to, e.g. a human commander. On the other hand, A-MCT is more relevant to scenarios when knowledge of mission completion, e.g., by a human, is an important consideration.

C. MAJOR ASSUMPTIONS

We assume that tasks involve visiting targets located in the environment. Each agent knows its own unique ID number and the locations of all targets. We assume that agents do ***not*** know the locations of other agents. We assume agents travel at a constant speed, and that each agent has enough fuel to complete all tasks. For *SDPbA* we also assume that agents have a deterministic way to calculate the center of the workspace (such as at the centroid of task locations). The multi-agent system is "decentralized", implying that there is no central computing element that assists the agents and so agents are assumed to be capable of performing independent computation.

Agent knowledge of task locations is arguably our most restrictive assumption. In practice, agents' knowledge of the task locations may happen in a number of ways. For example, agents may receive a "one-way" message, e.g., from a distant transmitter or satellite. Agents may not be able to respond to this message due to requirements of stealth or because the communication channel is expected to be jammed after a single message is sent. Alternatively, agents may each be in the environment performing separate missions when an unanticipated event occurs that both prevents communication and necessitates the completion of tasks. For example, visiting the last known locations of human personnel for safety verification, rescue, or supply delivery after a disaster event.

We assume that agent-to-agent communication is very low. There are many reasons why the communication level may be very low. For example, the existence of fluctuating environmental factors such as jamming or naturally occurring forms of electromagnetic interference (solar flares, etc.). Alternatively, if stealth is the primary factor for very low communication then communication may be very low for at least two different reasons. Agents may choose to communicate very infrequently and in a stochastic manner to minimize information observable by an adversary. Alternatively, if agents are equipped with a means of determining when it is "safe" to communicate, then the set of circumstances that permit "safe" communication may only occur infrequently and/or at random.

D. 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 set 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 runtime after agents are already in the field, determine agent movement in order to minimize the mission completion times (R-MCT and A-MCT).

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

E. COMMUNICATION MODELS

Communication between agents can be modeled in a number of ways. In this paper, we compare the results obtained using three different communication models. These include:

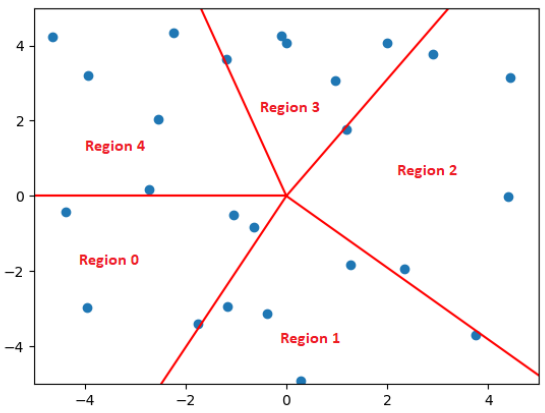

FIGURE 4. Map slicing as done by the spatial division playbook algorithm. Regions are calculated using a deterministic method such that each region contain either $\lfloor n/m \rfloor$ or $\lceil n/m \rceil$ tasks.

the Bernoulli Mode (a stochastic model), the Gilbert-Elliot Model (a Markov model), and the Rayleigh Fading Model (a limiting case of communication reflections that accounts for distance fading and self interference). The details of all three communication models are presented in detail in Appendix.

All three of the aforementioned communication models have previously been used to study task allocation in scenarios with imperfect communication, e.g., by Otte et al. [6] and Nayak et al. [5]. Each model has different strengths and weaknesses, and so evaluating algorithmic performance using multiple models provides a more comprehensive understanding of an algorithms performance, e.g., with respect to the performance of other algorithms in a low communication setting.

IV. ALGORITHMS

In this section, we provide a detailed description of the two proposed Playbook algorithms as well as a discussion of caveats related to using the algorithms. The details of existing algorithms—ACBBA, PIA, DHBA, and GA—to which we compare our work appear in Appendix.

A. SPATIAL DIVISION PLAYBOOK ALGORITHM (SDPbA)

Using the Spatial Division Playbook Algorithm, agents distribute tasks amongst themselves such that each agent gets a set S_i of tasks and satisfies $\cap_{i=1}^{n} S_i = \phi$. These sets of tasks are mutually exclusive and collectively exhaustive. As shown in Figure 4, for a scenario with 25 targets (blue) and five agents, each agent running SDPbA divides the map into five regions (red lines mark the region boundaries). Each region contains five targets that are assigned to the agent corresponding to that region.

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

Algorithm 1 Spatial Division Playbook Algorithm

```
1: function SDPbA($T$, $n_a$, $ID$)
2:     $region, path \leftarrow None$
3:     $i \leftarrow 0$
4:     $targsLeft \leftarrow$ len($T$)
5:     $TPR \leftarrow \lceil targsLeft/n_a \rceil$
6:     $boundary \leftarrow -\pi/2$
7:     $targList \leftarrow$ sorted(T)
8:     while $targsLeft > 0$ do
9:         $region[i]$.insert(nextTargs($TPR$, $T$))
10:        $boundary \leftarrow$ lastAngle($region[i]$)
11:        $targsLeft \leftarrow targsLeft - 1$
12:        $i \leftarrow i + 1$
13:        $n_a \leftarrow n_a - 1$
14:        $TPR \leftarrow \lceil targsLeft/n_a \rceil$
15:    end while
16:    $path \leftarrow$ TSPPath($region[ID]$)
17:    appendRemainingTasks($path$, $region$, $ID$)
18:    while $missionNotComplete$ do
19:        sendCompletedTasksList()
20:        removeCompletedTasks($path$)
21:    end while
22: end function
```

from the center of the map to the edges. The algorithm starts by sorting targets based on their polar coordinate θ in the function sorted() (line 7 of Algorithm 1). In case of two or more targets having the same value of θ, targets are sorted based on their polar coordinate r, which is the distance from the origin.

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 nextTargs() sorts the targets according to their angles (polar coordinate θ) and returns TPR number of targets that lie in the slice corresponding to the current region. *boundary* is defined (line 6) as the angle at which the current region begins. Thus, for region 0, the boundary is $-\pi/2$. This value of *boundary* is updated every time a new region is being considered (line 10).

TPR is re-calculated for each region that is considered (line 14), using updated values of targets left and agents left for assignment (line 11, 13). 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 12). Using this approach, each agent computes the exact same *region* list. The algorithm returns an approximate Traveling Salesman Problem (TSP) solution corresponding to the region that has the same ID as

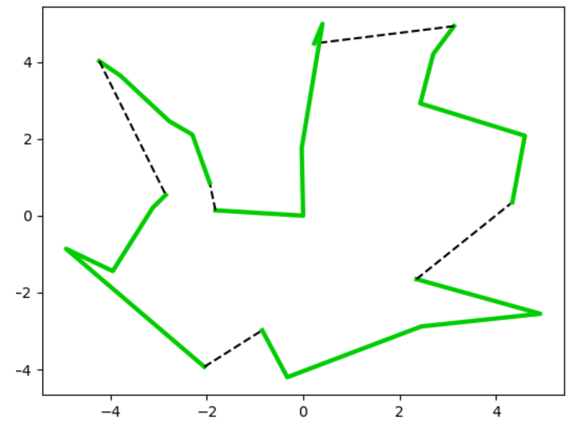

FIGURE 5. **Division of TSP solution as done by the traveling salesman playbook algorithm. Green segments denote pieces of the TSP solution that are assigned to agents.**

the agent (line 16), in the function `TSPPath`(). This approximate solution is computed using Christofides' algorithm [25], which guarantees a solution that is no worse than 1.5 times the optimal TSP solution in $O(m^3)$ time for visiting m cities. 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).

Since task allocation in SDPbA involves sorting the m tasks, its runtime complexity is $O(m \log m)$. After tasks are allocated to agents, the algorithm plans for the sequence with which the tasks should be completed. While a variety of methods can potentially be used to calculate each single agent's tour of its own tasks, we use Christofides' approximation of the TSP solution. This operation has a runtime complexity of $O(\frac{m^3}{n^3})$. Hence, the overall runtime complexity for SDPbA is $O(m \log m + \frac{m^3}{n^3})$

B. TRAVELING SALESMAN PLAYBOOK ALGORITHM (TSPbA)

The Traveling Salesman Playbook Algorithm programs the agents to compute identical approximate solutions to the Traveling Salesman Problem (TSP) where each target in the map is a node of the TSP. Each of the n agents computes an 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. The orders of tasks in pieces are determined based on their sequential order in the global TSP solution. To equalize the load distribution among the agents, the algorithm divides the path into pieces that minimizes $max(L)$, where L is the set $\{l_1, \ldots, l_n\}$.

An example is illustrated in Figure 5, where the algorithm computes a TSP-approximate path through 25 targets. Assuming five agents, this path is divided into five 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 black 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, and iteration number $iter$ (described below). 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 6 and 7) such that the maximum 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 11) 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 13). 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 14). These actions are repeated till the mission is complete, which is checked by the flag *missionNotComplete* (line 12).

Task allocation in TSPbA uses Christofides' algorithm, resulting in a runtime complexity of $O(m^3)$. Since tasks are allocated to agents in the same sequence as the global TSP solution, agents do not require further path planning as they can follow the same sequence. Hence, the total runtime complexity of TSPbA is $O(m^3)$.

C. 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

Algorithm 2 Traveling Salesman Playbook Algorithm

```
function TSPbA($T$, $n_a$, $ID$, $iter$)
    $pieces, partitions, edges, path \leftarrow None$
    $path, edges \leftarrow$ TSPPath($T$)
    $partitions \leftarrow$ getNaiveParts($edges, n_a$)
    while $iter$ do
        for $i$ in $partitions$ do
            $partitions \leftarrow$ minMax($partitions, i$)
        end for
        $iter = iter - 1$
    end while
    $pieces \leftarrow$ dividePath($path, partitions, ID$)
    while $missionNotComplete$ do
        sendCompletedTasksList()
        removeCompletedTasks($path$)
    end while
end function
```

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 unknown *a priori* (i.e., agents may happen to be anywhere when the search mission starts), 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 traveled 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 R-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.

V. ANALYSIS

The startup cost of agents results in a variability in the performance of the playbook algorithms, depending on the starting locations of agents and the division of targets that the agents come up with. On observing how the agents behave when running the playbook algorithms, we conjecture that as the number of targets increases, the startup cost gets smaller in comparison to the time required to visit targets assigned to that agent. In the Section V-A we prove this conjecture.

In Section V-B we analysis of the bandwidth requirements of the playbook algorithms and the other algorithms used in our experiments.

A. ANALYSIS OF STARTUP COST OF PLAYBOOK ALGORITHMS

We start by proving the aforementioned conjecture for TSPbA, before moving on to prove a similar result for SDPbA.

Definition 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, denoted by $\mathbf{W}$.

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, since all events share the same workspace. Thus, $\mathbf{X}_{n,m} = \mathbf{X}_n \times \mathbf{X}_m$.

Definition 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 3 (TSP Distance Function): The TSP distance function, $F\colon \mathbf{X}_{n,m} \to \mathbb{R}$, is defined as the distance (cost) of the optimal TSP solution to the m-city problem defined by event $\mathbf{x}_{n,m}$.

Beardwood et al. [26] state 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 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 1: In a square map of dimensions $M \times M$, the length of the optimal solution to the Traveling 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 1, 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. Since targets and agents can exist anywhere in the workspace, 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 for some finite, positive c, almost surely: $\lim_{m\to\infty} \frac{F(x_{n,m})}{\sqrt{mv}} = c$.

In the scenarios under consideration, the map is a square with area M^2. Hence, $\lim_{m\to\infty} \frac{F(x_{n,m})}{M\sqrt{m}} = c$. ∎

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

Proof: The startup cost (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 1: 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 (R-MCT or A-MCT) is proportional to the total distance traveled (L_{total}). The total distance is the sum of the startup cost and the cost of the solution to the TSP. Thus, $L_{total} = L + L_S \le L + L_{Smax}$. Hence, $L_{total} \le L + M\sqrt{2}$. The fraction of time spent in the first haul (f) is given by $f = \frac{L_S}{L_{total}}$. From Lemma 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 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 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 2 still holds true, making $L_{Smax} = M\sqrt{2}$. Thus, it follows from theorem 1 that $\lim_{m\to\infty} f = 0$. ■

B. BANDWIDTH REQUIREMENTS

Different task allocation algorithms have different bandwidth requirements in terms of message size and the frequency of sending messages to work effectively. In this subsection we discuss these requirements for all the algorithms being compared in this study.

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 $O(mn)$.

In each time step of GA, an agent generates a population of solutions, which is an $O(m)$ operation. It then selects the best solution and sends it, and in turn receives solutions from the other agents, then repopulates, and then mutates the solutions. The time complexity of the overall operation is hence $O(m + n)$. Since the messages sent are individual solutions, the message size is $O(m)$.

Agents running SDPbA or TSPbA only communicate completed tasks with each other. The computation of initial paths to visit targets happens at the beginning of the mission. During the mission, agents may (although, only very rarely) send out task completion messages to each other. When agents receive messages from other agents that indicate the completion of specific tasks, the receiving agent removes the completed tasks from their own path and then recomputes the tour of paths (in group order). In TSPbAthis is done by shortcutting the completed tasks, and in SDPbAthis can be done by recomputing tours of modified regions. 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 1. Note that the computational complexity referred to in this case is not the overall complexity of the algorithms, but only the complexity of the computations required when sending messages to other agents. This complexity dictates the frequency with which agents send and receive messages.

TABLE 1. Bandwidth requirements for algorithms.

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

VI. EXPERIMENTS

This section outlines the simulation framework used to run experiments and the rationale behind the design of experiments.

A. SIMULATION FRAMEWORK

Experiments were carried out using a simulator developed in ROS [27], [28] 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. We evaluate how performance is affected by different levels of communication quality. We perform three different sets of trials, each set using a different communication model: the Bernoulli model, the Gilbert-Elliot model, or the Rayleigh fading model. Agents exchange messages with each other based on the particular communication model used in each set of trials.

The comparison of performance trends across different communication models provides insight into how the way that communication is modeled affects the algorithms' relative performance. We believe that performance trends that are observed in all three models are more likely to generalize to other communication models outside the scope of our current work, e.g., as compared to trends that are only observed in one or two of the communication models we explore.

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.

B. DESIGN OF EXPERIMENTS

An experiment instance is defined by the number of targets, the target locations, the agent starting locations, target cluster locations and a communication level. In this paper we use a constant value of five agents, and other parameters are generated within constraints as given in Table 2. 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 3 and 4. Level C0 corresponds to the *least* communication availability and then communication availability increases with each level C1, C2, C3, and C4, respectively.

The new methods we evaluate are designed for the case of no communication, while existing methods have been designed for the case of perfect communication. Therefore, it seems reasonable to assume that as communication quality increases, existing methods will perform better and better—and are likely to outperform the playbook algorithm at sufficiently high levels of communication quality.

For each communication model, the lowest communication level (C0) corresponds to *no* communication, while the highest communication level C4 is chosen such that most of the comparison algorithms outperform at least one of the playbook algorithms (based on a preliminary exploration of the experiment space). C1, C2, and C3 are then chosen to provide increasing communication quality between C0 and C4. Thus, for each set of experiments (i.e., per communication model), the range of communication qualities has been selected to highlight the region in which the relative performance of the different algorithms is shifting.

TABLE 2. 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]

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 separately included in this study. Also, as a consequence, the final communication level is denoted as C5 for the Gilbert-Elliot model.

TABLE 3. Definition of communication levels: bernoulli model and rayleigh fading model.

Communication Level	Bernoulli Parameter	Rayleigh Sensitivity Threshold
C0	0.0	30
C1	0.00005	25
C2	0.0001	20
C3	0.0002	10
C4	0.0005	0

TABLE 4. 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

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 region of radius given by cluster radius, centered at the cluster center. A number of target clusters is chosen, followed by the cluster center locations and the cluster radii, all chosen randomly with the constraints provided in Table 2. Using these parameters, targets are then placed within the clusters based on a uniform distribution. Figure 6 illustrates an instance of random

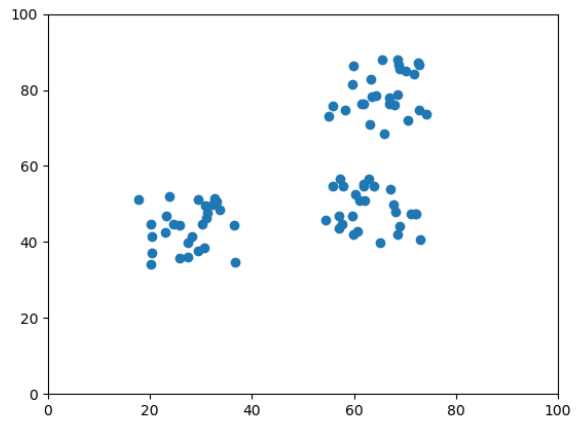

FIGURE 6. Illustration of random target generation in clusters.

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.

For a single task allocation algorithm, there are 90 possible combinations of communication model (3 types), communication level (5 values) and number of targets (6 values). For each combination, 40 random scenarios were 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 six task allocation algorithms were compared, a total of 21,600 experiments were conducted for this analysis. For each of these experiments, we recorded the performance of the task allocation algorithms in terms of R-MCT and A-MCT. Depending on the target locations, communication model and communication level, agents may come to know of the mission completion (the event that defines A-MCT) very late, thus delaying experiments and increasing the value of A-MCT by a large amount. Hence, we set a timeout of 100 seconds, so that any experiment running longer than 100 seconds was killed and the A-MCT was recorded as 100 seconds.

C. EXPERIMENTAL RESULTS

We performed a number of experiments to test our algorithms, and evaluate performance using the metrics R-MCT and A-MCT. A single data point represents the mean metric (R-MCT or A-MCT, in seconds) for an algorithm for 40 distinct experiments corresponding to a single combination of the parameters shown in Table 2. Standard deviation over the 40 experiments is shown using error bars. The 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.

A high-level summary of experimental results appears in Figures 7 and 8, which depict which algorithms had the best mean performance in different scenarios. Figure 7 summarized performance with respect to the R-MTC metric, while Figure 8 summarizes results with respect to the A-MTC metric. The new methods we propose have relatively good performance with respect to the R-MTC metric when there are many targets (e.g., over 30) and communication is very poor.

Figures 9-10 present an in-depth summary of mean, standard deviation, and statistical significance values with respect to the R-MTC metric. Likewise, Figures 12-13 present an in-depth summary of mean, standard deviation, and statistical significance values with respect to the A-MTC metric. The playbook algorithms have relatively small standard deviations, in general, compared to the other methods.

Considering the R-MTC metric, the difference in performance between the playbook algorithms and the other methods become more statistically significant in two cases. The first case is when communication is "very poor" and there is a large number of targets—**when the playbook algorithms tend to perform relatively well**. The second case is when communication is only "poor" (and not "very poor") and there is a small numbers of targets—when the playbook algorithms tend to perform relatively poorly.

For the A-MCT performance metric our experiments show that best performing algorithm appears to be related to the communication model that is used. This is an unexpected and interesting result in its own right. That said, we find that the playbook algorithm does not often perform the best the with respect to the A-MCT performance metric.

VII. DISCUSSION

In this section we discuss in more depth the results of our experiments, including: the overall performance of the playbook algorithms (Section VII-A), the effect of the number of targets (Section VII-B), the effect of the communication level and communication model (Section VII-C), and the standard deviation of the playbook algorithms (Section VII-D).

A. OVERALL PERFORMANCE OF THE PLAYBOOK ALGORITHMS

In general, the proposed playbook algorithms tend to perform well with respect to the R-MTC performance when metric when communication is very poor and there are more than 30 targets. The playbook algorithms do not perform as well when communication is not very poor, when there are fewer than 30 targets, or when considering A-MTC metric. In general, the TSPbA playbook variant tended to outperform the SDPbA playbook variant.

Our experiments indicate that the playbook algorithms may be useful in scenarios with many targets, when communication is very poor (or non existent) communication.

On the other hand, previously existing algorithms appear to be more useful in scenarios when the communication is *not* very poor, when target numbers are small (less than 30), or when we required that at least one agent can quickly discern that the mission has been completed.

The experiments presented in Section VI rely on communication models that account for communication variability

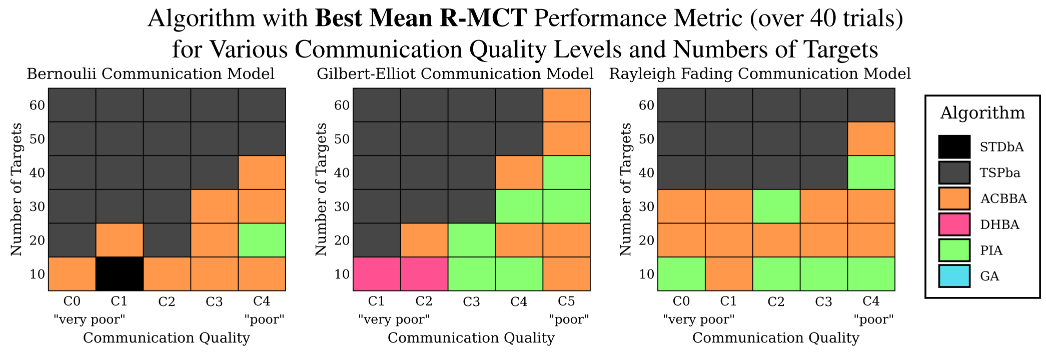

FIGURE 7. Summary of algorithm comparison for six different algorithms in different scenarios with respect to the R-MTC metric. 40 trials are run for each combination of target number and communication quality level, for three different communication models (sub-figures left to right). Five agents are used in all experiments. Each grid's color indicates which algorithm had the best mean performance for a particular combination of target number, communication quality level, and communication model. See Tables 3 and 4 for details about how communication levels map to communication model parameters. See Figures 9-10 for more in-depth plots of mean, standard deviation, and statistical significance values. Real Mission Completion Time (R-MTC) is the time when all targets have been visited at least once. The new methods we propose tend to perform relatively well with respect to R-MTC when communication is very poor and there are large numbers of targets.

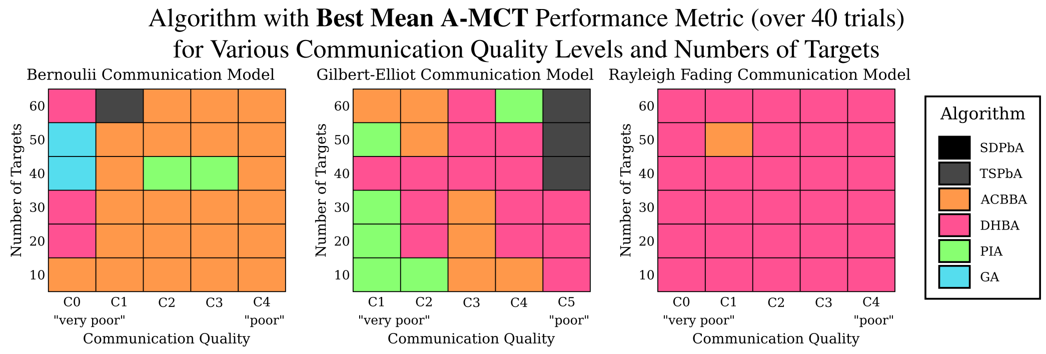

FIGURE 8. Summary of algorithm comparison for six different algorithms in different scenarios with respect to the R-MTC metric. 40 trials are run for each combination of target number and communication quality level, for three different communication models (sub-figures left to right). Five agents are used in all experiments. Each grid's color indicates which algorithm had the best mean performance for a particular combination of target number, communication quality level, and communication model. See Tables 3 and 4 for details about how communication levels map to communication model parameters. See Figures 12-13 for more in-depth plots of mean, standard deviation, and statistical significance values. Agent Mission Completion Time (A-MCT) is the first time when at least one agent becomes knowledgeable of the fact that all targets have been visited at least once. In contrast to results for R-MTC, the methods we propose do not appear to provide a relative advantage with respect to the A-MCT metric. Moreover, the communication model that is used appears to have an effect on which algorithm performs the best with respect to A-MCT.

based on randomness (Bernoulii and Gilbert-Elliot models) and/or motivated by wireless signal interference caused by reflections off of objects in the environment (Rayleigh Fading model). Considering systematic shifts in communication quality over time could be an interesting direction for future study (for example, decreasing average communication quality as a function of time). However, this is beyond the scope of our current work.

B. EFFECT OF NUMBER OF TARGETS

We note from our observations 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, PIA and GA. We hypothesize that this behavior is the result of the startup cost (as described in the Analysis section). The first haul of agents is depicted by a dotted black line in Figure 11. 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 the time required to visit all targets in its region. In such cases, the other four algorithms perform better than either of the playbook algorithms. As a result of theorem 1 and corollary 1, as the number of targets increases, SDPbA and TSPbA become favorable algorithms in comparison to the four existing task allocation algorithms.

C. EFFECT OF COMMUNICATION LEVEL AND COMMUNICATION MODEL

From the results, we observe the range of communication levels and number of targets in which the playbook algorithms perform better than existing task allocation algorithms.

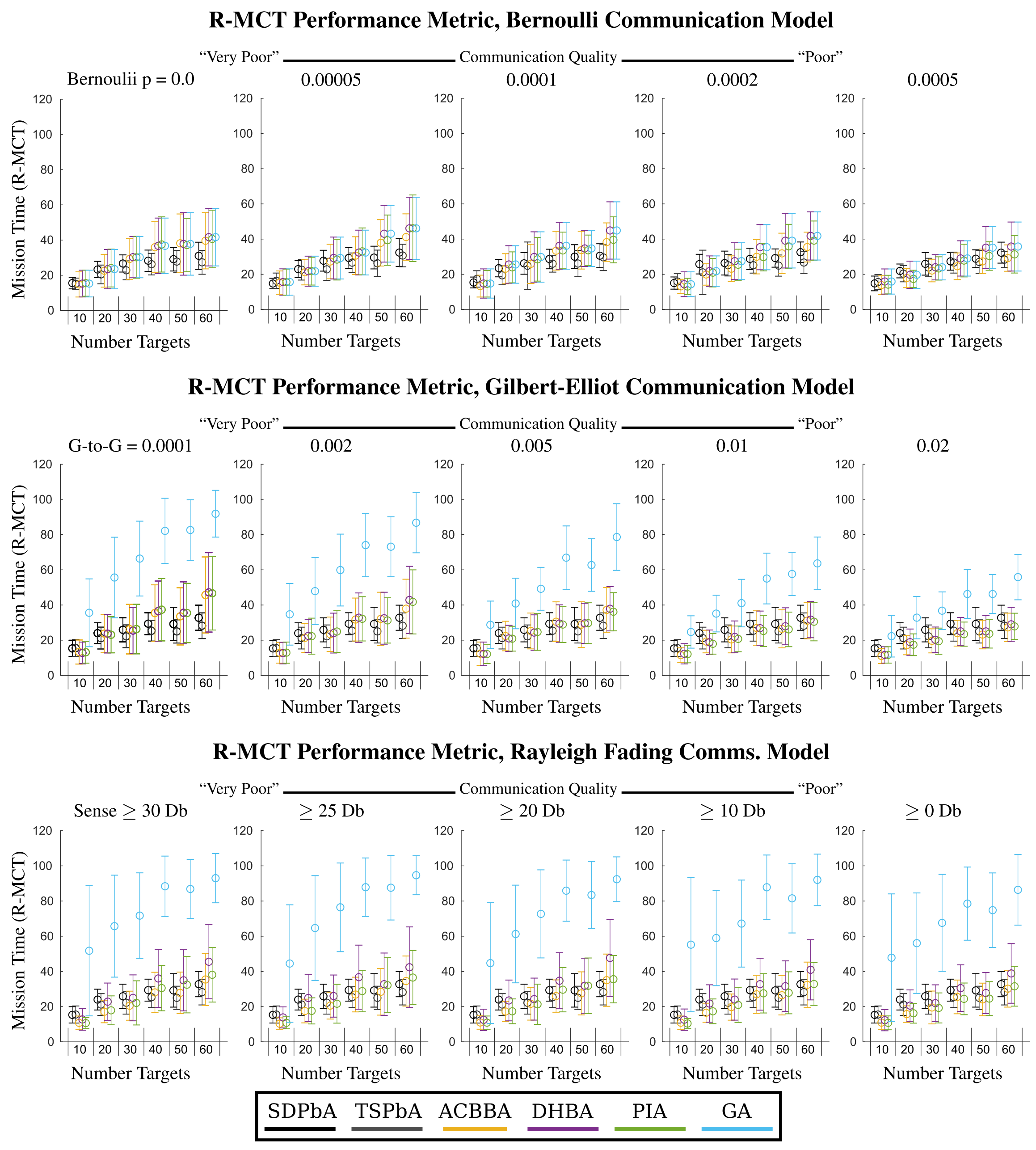

FIGURE 9. Mean (color circles) and standard deviation (color error bars) of performance with respect to the R-MTC metric. Six algorithms are compared in different scenarios. 40 trials are run for each combination of target number (x-axes) and communication quality level (sub-plots) for three different communication models (sub-figures left to right). Five agents are used in all experiments.

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.

D. SMALLER STANDARD DEVIATION IN METRICS

We observe that the standard deviation in R-MCT for the playbook algorithms is smaller than the standard deviation in R-MCT for existing task allocation algorithms.

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

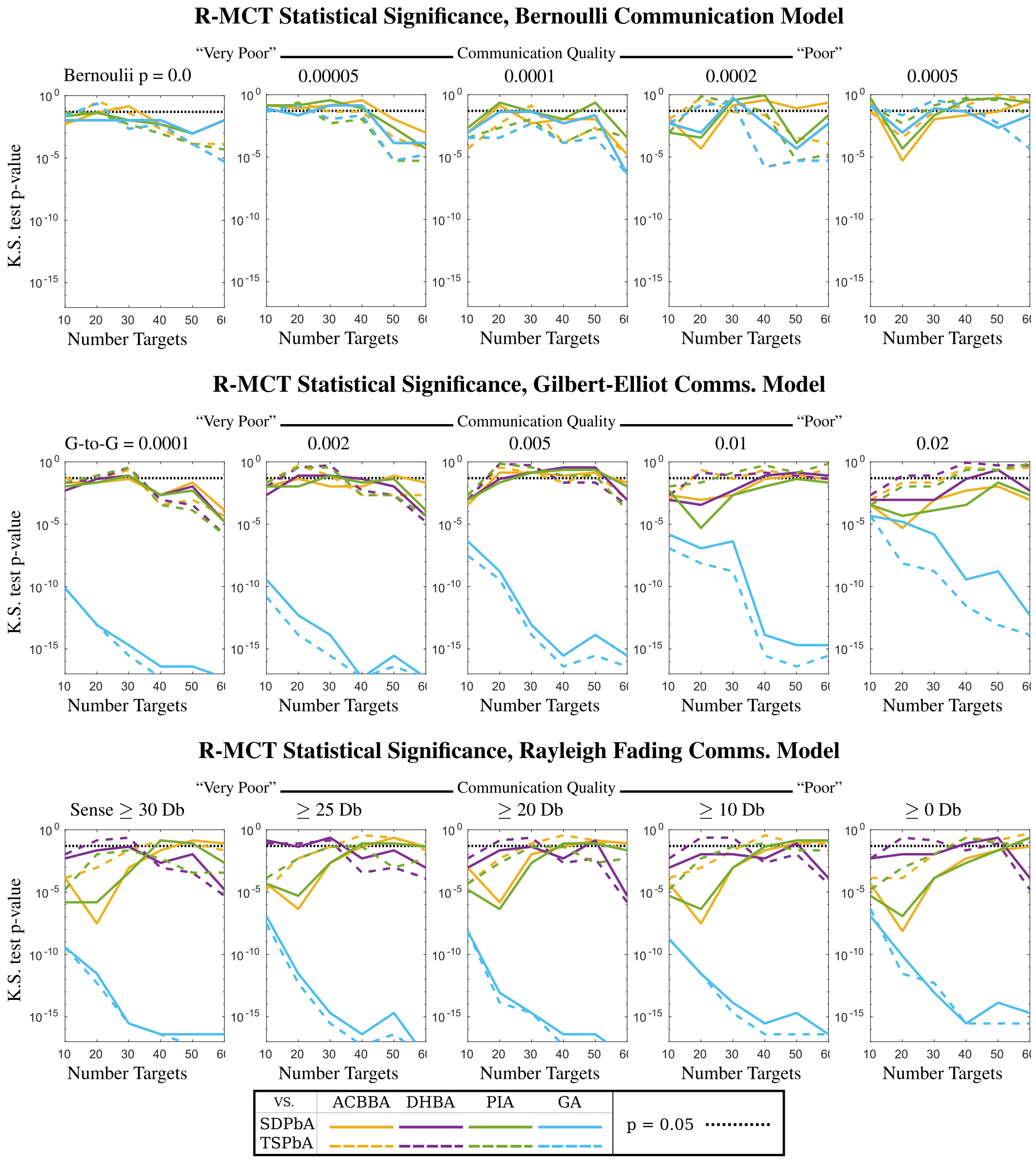

FIGURE 10. p-value results of Kolmogorov–Smirnov (K.S.) test for significance. Lower values indicate greater statistical significance. A comparison is preformed between each of the two playbook algorithms and each of the four comparison methods. Significance tests are based on 40 trials are run for each combination of target number (x-axes) and communication quality level (sub-plots) for three different communication models (sub-figures left to right). A p-value of 0.05 is also shown for comparison (dotted line).

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

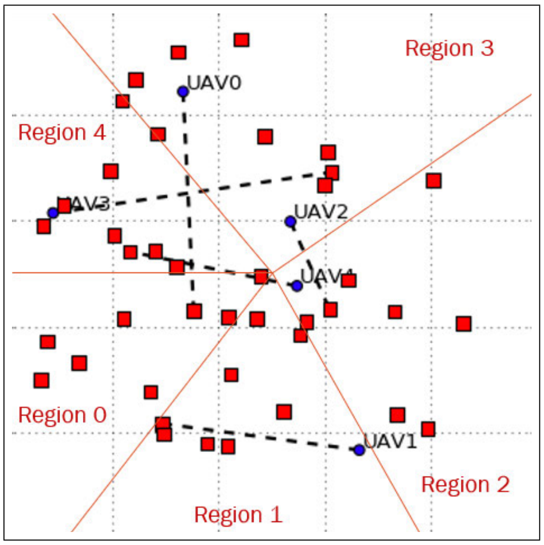

FIGURE 11. An instance of agents completing long first hauls to enter their respective regions.

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, PIA or GA. 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 R-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, PIA and GA, 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.

VIII. CONCLUSION

In this paper, we propose two new algorithms — the Spatial Division Playbook Algorithm (SDPbA), and the Traveling 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 four task allocation algorithms, ACBBA, DHBA, PIA and GA on the basis of two time-based metrics, R-MCT and A-MCT.

The experimental results show that SDPbA and TSPbA perform better on an average than ACBBA, DHBA, PIA and GA at lower communication levels and at number of targets greater than 30, especially with respect to average mission completion time (R-MTC). Among the two playbook algorithms, TSPbA performs better than SDPbA in all cases except for when the number of targets are 10.

Our experiments showed that the relative performance of algorithms was effected by the particular communication model was used. This effect was easier to see for the alternative A-MTC performance metric (where the best performing algorithm was highly correlated with the communication model) but it can also be seen with respect to the R-MTC performance metric by observing the ranking of non-best algorithms.

On the other hand, the favorable relative performance of the playbook algorithms with respect to the R-MTC did not appear to be impacted by the communication model being used. The fact that the playbook algorithms performed well in scenarios with very low communication and many targets *regardless of communication model* suggests that the playbook algorithms, and especially TSPbA, may have useful applications in a variety of low-communication settings.

APPENDIX COMMUNICATION MODELS USED IN OUR EXPERIMENTS

We now describe the three communication model used in our experiments, each in its own subsection.

A. 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. p is assumed to remain constant throughout a single simulation run. The communication attempts are assumed to be independent and identically distributed.

B. 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

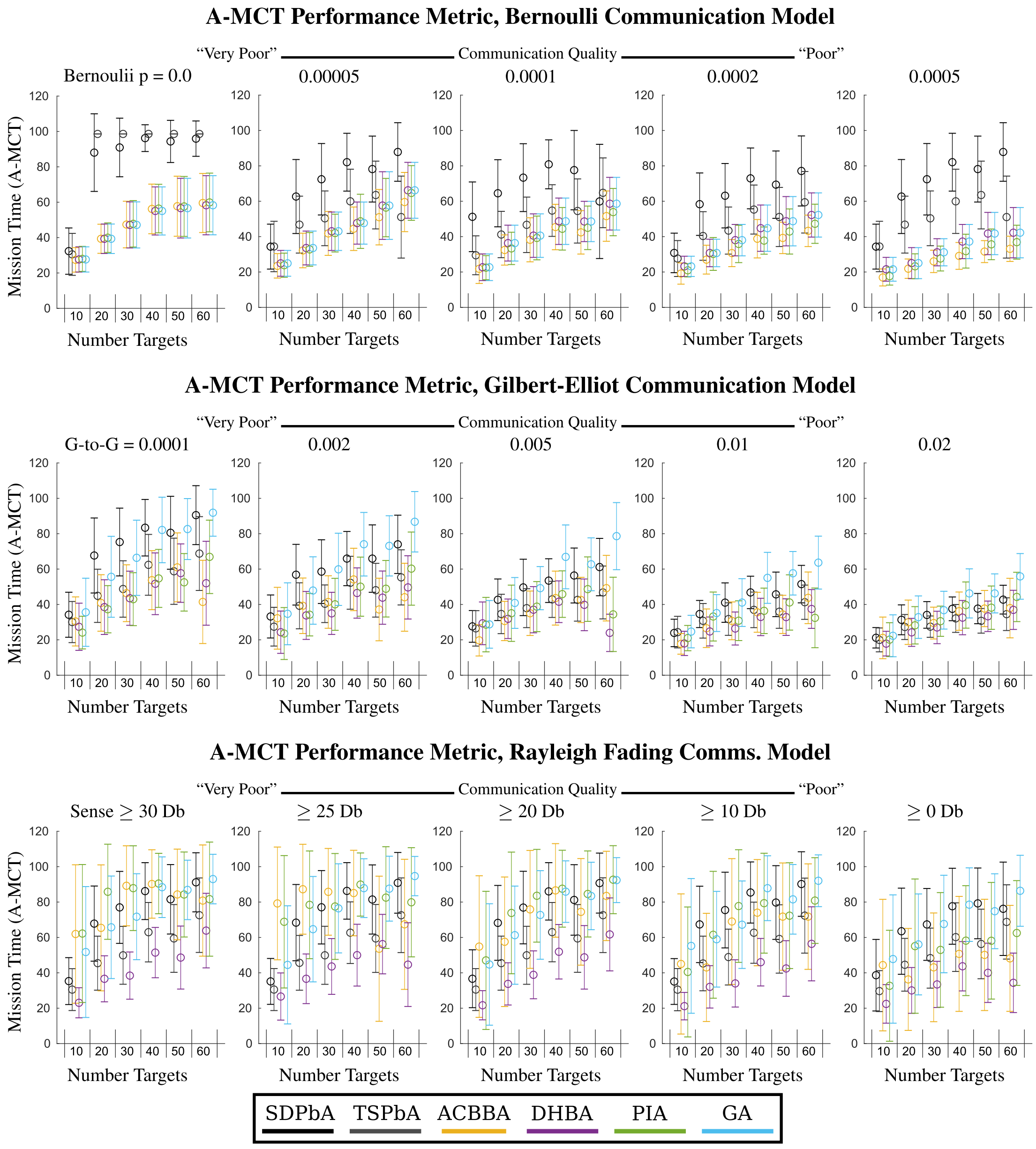

FIGURE 12. **Mean (color circles) and standard deviation (color error bars) of performance with respect to the A-MTC metric. Six algorithms are compared in different scenarios. 40 trials are run for each combination of target number (x-axes) and communication quality level (sub-plots) for three different communication models (sub-figures left to right). Five agents are used in all experiments.**

Bad (messages will be dropped). Figure 14 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 (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.

C. RAYLEIGH FADING MODEL

This model considers two effects that attenuate the signal strength: fading and path loss. Fading is the attenuation in

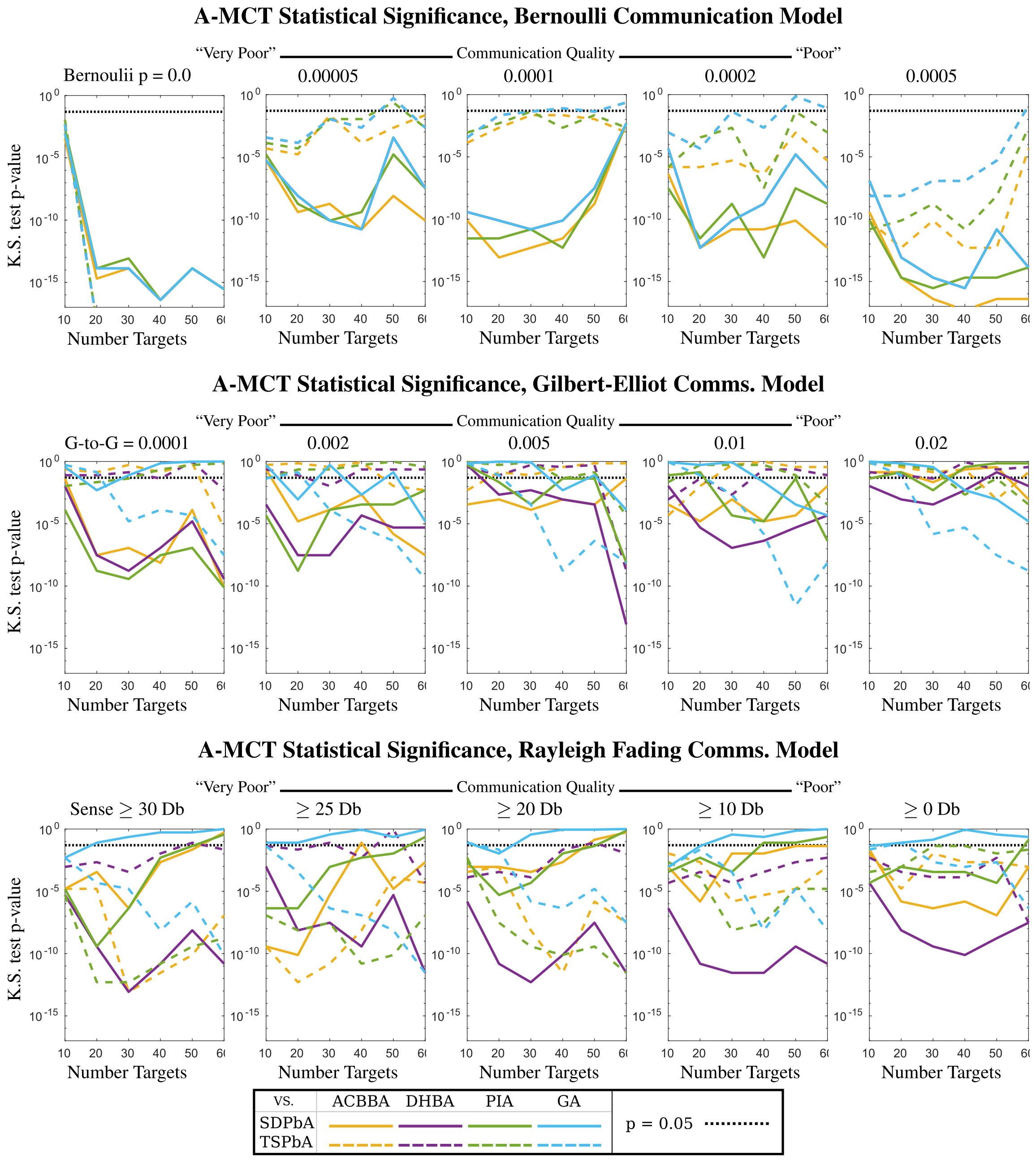

FIGURE 13. p-value results of Kolmogorov–Smirnov (K.S.) test for significance. Lower values indicate greater statistical significance. A comparison is preformed between each of the two playbook algorithms and each of the four comparison methods. Significance tests are based on 40 trials are run for each combination of target number (x-axes) and communication quality level (sub-plots) for three different communication models (sub-figures left to right). A p-value of 0.05 is also shown for comparison (dotted line).

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 of the channel response varies according to the Rayleigh distribution. Nayak et al. [5] 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

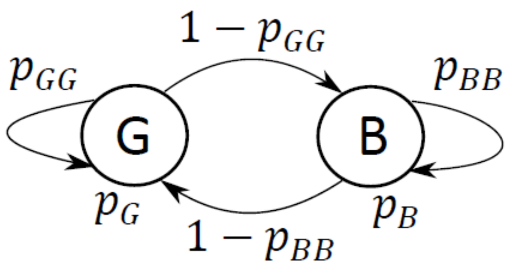

FIGURE 14. Communication channel in the Gilbert-Elliot modeled as a Markov chain (from Nayak et al. [5]).

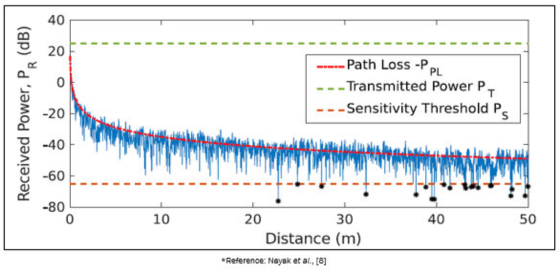

FIGURE 15. Received signal strength using the Rayleigh fading model. Dropped packets are marked with black dots. Note that chance a packet is dropped increases with distance.

packet is dropped. This drop in received signal strength due to path loss and fading is illustrated in Figure 15.

APPENDIX EXISTING TASK ALLOCATION ALGORITHMS USED FOR COMPARISON

The following four algorithms are used for comparison with the Playbook Algorithms: ACBBA, PIA, DHBA, and GA. These algorithms are chosen because they all have different approaches to task allocation, and these different approaches may or may not work better than the Playbook algorithm in the scenarios under consideration. ACBBA and PIA are auction-based algorithms derived from CBBA, and are shown to perform well when communication is imperfect [5]. DHBA uses the Hungarian Assignment algorithm to allocate tasks, while GA is a decentralized version of the Genetic Algorithm that mutates and preserves the best solutions while discarding the other solutions.

A. ASYNCHRONOUS CONSENSUS-BASED BUNDLING ALGORITHM (ACBBA)

This algorithm (Algorithm 3) is an improved version of CBBA. The input arguments are distance traveled (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.

Algorithm 3 ACBBA

```
1: function ACBBA($d_i, W_i, T, I, B$)
2:     $b_i \leftarrow None$
3:     for $k \leftarrow 1$ to $I$ do
4:         $(b_i, W_i) \leftarrow$ Assignment($b_i, d_i, W_i, T, B$)
5:         SendBids($W_i$)
6:         $(b_i, W_i) \leftarrow$ Consensus($b_i, W_i, B$)
7:     end for
8:     return $b_i$
9: end function
```

Algorithm 4 PIA

```
1: function PIA($d_i, S_i, T, I, B$)
2:     $b_i \leftarrow None$
3:     for $k \leftarrow 1$ to $I$ do
4:         $(b_i, S_i) \leftarrow$ Assignment($b_i, d_i, S_i, T, B$)
5:         SendSignificanceList($S_i$)
6:         $(b_i, S_i) \leftarrow$ ConsensusAndRemoval($b_i, S_i, B$)
7:     end for
8:     return $b_i$
9: end function
```

B. PERFORMANCE IMPACT ALGORITHM (PIA)

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

C. DECENTRALIZED HUNGARIAN-BASED ALGORITHM (DHBA)

In DHBA, as shown in Algorithm 5, the input arguments are distance traveled (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.

D. GENETIC ALGORITHM (GA)

The decentralized Genetic Algorithm (Algorithm 6) is run in parallel by all agents, and takes the task list (T) and the initial population (p_i) as input. The initial population consists

Algorithm 5 DHBA

```
1: function DHBA($d_i, T, I$)
2:     $t_i \leftarrow None$
3:     $C_i \leftarrow None$
4:     for $k \leftarrow 1$ to $I$ do
5:         $t_i \leftarrow$ Assignment($C_i$)
6:         SendCostMatrix($C_i$)
7:         $C_i \leftarrow$ Update($C_i$)
8:     end for
9:     return $C_i$
10: end function
```

Algorithm 6 GA

```
1: function GA($T, p_i$)
2:     $t_i \leftarrow$ currentSolution($p_i$)
3:     SendCurrentSolution()
4:     receiveSolutions()
5:     $p_i \leftarrow$ mutatePopulation($t_i$)
6:     $p_i \leftarrow$ updatePopulation()
7:     return $p_i$
8: end function
```

of solutions to the multi-agent task allocation problem. The function currentSolution(p_i) (line 2) returns the best solution (t_i) from the current population. With each iteration, the population reproduces and mutates to provide better solutions. The agents continue executing the current solution till a better solution is returned by the function. Agents also keep broadcasting their current best solutions (line 3). The current solution is mutated and these mutations are added to the population (line 5). Using this approach of the GA, the best solutions stay in the population while the worst solutions are discarded. Solutions that are received from other agents (line 4) are incorporated into the current population of that agent's genetic algorithm (line 6).

ACKNOWLEDGMENT

The authors would like to thank Estefany Carrillo, Sharan Nayak, and Suyash Yeotikar, whose work on developing the simulation framework for running experiments on multi-agent task allocation algorithms has provided them with a strong foundation for conducting this research. Akshay Bapat worked on this project while at the University of Maryland and then later moved to Magna International.

REFERENCES

[1] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *Proc. Int. Conf. Emerg. Secur. Technol.*, Sep. 2010, pp. 142–147.

[2] A. Barrientos, J. Colorado, J. D. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *J. Field Robot.*, vol. 28, no. 5, pp. 667–689, 2011.

[3] N. Nigam, S. Bieniawski, I. Kroo, and J. Vian, "Control of multiple UAVs for persistent surveillance: Algorithm and flight test results," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 5, pp. 1236–1251, Sep. 2012.

[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. 78868–78880, 2018.

[5] 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," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 572–579, Apr. 2020.

[6] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Auto. Robots*, vol. 44, nos. 3–4, pp. 547–584, Mar. 2020.

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

[8] 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 *Proc. AIAA Guid., Navigat., Control Conf.*, Aug. 2010, p. 8421.

[9] S. Ismail and L. Sun, "Decentralized hungarian-based approach for fast and scalable task allocation," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2017, pp. 23–28.

[10] W. Zhao, Q. Meng, and P. W. H. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[11] J. Kirk, "Fixed start/end point multiple traveling salesmen problem—Genetic algorithm," MATLAB Central File Exchange, 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/21299-fixed-start-end-point-multiple-traveling-salesmen-problem-genetic-algorithm

[12] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[13] 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*. Cham, Switzerland: Springer, 2016, pp. 221–233.

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

[15] R. Patel, E. Rudnick-Cohen, S. Azarm, M. Otte, H. Xu, and J. W. Herrmann, "Decentralized task allocation in multi-agent systems using a decentralized genetic algorithm," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 3770–3776.

[16] A. Samiei, S. Ismail, and L. Sun, "Cluster-based Hungarian approach to task allocation for unmanned aerial vehicles," in *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, Jul. 2019, pp. 148–154.

[17] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-MCTS: Decentralized planning for multi-robot active perception," *Int. J. Robot. Res.*, vol. 38, nos. 2–3, pp. 316–337, 2019.

[18] M. Rantanen, N. Mastronarde, J. Hudack, and K. Dantu, "Decentralized task allocation in lossy networks: A simulation study," in *Proc. 16th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2019, pp. 1–9.

[19] M. Alighanbari and J. How, "Robust decentralized task assignment for cooperative UAVs," in *Proc. AIAA Guid., Navigat., Control Conf. Exhib.*, Aug. 2006, p. 6454.

[20] P. B. Sujit, A. Sinha, and D. Ghose, "Multi-UAV task allocation using team theory," in *Proc. 44th IEEE Conf. Decis. Control*, Dec. 2005, pp. 1497–1502.

[21] J. Radak, D. Schneider, C. Henke, and H. Frey, "Performance of consensus and formation control subject to Bernoulli, slotted Aloha and IEEE 802.11p simulation models," in *Proc. 12th IFIP Wireless Mobile Netw. Conf. (WMNC)*, Sep. 2019, pp. 63–70.

[22] J. W. Herrmann, "Data-driven metareasoning for collaborative autonomous systems," Inst. Syst. Res., Univ. Maryland, College Park, MD, USA, Tech. Rep. ISR;TR_2020-01, 2020. [Online]. Available: https://drum.lib.umd.edu/handle/1903/25339 and http://hdl.handle.net/1903/25339, doi: 10.13016/0xho-koz2.

[23] E. Carrillo, S. Yeotikar, S. Nayak, M. K. M. Jaffar, S. Azarm, J. W. Herrmann, M. Otte, and H. Xu, "Communication-aware multi-agent metareasoning for decentralized task allocation," *IEEE Access*, vol. 9, pp. 98712–98730, 2021.

[24] A. V. Bapat, "Development of decentralized task allocation algorithms for multi-agent systems with very low communication," M.S. thesis, A. James Clark School Eng., Inst. Syst. Res., Univ. Maryland College Park, College Park, MD, USA, 2020.

[25] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School Ind. Admin., Carnegie-Mellon Univ. Pittsburgh Manag. Sci. Res. Group, Pittsburgh, PA, USA, Tech. Rep., 388, 1976.

[26] J. Beardwood, J. H. Halton, and J. M. Hammersley, "The shortest path through many points," *Math. Cambridge Philos. Soc.*, vol. 55, no. 4, pp. 299–327, 1959.

[27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, and A. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, p. 5.

[28] A. Koubâa, *Robot Operating System (ROS)*, vol. 1. Cham, Switzerland: Springer, 2017.

[29] X. Jia and M. Q.-H. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2013, pp. 2280–2285.

[30] 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*, A. Koubâa and J. R. Martínez-de Dios, Eds. Switzerland: Springer, 2015, pp. 31–51, doi: 10.1007/978-3-319-18299-5.

[31] A. A. Khuwaja, Y. Chen, N. Zhao, M.-S. Alouini, and P. Dobbins, "A survey of channel modeling for UAV communications," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2804–2821, 4th Quart., 2018.

[32] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[33] D. Ruppert and D. S. Matteson, *Statistics and Data Analysis for Financial Engineering*, vol. 13. Cham, Switzerland: Springer, 2011.

AKSHAY BAPAT received the B.S. degree in mechanical engineering from IIT Jodhpur, India, in 2018, and the M.S. degree in systems engineering from the University of Maryland, College Park, MD, USA, in 2020.

In 2020, he was a Faculty Assistant at the Department of Aerospace Engineering, University of Maryland. From 2020 to 2021, he was a Robotics Test Engineer at Magna International in Troy, MI, USA. Since 2021, he has been working as a Robotics Perception Engineer with Magna International. His research interests include multi-agent task allocation, motion planning, and robot perception.

BHARATH REDDY BORA received the B.Tech. degree in electronics and communication engineering from GITAM University, Vizag, Andhra Pradesh, India, in 2020. He is currently pursuing the M.E. degree in robotics with the University of Maryland, College Park, MD, USA.

He was an Intern at the Motion and Teaming Laboratory at the University of Maryland. He is working as a Research Intern with the Bio-Imaging and Machine Vision Laboratory. His research interests include robotics, task planning/allocation, machine vision, autonomous systems, embedded systems, the Internet of Things, and robotics.

Mr. Bora contributed to the IEEE Student Body as the Technical Head and a Board Member of the Governing Council at Sure Trust, GITAM University.

JEFFREY W. HERRMANN received the B.S. degree in applied mathematics from the Georgia Institute of Technology, Atlanta, GA, USA, in 1990, and the Ph.D. degree in industrial and systems engineering from the University of Florida, Gainesville, FL, USA, in 1993.

He is currently a Professor with the University of Maryland, College Park, MD, USA, where he holds a joint appointment with the Department of Mechanical Engineering and the Institute for Systems Research. He is the author of the textbook *Engineering Decision Making and Risk Management* (Wiley, 2015). His current research interests include collaborative search and tracking and engineering design decision making.

Dr. Herrmann is a member of ASEE, IISE, ASME, the Design Society, and INFORMS. He is an Associate Editor of the *Journal of Autonomous Vehicles and Systems*.

SHAPOUR AZARM received the B.S. degree in mechanical engineering from the University of Tehran, Tehran, Iran, in 1977, the M.S. degree in mechanical engineering from George Washington University, Washington, DC, USA, in 1979, and the Ph.D. degree in mechanical engineering from the University of Michigan, Ann Arbor, MI, USA, in 1984.

He joined the University of Maryland, College Park, in 1984, where he is a Professor of mechanical engineering. He is a Senior Advisor of (journal of) *Structural and Multidisciplinary Optimization* (SMO). His research interests include predictive modeling, engineering optimization, and decision analysis models and methods.

Dr. Azarm is a fellow and a Life Member of the ASME. He was the Editor-in-Chief of the *Journal of Mechanical Design* (ASME Transactions), a Review Editor of *SMO*, an Associate Editor of (journal of) *Mechanics Based Design of Structures and Machines*, and several other journals.

HUAN XU (Member, IEEE) received the B.S. degree in mechanical engineering from Harvard University, Cambridge, MA, USA, in 2007, and the M.S. and Ph.D. degrees in mechanical engineering from the California Institute of Technology, Pasadena, CA, USA, in 2008 and 2013, respectively.

She joined the University of Maryland, College Park, in 2013, where she is currently an Associate Professor of aerospace engineering with a joint appointment at the Institute for Systems Research and a member of the Maryland Robotics Center. Her research interests include controls and dynamical systems, safety certification for autonomous systems, and multi-agent control.

Dr. Xu is a member of AIAA, AUVSI, and INCOSE.

MICHAEL W. OTTE (Member, IEEE) received the B.S. degree in aeronautical engineering and computer science from Clarkson University, Potsdam, NY, USA, in 2005, and the M.S. and Ph.D. degrees in computer science from the University of Colorado Boulder, Boulder, CO, USA, in 2007 and 2011, respectively.

From 2011 to 2014, he was a Postdoctoral Associate at the Massachusetts Institute of Technology. From 2014 to 2015, he was a Visiting Scholar at the U.S. Air Force Research Laboratory. From 2016 to 2018, he was a National Research Council RAP Postdoctoral Associate at the U.S. Naval Research Laboratory. He has been with the Department of Aerospace Engineering, University of Maryland, College Park, MD, USA, since 2018. His research interests include autonomous robotics, motion planning, and multi-agent systems.

Dr. Otte is a member of AIAA. He has been an Associate Editor of IEEE International Conference on Robotics and Automation and IEEE/RSJ International Conference on Intelligent Robots and Systems, since 2020.