Failure-Aware Tasking for Teams of Drones

Jonathan Diller[†], Yee Shen Teoh[†], Robert Byers[†], Qi Han[†], John G. Rogers[‡], Neil T. Dantam[†]
[†]*Colorado School of Mines*, Golden, USA
[‡]*Army Research Laboratory*, Adelphi, USA
{jdiller, yeeshen_teoh, robert_byers, qhan, ndantam}@mines.edu, john.g.rogers59.civ@army.mil

***Abstract*—Teams of drones have been proposed for many monitoring and data collection applications, including forest fire monitoring, search and rescue, disaster response, and infrastructure inspection. However, robot systems can be stochastic, and uncertainty arises when operating environments are dynamic or hostile. This paper investigates the problem of assigning drones to tasks where the probability that a given group of drones can cooperatively complete a task follows a Poisson-Binomial distribution. We show how to determine if a solution exists and how to calculate an upper bound on the optimal solution. We present a variation of the branch-and-bound algorithm – termed Branch-and-Match – that is tailored to our problem and always finds an optimal solution at the cost of computation time. For a more tractable approach, we present a heuristics-based algorithm – termed M+ILS – that turns the problem into a balanced matching problem to find an initial solution then runs a variation of the Iterated Local Search (ILS) algorithm. Our M+ILS algorithm is applicable to distributed scenarios but finds suboptimal solutions. We evaluate these various algorithms in a simulated forest fire monitoring scenario based on the characteristics of a fleet of real drones. Our empirical results show that the M+ILS algorithm finds solutions with an average performance gap of 2.68% compared to the optimal solution found using the Branch-and-Match algorithm.**

***Index Terms*—Distributed Task Assignment, Drone Swarms, Failure-Aware Assignments, Poisson-Binomial Distributions**

I. Introduction

In July of 2022, the Washburn Fire of Mariposa Grove, CA broke out in Yosemite National Park and threatened a grove of 500 giant sequoias and 1,600 people when it burned 19.7 km^2 of the park and national forest [1]. In this paper, we study how teams of drones could have been deployed to aid firefighters in containing and monitoring the movements of the fire.

Using data captured by NASA's Fire Information for Resource Management System[1] (shown in Fig. 1), we have created a forest fire monitoring case study based on the Washburn Fire. From this data, we have designed four drone monitoring tasks to support the initial response to the fire (depicted as green flags in the figure). For an effective response, we need all tasks completed. Given a fleet of heterogeneous drones, the problem becomes determining which drones should perform which tasks.

Initially, assigning drones to the monitoring task may appear to be a standard assignment problem, a well-studied area [2]. However, drones exhibit non-deterministic behavior, particularly in terms of energy consumption rates, making their performance and operational endurance challenging to predict accurately. Environmental factors, such as wind, hardware factors, and the age of a battery will impact performance and make a drone's operational abilities stochastic. We argue that the stochastic nature of energy consumption and the probability that a drone may run out of energy before successfully executing a task (that is, system failure) should be considered when assigning tasks to drones for data collection and monitoring applications.

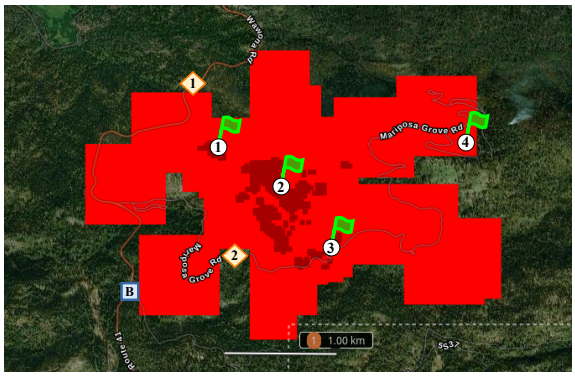

Fig. 1. Case study on 2022 Washburn Fire in Mariposa Grove, California. Green flags are monitoring tasks, orange diamonds are drone launch locations, and the blue square is the base station. Light red markers show low resolution hot spots while dark red markers indicate active fires within 30 meters.

When faced with uncertainty, it is advantageous to assign additional drones to a task to increase the probability that the task is completed. Suppose that Task 4 is an early fire detection task that only requires two drones to cover the area. However, if we assign more than two drones to Task 4 then the probability that at least two of the drones successfully monitor the area will increase. If we treat the probability that any individual drone successfully participates in the task as an independent trial and have a minimum number of required participating drones for the task to be completed, then the probability that the task is completed will follow a Poisson-Binomial distribution [3].

Furthermore, the tasks themselves are expected to be dynamic. Suppose that after deploying teams of drones for each task in our case study, the epicenter of the fire moves, changing the location and demands of each task. The dynamic nature of a forest fire creates the need for distributed algorithms that can quickly replan drone tasking during deployment.

In this work, we investigate a new class of task assignment problems where the objective function is a product of Poisson-Binomial trials. We term this class of problems the Maximum

This work was funded in part by W911NF-22-2-0235.
[1]firms.modaps.eosdis.nasa.gov/map/#d:2022-07-08;@-119.60,37.50,14.00z

Probability of Task Success (MPTS) problem, which includes compatibility and minimum-worker constraints. To the best of our knowledge, we are the first to consider this class of problem. **We make the following contributions in this article**:

1) We show how to determine if a solution exists to the MPTS problem in polynomial time and provide a theoretical upper bound on a solution.
2) We propose the Branch-and-Match algorithm, a complete solution based on the classic branch-and-bound algorithm that reduces branching by utilizing our work on solution existence.
3) We propose the M+ILS algorithm, a heuristics-based approach to solving the MPTS problem that can be implemented as a distributed algorithm and can quickly run online during mission execution.
4) We show how to assign a probability to the operational capabilities of drones and use this to motivate the MPTS problem setup.

Our extensive numerical simulations and deployment on a physical, distributed testbed demonstrate the utility of our two proposed algorithms and highlight their tradeoffs.

Although we focus on drone tasking for forest fire monitoring in this work, we argue that our problem setup and proposed algorithms can function in many other applications and problem domains. Possible other areas of application include infrastructure inspection, agriculture, manufacturing, or disaster response. To help further the frontiers of research and to promote good research integrity, our algorithm implementation and data sets have been made open-source[2].

II. Related Works

In this section we review related bodies of work, starting with various assignment problems and then look at distributed algorithms commonly used for solving assignment problems.

A. Task Assignment Problems for Robotics

In the linear assignment problem, we are given n workers (agents), m jobs (tasks), worker-to-job costs and are asked to find a one-to-one matching such that we minimize the summation of worker-to-job costs, where $m \geq n$. This problem can be solved using the well studied Hungarian method (Kuhn-Munkres algorithm), which runs in polynomial time [4]. A related problem is the general task assignment (GTA) problem, where we are given m items with profits and n bins with capacities and are asked to find an assignment of items to bins such that we maximize the expected profit without exceeding bin weight limits. This problem is known to be NP-hard [2] and is often applied to robot-task assignment scenarios [5].

Many works have expanded the GTA problem to incorporate probabilities. Notably, in the Weapon-Target Assignment (WTA) problem we are given a set of targets with rewards, a set of weapons with a probability of destroying a target, and are asked to find weapon-to-target assignments that maximize the expected reward. The WTA problem is also known to be NP-complete [6]. A variety of solutions have been proposed for the WTA problem, such as branch-and-bound algorithms that exploit some aspect of the problem and find optimal solutions [7]. Among heuristics-based approaches, Very Large Scale Neighborhood Search (VLSN) algorithms are quite popular [8], [9], as are biology-based solutions (e.g. bee colony algorithms [10]). Similar in nature to exact methods for the WTA problem, we design a custom branch-and-bound algorithm that exploits the problem structure of the MPTS problem to find optimal solutions.

Although the WTA problem and the MPTS problem are similar in nature, there are some key differences between the two. The MPTS problem seeks to maximize the probability that *all* tasks are completed while the WTA problem seeks to maximize a reward for completing tasks. Furthermore, the WTA problem maximizes the probability that one or more agents complete a task while our MPTS problem maximizes the probability that $d_j \geq 1$ or more agents are successful, where d_j is the minimum number of agents required to complete task j. The former is a product of Bernoulli trials while the latter requires one to optimize the cumulative distribution function (CDF) of a Poisson-Binomial Distribution (PBD), which changes the dynamics of how moving an agent from one task to another impacts the objective.

There are few examples of assignment problems that involve phenomenon that exhibit a PBD. Examples include sensing wireless channel activity [11] or randomly occurring jobs [12]. These works optimize some cost function and only use the PBD as a constraint to limit errors or future infeasibility. In this work, our objective is to maximize the CDF of a PBD rather than incorporating the CDF into problem constraints.

B. Distributed Task Assignment Algorithms

Given the dynamic nature of the considered scenario, we seek a distributed assignment algorithm for the MPTS problem that is tractable enough for online decision making. Furthermore, we prefer a solution that does not require the robots to share all of their capabilities or energy status due to privacy concerns.

A common approach to solving assignment problems is to run distributed versions of centralized algorithms involving matchings over bipartite graphs. This is seen in [13], where they search for augmenting paths over a bipartite graph in a distributed solution to the Bottleneck Assignment problem (which seeks to maximize the minimum cost assignment). Similarly, the authors in [14] propose a distributed version of the Hungarian method for solving the linear assignment problem to optimality that builds a weighted covering in a bipartite graph of agents to tasks by finding minimum cost matchings then searches for augmenting paths over the graph. Another common distributed approach for the linear assignment problem is for the agents to share costs and then solve the bipartite matching locally, as seen in [15], [16], while others have investigated how to reduce message passing by limiting what information is shared [17]. Similar to the idea presented

[2]https://github.com/pervasive-computing-systems-group/MPTS

in [15], our M+ILS algorithm forms a balanced matching problem out of the MPTS problem and uses a distributed version of the Hungarian method to find an initial solution. To reduce information sharing, we recommend Giordani's distributed implementation of the Hungarian method [14], but this can be changed for more efficient algorithms when information privacy is not a concern.

Among distributed algorithms for finding near-optimal solutions, a common approach is to allow agents to run local searches over iterations. One example of this is gradient decent algorithms, which are popular for training machine learning models [18] and for multi-agent, multi-objective convex optimization problems [19]. In [20], the authors use a distributed local search to find graph partitions in large social networks. The authors in [21] propose an iterative Monte-Carlo tree search for searching local robot-action spaces for actions that improve cumulative team reward.

An interesting aspect of the MPTS problem is that not all drones are required for a valid solution, giving the drones more freedom to search their local solution space without creating an invalid solution. Inspired by this fact, our M+ILS algorithm utilizes the ILS algorithm [22] to improve on the initial solution. In the original ILS algorithm, we perform a local search then randomly "perturb" the search space to move out of local minimum. There is limited work on distributed versions of the ILS algorithm (see [23], [24] where they still require a leader-follower structure). In this work, we propose a distributed variation of the ILS algorithm where each drone searches its local solution space then perturbs its neighbor without sharing all of its capability information. When combined with the initial solution, our M+ILS algorithm greatly improves solution quality over the classic ILS algorithm.

III. Maximum Probability of Task Success Problem Formulation

In this section we demonstrate how to calculate the probability that a drone has enough energy to complete an assigned task then formally define our problem, termed the Maximum Probability of Task Success (MPTS) problem.

A. Drone Energy-based Probability of Contribution

Drone energy depletion has been shown to depend on the physical characteristics of the drone platform and on the speed that the drone travels at [25]. Building on the model presented in [26], [27], let the total energy budget of drone i be ${}^{m}s_i$, the total time duration that the drone can travel at max speed ${}^{m}v_i$ starting with a fully charged battery and operating in an ideal environment, which is measured in seconds and termed *seconds-moving*. Mathematically,

$$ {}^{m}s_i = \frac{V_{bat} B_{rate}}{\mathcal{P}({}^{m}v_i)} \tag{1} $$

where V_{bat} is the voltage rating of the equipped battery, B_{rate} is the depletion rate of the battery, and $\mathcal{P}({}^{m}v_i)$ is the power used by drone i when moving at ${}^{m}v_i$, which is further discussed in detail in [25]. We will assume that all drones travel at max speed in this work as this has been shown to generally outperform moving at slower speeds [28], [29]. From (1), we see that the total time duration that a drone can hover (i.e. when $v_i = 0$) with a fully charged battery will be

$$ {}^{h}s_i = \frac{V_{bat} B_{rate}}{\mathcal{P}({}^{h}v_i)} \tag{2} $$

where $\mathcal{P}({}^{h}v_i)$ is power consumption when hovering.

Based on this model for drone energy, if task j requires drone i to travel a total of $dist_{ij}$ meters and hover for t_j seconds, then the predicted energy budget required (in *seconds-moving*) to complete the task will be:

$$ s_{ij} = \frac{dist_{ij}}{{}^{m}v_i} + t_j \frac{{}^{m}s_i}{{}^{h}s_i} \tag{3} $$

However, wear and age will impact the capacity and depletion rate of a battery. The authors in [30] found experimentally that the capacity of a lithium ion battery will decrease linearly as a function of the number of charging cycles. Building on this work, let the percent reduction in battery capacity after c recharging cycles be:

$$ r(c) = 1 - \varphi c \tag{4} $$

where φ is a constant determined experimentally. In addition to battery wear, several external factors such as wind, precipitation, elevation, and others will also impact how quickly the drone consumes energy during deployment. All of these factors are challenging to accurately predict. Instead, we propose adding an error term to our energy budget that follows a continuous probability distribution. If we choose to model the error as a normal distribution, our energy budget becomes:

$$ s_i = r(c) \frac{V_{bat} B_{rate}}{\mathcal{P}({}^{m}v_i)} \sim \mathcal{N}(\mu_i, \sigma_i) \tag{5} $$

where mean $\mu_i = {}^{m}s_i$ and standard deviation $\sigma_i = \gamma\, {}^{m}s_i$ for some interval of confidence $\gamma \in [1, 0]$. We chose a normal distribution due to its universality but acknowledge that other continuous probability distributions could be substituted into our formulation without major change. Determining the exact characteristics of the distribution requires further field experimentation and is left for future work.

The probability that drone i completes task j without running out of energy will be

$$ p_{ij} = \int_{s_{ij}}^{\infty} \frac{1}{\sigma_i \sqrt{2\pi}} e^{\frac{-(t-\mu_i)^2}{2\sigma^2}} dt. \tag{6} $$

In other words, p_{ij} is the probability that the actual energy budget of drone i is greater than or equal to s_{ij}.

B. Maximum Probability of Task Success Problem

We now build on (6) to formally define the MPTS problem. We refer the reader to Table I for a list of symbols used in our formulation.

We are given a set I of n drones and a set J of m tasks, where $n \geq m$, and are asked to assign drones to tasks such that the probability of all tasks being completed is maximized.

Each drone has a set of capabilities (e.g. sensors) and each task has capability requirements (e.g. sensors required). Let $\boldsymbol{c}_i$ be a binary array of capabilities for drone i and $\boldsymbol{C}$ be the capability matrix for all drones. Let $\boldsymbol{r}_j$ be a binary array of capability requirements for task j with $\boldsymbol{R}$ being the matrix of all task requirements. Each $c_{ik} \in \boldsymbol{c}_i$ is 1 if drone i has capability k and 0 otherwise. Similarly, each $r_{jk} \in \boldsymbol{r}_j$ is 1 if task j requires capability k and 0 otherwise. From (6) we get p_{ij}, the probability that drone i successfully contributes to task j, for $i \in I$ and $j \in J$. Although we focus on drones, our problem scenario can be reapplied to any scenario where one can assign a value to p_{ij}.

Each task requires a minimum of d_j drones although we allow additional drones (so called "floating agents") to be assigned to a task. For task j to successfully complete, at least d_j drones assigned to task j must successfully contribute to the task. For example, suppose we determined that we need two drones to canvas the area with thermal cameras for eight minutes to determine the risk to the giant sequoias for Task 4 ($j = 4$) from our case study. However, we can increase the probability that at least two drones return with thermal images by assigning more than two drones to survey j. Determining how to coordinate data collection between the drones in the event of failures such that required data is returned will depend on the specifics of the monitoring task, which we leave for future work.

Let I_j be the set of drones assigned to task j, where $I_j \subseteq I$. The probability that task j gets completed (denoted as $P_s(d_j, I_j)$) is the probability that d_j or more drones from I_j successfully contribute to the task. We treat each drone $i \in I_j$ as a sequential independent binomial trial where each trial has probability p_{ij} of success. This means that $P_s(d_j, I_j)$ is the probability that d_j or more trials are successful. Because each trial is independent (i.e. $\sum_{i \in I_j} p_{ij}$ may not be 1 and we are not guaranteed that $p_{ij} = p_{i'j}$ or $p_{ij} = p_{ij'}$), the probability that d_j or more drones successfully contribute to a task follows a PBD [3]. Given set I_j, the probability that *exactly* k of the trials are successful is found using the probability mass function (PMF) of a PBD:

$$P_r(k, I_j) = \sum_{A \in F_k} \prod_{i \in A} p_{ij} \prod_{i' \in A'} (1 - p_{i'j}) \tag{7}$$

where F_k is the set of all combinations of size k of the drones in I_j and $A' = I_j \setminus A$ is the compliment of A. For example, suppose $I_j = \{1, 2, 6\}$ and $k = 2$, then $F_k = \{\{1, 2\}, \{1, 6\}, \{2, 6\}\}$ and when $A = \{1, 2\}$ we get $A' = \{6\}$.

To determine $P_s(d_j, I_j)$, the probability that d_j *or more* drones contribute to the task, we must compute the complement of the cumulative distribution function (CDF), termed the "survival function". In the general, case where $|I_j| = n_j \geq d_j$,

$$P_s(d_j, I_j) = \sum_{k=d_j}^{n_j} P_r(k, I_j) \tag{8}$$

We can now formally define the MPTS problem:

Definition 3.1 (Maximum Probability of Task Success Problem): Given set of drones I and set of tasks J with minimum numbers of drones d_j for each $j \in J$, find drone-to-task assignment sets $I_j \subseteq I$ that:

$$maximize \prod_{j \in J} P_s(d_j, I_j) \tag{9}$$

subject to

$$I_j \cap I_{j'} = \emptyset, \forall j, j' \in J \tag{10a}$$

$$|I_j| \geq d_j, \forall j \in J \tag{10b}$$

$$(\boldsymbol{c}_{ik} - \boldsymbol{r}_{jk}) \geq \boldsymbol{0}, \forall k \in K, j \in J, i \in I_j \tag{10c}$$

Constraint (10a) ensures drones are assigned at most once. Constraint (10b) requires each task to be assigned at least the minimum number of drones, d_j, for the task to be completed. Constraint (10c) forces task requirements to be met.

Observe that in (7) there will be $\frac{n_j!}{(n_j - k)!\ k!}$ elements in F_k, making (9) difficult to compute as $n_j \gg d_j$. In [31], they show how to rewrite the PMF as a Vandermonde matrix and simplify it into a DFT matrix, the Fourier transform of which leads to a closed-form expression of the PMF for the PBD. From their work, we find that the PMF can be computed as

$$P_f(k, I_j) = \frac{1}{n_j + 1} \sum_{l=0}^{n_j} \left\{ e^{\frac{-2\pi l k \sqrt{-1}}{n_j + 1}} \prod_{i \in I_j} \left\{ p_{ij} e^{\frac{2\pi l k \sqrt{-1}}{n_j + 1}} + (1 - p_{ij}) \right\} \right\} \tag{11}$$

For brevity, we do not show the steps to derive (11) but refer the reader to [31] for these details. The survival function then simplifies to

$$P_s(d_j, I_j) = \sum_{k=d_j}^{n_j} P_f(k, I_j) \tag{12}$$

Note that the number of terms in (12) grows as a polynomial with the size of I_j and we can compute $P_s(d_j, I_j)$ in $\mathcal{O}(n_j^2)$.

We can formulate an integer non-linear program to solve the MPTS problem as follows: Let x_{ij} be a binary variable,

TABLE I
SYMBOLS

Symbol	*Type*	*Description*
$\boldsymbol{C}$	Param. Matrix	$c_{ik} = 1$ if drone i has capability k, 0 otherwise
$\boldsymbol{c}_i$	Param. Array	Capability row for agent i in parameter matrix $\boldsymbol{C}$
d_j	Parameter	Number of drones required for task j
I	Set	Set of all available drones
I_j	Set	Set of drones assigned to task j
J	Set	Set of all tasks
K	Set	Set of all considered capabilities
$p_{ij}(t)$	Function	Probability that drone i contributes to task j
$\boldsymbol{R}$	Param. Matrix	$r_{jk} = 1$ if task j requires capability k, 0 otherwise
$\boldsymbol{r}_j$	Param. Array	Requirement row for task j in parameter matrix $\boldsymbol{R}$

where $x_{ij} = 1$ if drone i is assigned to task j and 0 otherwise. Let a_{jl} be a binary variable, for $j \in J$ and $l \in \{0, 1, \cdots n\}$, where $a_{jl} = 1$ if there are at least l drones assigned to task j and 0 otherwise. We re-write (11) as

$$P_{IP}(k,j) = \frac{1}{n_j+1} \sum_{l=0}^{n} \left\{ (a_{jl}) e^{\frac{-2\pi lk\sqrt{-1}}{n_j+1}} \prod_{i \in I} \left\{ p_{ij} e^{\frac{2\pi lk\sqrt{-1}}{n_j+1}} + (1 - p_{ij}) \right\}^{x_{ij}} \right\} \tag{13}$$

The survival function for our Integer Program is

$$P_{sIP}(j) = \sum_{k=d_j}^{n} P_{IP}(k,j) \tag{14}$$

The Integer Non-Linear program to determine the optimal drone-to-task assignment to maximize the probability that all tasks are completed is:

$$maximize \prod_{j \in J} P_{sIP}(j) \tag{15}$$

subject to

$$\sum_{j \in J} x_{ij} \leq 1, \, \forall i \in I \tag{16a}$$

$$\sum_{i \in I} x_{ij} \geq d_j, \, \forall j \in J \tag{16b}$$

$$(\boldsymbol{c}_{ik} - \boldsymbol{r}_{jk}) x_{ij} \geq \mathbf{0}, \, \forall k \in K, j \in J, i \in I \tag{16c}$$

$$\sum_{i \in I} x_{ij} \geq l a_{jl}, \, \forall j \in J, l \in \{0, 1, \cdots n\} \tag{16d}$$

We focus on drone energy to determine p_{ij}, but the MPTS problem can be reapplied to new domains where p_{ij} can be defined for tasking independent events. For example, p_{ij} could be the probability that a machine fails while producing widgets in a manufacturing scenario.

IV. Solution Existence and Upper Bound

As previously stated, the objective function for the MPTS problem as presented in (9) will take exponential time to compute, making the problem formulation sit in EXPTIME. The integer program presented in (15) is non-convex, a class of problems known to be NP-Hard. At the time of writing, there is no known method for formulating the MPTS problem in a form that can be solved in polynomial time.

Although the MPTS problem is hard to solve to optimality, we can determine if a solution exists in polynomial time by mapping the problem to a minimum cost balanced matching problem – a variation of the linear assignment problem where there is an equal number (n) of workers and jobs, with cost k_{ij} for each worker i to job j assignment.

Figure 2 helps demonstrate how to map the MPTS problem into a balanced matching problem that can determine if a valid solution exists. We start by creating d_j jobs for each $j \in J$. These jobs are depicted in the figure as $j.k$ where $1 \leq j \leq m$ and $1 \leq k \leq d_j$. These jobs represent the minimum number of drones that must be assigned to tasks in the MPTS formulation. Let n_r be the total number of drones required to meet d_j for every task and n_f the number of floating agents. That is, $n_r = \sum_{j \in J} d_j$ and $n_f = n - n_r$. For each task j in MPTS, we create n_f jobs in the balanced matching problem in addition to the d_j jobs for j. These are shown in the figure as $j.x$ for $1 \leq j \leq m$. Let n_t be the total number of jobs required to form a balanced matching. That is, $n_t = \sum_{j \in J} d_j + n_f |J|$. To make the problem balanced, we add $n_t - n$ new "phantom workers", which are indicated as "x" in the workers column.

For each worker i and job j, we set $k_{ij} = 0$ if $(\boldsymbol{c}_{ik} - \boldsymbol{r}_{jk}) \geq \mathbf{0}$ and to 1 otherwise. The cost of assigning a phantom worker to any of the required d_j jobs is 1 (i.e. a phantom worker will fail to complete a required task), and the cost of assigning a phantom worker to a floating job is 0 (i.e. a phantom worker is guaranteed to complete a floating task). We call our formulated matching problem $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$. We can find a least cost assignment to our balanced matching using the Hungarian method [4], which runs in polynomial time.

Theorem 1: There exists a solution to the MPTS problem formulated from $I, J, \boldsymbol{C}$ and $\boldsymbol{R}$ if and only if there exists a minimum cost matching of zero in $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$.

Proof 1: Assume that there is a minimum cost matching of zero for $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$ and no solution exists to the MPTS problem. Because there is a minimum cost matching of zero for $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$, and job assignment costs in $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$ are, by construction, zero for drones in the MPTS problem that can actually perform each task, then matching of workers to jobs in $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$ must be a valid matching of drones to tasks in the MPTS problem. Therefore, our assumption that no solution exists to the MPTS problem must have been wrong.

The same proof can be applied in reverse to show that if a solution exists for the MPTS problem then the minimum cost matching in $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$ must be zero. □

If the minimum cost matching in $\mathcal{B}(I, J, \boldsymbol{C}, \boldsymbol{R})$ is greater than zero, this tells us that there are not enough drones to meet the capability requirements of every task. That is, at least one task will have less than d_j drones that meet requirements $\boldsymbol{r}_j$ and objective function (15) will evaluate to zero.

Additionally, we can provide an upper bound on a solution by dropping constraint (16a), allowing robots to be assigned to an unlimited number of tasks. In this case, we set $x_{ij} = 1$ for all drone i and task j that do not break constraint (16c)

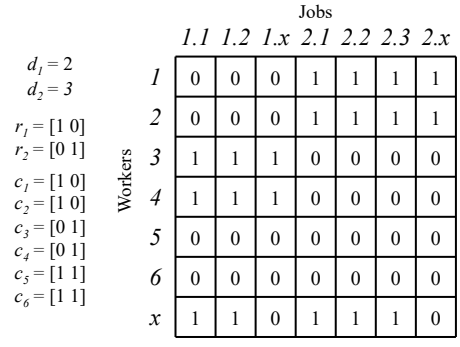

Fig. 2. Balanced matching problem example to prove solution existence.

and then compute:

$$\prod_{j \in J} P_{sIP}(j) \tag{17}$$

Equation (17) provides an upper bound on a given problem input, which we use in the following section to speed-up a branch-and-bound algorithm for the MPTS problem.

V. Algorithms to Solve the MPTS Problem

In this section we present various algorithms for solving the MPTS problem. We first present the *Match + Iterated-Local-Search* (M+ILS) algorithm, a distributed algorithm that builds on the problem structure introduced in Theorem 1 to quickly find a solution to the MPTS problem. We then look at a modified version of the classic branch-and-bound algorithm – termed branch-and-match – that utilizes our solution existence work in Section IV and our upper bound on a solution from (17) to speed-up solve time.

A. Match + Iterated-Local-Search Algorithm

Observe that (15) will increase as more drones are assigned to tasks that already have d_j drones. This is particularly true for tasks with smaller values for (14). Furthermore, if any task j has less than d_j drones assigned to it, then (15) evaluates to zero. This observation motivates our proposed M+ILS algorithm. For each task j, the algorithm attempts to assign the d_j best suited drones to the task in an initial solution and leaves the remaining drones free. The remaining drones can then be used to perturb the search space while running a distributed variation of the ILS algorithm. We will first look at how to find an initial solution then discuss our adapted ILS routine. Algorithm 1 shows the outline of the M+ILS algorithm.

We introduce the algorithm as a distributed solution but a centralized version can easily be implemented by placing the algorithm into a loop and iterating over each drone. For the distributed version of the algorithm, we assume that each drone knows its own probability of contributing to each task. We argue that this assumption is reasonable because each drone could monitor the status of its battery and calculate (6). We also assume that each drone knows d_j, its own capabilities vector c_{ik}, and the requirements vector of each task r_{jk}, but has no knowledge of the probability that any other drone can contribute to a task or what the capabilities of the other drones are. We also assume that the drones have a numbered ordering and are wirelessly connected through one or more hops.

The idea behind our approach to finding an initial solution is to map the problem into a balanced matching problem with n workers (drones) and n jobs (tasks), where some jobs represent actual tasks and others are "null-jobs" that are not associated with any task. Figure 3 shows an example of how we form this balanced matching problem. We first create d_j jobs for each $j \in J$, shown as $j.k$ for $1 \leq j \leq m$ and $1 \leq k \leq d_j$ in the figure. These jobs represent the minimum number of drones that must be assigned to tasks in the original MPTS formulation. We are required to assign a total of $n_r = \sum_{j \in J} d_j$ drones to the tasks in MPTS and have

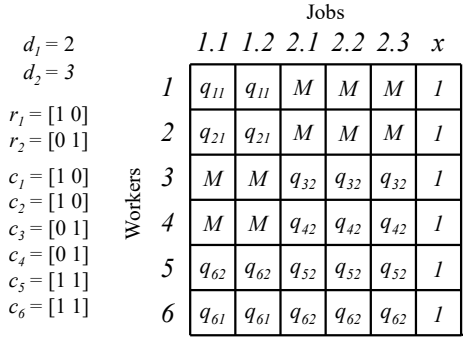

Fig. 3. Matching problem example for the M+ILS algorithm.

$n_f = n - n_r$ floating drones that are not required to complete the tasks but can be used to reinforce the other drones in the case of failure. We create n_f additional null-jobs, indicated as an "x" in the example. We now have n workers and n jobs.

For each worker-job combination, where the worker represents drone i and the job represents task j, we set the cost of the worker doing the job to $q_{ij} = 1 - p_{ij}$, the probability that drone i fails to contribute to task j. For any drone-to-task combination where the drone does not meet the capabilities of the task, we set a cost of M, some arbitrarily large number. For all null-jobs (i.e. "x" in the figure), we assign a cost of 1 because drones that do not get assigned to a task will fail to complete a task with probability 1. This entire process is done on each individual drone on line 1 of Algorithm 1, where $\boldsymbol{w}_i$ is a vector of task weights for drone i. For example, $\boldsymbol{w}_1 = [q_{11}, q_{11}, M, M, M, 1]^T$ in the example from Fig. 3. We note that this method is slightly different from the mapping discussed in Section IV but can be used to determine if a solution exists by the same proof used previously.

The drones then find an initial matching of drones to tasks

Algorithm 1 M+ILS

Function: *Start*():
1: $\boldsymbol{w}_i \leftarrow calculateWeights()$
2: $\mathcal{X} \leftarrow HungarianMethod(\boldsymbol{w}_i)$
3: *Iterated-Local-Search*($\mathcal{X}$)

Function: *Iterated-Local-Search*($\mathcal{X}$):
4: **if** no changes made to $\mathcal{X}$ since last iteration **then**
5: *stop algorithm*
6: **else if** $\exists j \in J$ where $\sum_{i \in I} x_{ij} < d_j$ **then**
7: *assign robot to* j in $\mathcal{X}$
8: **else**
9: $j \leftarrow$ *localSearch*($\mathcal{X}$)
10: *assign robot to* j in $\mathcal{X}$
11: **end if**
12: **if** neighbor still up **then**
13: *send* $\mathcal{X}$ *to neighbor*
14: **else**
15: *erase neighbor's entry in* $\mathcal{X}$
16: *send* $\mathcal{X}$ *to next neighbor*
17: **end if**

using the Hungarian method (on line 2). There are several distributed versions of the Hungarian method in existing literature (see [14]–[17]). We recommend Giordani's version of the algorithm in [14] because it does not require the drones to share all of their capability information and can adapt to failures. A brief description of this algorithm is found in Section II but we refer the reader to [14] for a more detailed description.

After finding an initial matching, the drones form solution $\mathcal{X}$, a $2 \times m$ matrix where x_{i1} stores which task drone i is currently assigned to and x_{i2} stores the probability that the drone can contribute to this task. The drones then begin the ILS portion of the algorithm (line 3) and pass $\mathcal{X}$ from drone-to-drone in ascending order.

When drone i receives table $\mathcal{X}$, it searches for a task j that does not have at least d_j drones assigned, checking tasks in descending order of p_{ij} (line 6). If there exists a task j with less than d_j drones assigned, and the drone is capable of performing task j, then drone i assigns itself to task j (i.e. the drone sets $x_{i1} = j$ and $x_{i2} = p_{ij}$). This section of the algorithm is needed should a drone drop-out of the system or if the *Iterated-Local-Search*() function is run by itself (as shown in Section VI as a baseline method). If no such task exists, then the drone runs a local search by iterating through all tasks that the drone is eligible to perform and computing (15) to find the task that maximizes the objective (line 9). Once assigned to a task, the drone passes $\mathcal{X}$ to its next neighbor. Should the next drone in the order fail (line 12 is false), then drone i removes drone $i + 1$'s entry in $\mathcal{X}$ and sends $\mathcal{X}$ to drone $i+2$. The algorithm ends when n iterations have passed without any drone making an update in $\mathcal{X}$.

As presented, the computation time of the algorithm will depend on the selected distributed Hungarian method. Giordani's implementation has a global computation time of $\mathcal{O}(n^3)$ with a message complexity of $\mathcal{O}(n^3)$. In the worst-case, the *Iterated-Local-Search()* function will run in $\mathcal{O}(m \cdot n^2)$ - the time required to compute (12) for every task - on each drone for each iteration. However, because each drone can only be assigned to a single task, the computation time for (12) is expected to be $\mathcal{O}\big((\frac{n}{m})^2\big)$, making the amortized runtime for a single iteration of the function $\mathcal{O}\big(\frac{n^2}{m}\big)$. We can guarantee that the *Iterated-Local-Search()* function converges in a fixed number of iterations by only allowing task changes if $\delta_j \geq \epsilon$, $\epsilon \in [0, 1]$. That is, a drone can change tasks only if the improvement in the objective function is greater than ϵ. In this case, the function will run for at most $\frac{1}{\epsilon}n$ iterations and will pass at most $\frac{1}{\epsilon}n$ messages in the worst-case, giving the *Iterated-Local-Search()* function a computation time of $\mathcal{O}\big(\frac{n^3}{\epsilon m}\big)$. This means that the Hungarian method will dominate and the computation complexity of the M+ILS algorithm is $\mathcal{O}(n^3)$. However, we hypothesize that the *Iterated-Local-Search()* function is likely to converge after m iterations, making it complete in n^2 time. Additionally, there are alternative distributed implementations of the Hungarian method that trade-off information privacy for speed (see [16]) that could be used without changing the structure of the M+ILS algorithm.

B. Optimal Solution: Branch-and-Match (BnM)

Our branch-and-match algorithm conceptually works the same as the classic branch-and-bound approach. We iterate through each drone, assigning them to individual tasks in turn and record when we find a better solution. We modify the traditional algorithm by considering two methods for pruning a branch. Given a partial assignment, we prune a branch if:

1) The upper bound on a solution by holding the partial assignment and solving (17) on unassigned drones is less than the incumbent solution, or
2) No solution exists to the sub-problem formed from the unassigned drones and unfilled tasks.

To check for the second condition, we remove drones that were assigned from the problem and reduce each d_j (the minimum number of drones required to complete task j) by the number of drones that were already assigned to task j. We then use the linear matching routine described in the proof of Theorem 1 to determine if a solution exists in the sub-problem. Additionally, we can further speed up the algorithm by using a heuristics-based algorithm to find an initial incumbent solution (a so-called "warm start") or by sorting drones so that the drones that are able to perform the most tasks are considered first. The motivation for sorting the drones is that this locks in the more capable drones early in a branch and lowers the upper bound by leaving less capable drones unassigned.

Our custom branch-and-bound algorithm will find the optimal solution. However, the algorithm will run in $\mathcal{O}\big(m^n\big)$ in the worst-case and is not feasible for larger inputs. This approach is also not ideal in a distributed environment, should the drones need to rerun the algorithm online, further motivating the need for our M+ILS algorithm.

VI. Performance Evaluation Via Simulation

We implemented both centralized and decentralized versions of the M+ILS algorithm, as well as the branch-and-match (BnM) solver, in C++. The centralized M+ILS uses a centralized, open source implementation of the Hungarian method[1]. For the decentralized implementation, we used socket programming and implemented Giordani's distributed version of the Hungarian method, as described in [14]. In this section, we will use the abbreviation "dM+ILS" to refer to the distributed version and M+ILS to refer to the centralized version.

A. Realistic Inputs

To the best of our knowledge, there is no public dataset on drone energy statistics or thorough experiments needed to determine γ. To cover this gap, we used drone data from the drone fleet used by the U.S. Department of Interior (DOI) [32] to make our simulations and case study more realistic. The drones considered for this problem are listed in Table II, where model 1 is the Matrice 600 Pro[2], 2 is the Mavic Pro[2], 3 is the FireFLY6 Pro[3], 4 is the 3DR Solo[4], 5 is the Pulse Vapor

[1] github.com/mcximing/hungarian-algorithm-cpp [2] DJI, www.dji.com
[3] BirdsEyeView Aerobotics [4] 3DR, www.3dr.com

TABLE II
DRONE SPECIFICATIONS

Drone Model	1	2	3	4	5	6
${}^{m}v_i$ (m/s)	18	20	30.5	25.5	11	15
${}^{m}s_i$ (s)	740*	990*	2400	270*	1800*	720*
${}^{h}s_i$ (s)	1080	1740	1200	1080*	2700	1080*
Sensors	2, 3, 5	1, 4	2, 4	3	1, 3, 5	4

* estimated value

55TM[5], and 6 is the Parrot Anafi[6]. The sensor types are 1 for EO/IR Camera, 2 for Thermal Camera, 3 for HD Camera, 4 for 4K Camera and 5 for a Lidar. The values in the table were either found in [32], provided by the manufacturer, or were estimated as indicated.

To calculate the probability that a drone can contribute to a task, we used Equ. 6 with $\gamma = 0.25$ and the values provided in Table II. This value for γ means that we have a 25% interval of confidence on our prediction of the total time duration that each drone can travel at max speed. In [30] they experimentally determined that the degradation rate of a battery $\varphi \approx 0.0005$, which allows us to determine $r(c)$, the percent drop in battery capacity after recharging c times.

Using the drone data in Table II, we generated four sets of MPTS problem inputs. *Set 1* has randomly located tasks in a 5 km × 5 km space with task times varying from 5 to 15 minutes and randomized sensor requirements. We deployed drones from Table II at random starting points with randomly set values for c between 0 and 400. We varied the number of drones from 5 to 30 (in increments of 1) and set the number of tasks to be 25% the number of drones with 25% of the drones available as floating agents. *Set 2* has the same setup as *Set 1*, but increments the number of drones from 4 up to 40, initially in increments of 1 up to 8 drones, then in increments of 4. *Set 3* again copies the general setup of *Set 1* except that the number of drones is fixed at 15 with 3 tasks and the number of floating agents varying from 0 up to 12. *Set 4* varies the number of drones from 50 up to 150, in increments of 10. There are multiple base stations (equal to 10% of the number of drones) that are scattered throughout a 3 km × 3 km space. Drones are assigned to a specific base station and begin within a 50 m radius of their assigned base station. For all data sets we generated 50 inputs for each distinct configuration, giving us a total of 1,300 inputs for *Set 1*, 650 inputs for both *Set 2* and *3*, and 550 inputs for *Set 4*.

B. Baseline Methods

We consider four different baseline approaches to compare against our proposed heuristics-based algorithm. In the first baseline method we form a balanced matching problem as described in Section IV with slight modifications. Rather than setting costs to 1 or 0, we set the cost for each drone-to-task combination to $q_{ij} = 1 - p_{ij}$ if drone i can perform task j, and some arbitrarily large number M otherwise. We then solve the balanced matching problem using the centralized Hungarian method. We refer to this approach as the Balanced Matching (BM) algorithm. The intuition behind BM algorithm is that minimizing the sum of probabilities that the drones will fail at their assigned task is a good proxy for solving (15) and promises to always find a solution, if one exists. Our second baseline approach is to run the *Iterated-Local-Search()* function from the M+ILS algorithm without running the matching portion of the algorithm first (ILS). The first two baseline approaches were chosen to determine the utility of each part of the M+ILS algorithm individually.

As a third baseline, we implemented a distance minimizing algorithm (MinDist) that puts the probability of failure into its constraints. We form a balanced matching problem by setting the cost for drone i to perform task j as the total distance that i would have to travel to perform j and restrict i from being assigned to j if $p_{ij} < 0.5$. We then find the minimum drone-to-task assignment using the Hungarian method. We chose this as a baseline because many related works use a "limit risk" type approach for assignments while minimizing some other criteria [11], [12], [26], [33].

For a fourth baseline, we implemented the VLSN algorithm, as initially described in [8]. The VLSN algorithm takes an initial solution and iteratively searches for a reassignment by attempting to swap the assignment of two or more drones. If a reassignment is found that yields a higher objective function value, then that solution is stored as the incumbent solution. The algorithm checks all possible swaps from 2 up to 5 drones at once and uses our first baseline algorithm to find an initial solution. We chose this baseline because it was found to perform well for the WTA problem [8], [9].

C. Simulation Results

We ran our proposed algorithms on the input sets described above to compare their performance against the various baseline methods. All experiments in the subsection were run on a machine with an Intel 3.4 GHz 16-Core CPU and 64 GiB of RAM.

1) Performance Versus Optimal Solution: We used *Set 1* to evaluate how increasing the number of drones impacts performance by comparing each algorithm's output against the optimal solution found using our BnM algorithm. The top graph in Fig. 4 shows the average ratio of the found solution over the optimal solution of each input (a ratio of 1 means that the algorithm found the optimal solution). We stopped running the experiment when the number of drones reached 19 because the BnM algorithm became too slow after this point. On average, our M+ILS algorithm was within 2.68% of the optimal solution. All baseline algorithms initially performed fair but then began to diverge from the optimal as the number of drones increases. The best performing baseline approach was the ILS algorithm, with solutions averaging within 10.61% of the optimal. The bottom graph in Fig. 4 shows the average percent gap between the optimal solution and the upper bound found using (17). The average gap is 24.3% but is much tighter when there are fewer drones (eight or less).

2) Performance at Scale: We next used *Set 4* to evaluate how well the heuristic-based algorithms perform on much

[5]AeroVironment, www.avinc.com [6]Parrot Drone SAS, www.parrot.com

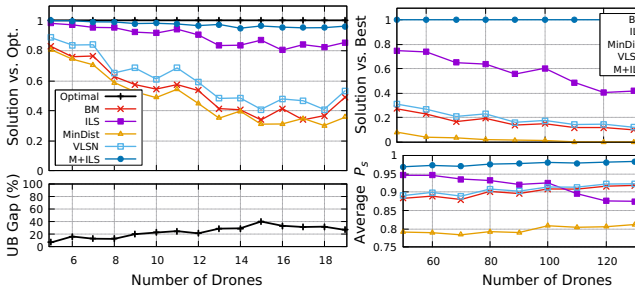

Fig. 4. Impact of increasing drones on solution.

Fig. 5. Impact of increasing drones at large scale.

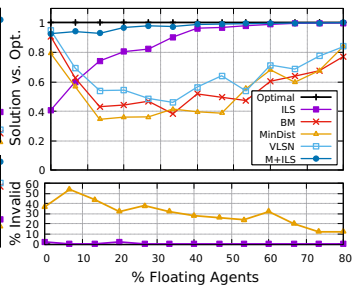

Fig. 6. Impact of increasing "floating agents."

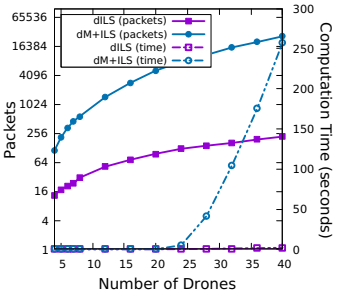

Fig. 7. Packet count and runtime of distributed algorithms.

larger inputs. The top graph in Fig. 5 shows the average ratio of the found solution for each algorithm over the best found solution for each input (a ratio of 1 means that the algorithm found the best known solution). The M+ILS algorithm found the best solution for every input while the ILS algorithm, although in clear second place and better than the other baseline approaches, begins to struggle as the number of drones greatly increases. The bottom graph of Fig. 5 shows the average probability that any individual task is completed – calculated using Equ. (12). Although Equ. (12) is not our considered objective function, we include this data because if an algorithm is finding assignments that are consistent the objective should trend downwards as the number of tasks increases (e.g. $0.99^2 \gg 0.99^{100}$). The plot shows us that the ILS algorithm struggles to find good assignments across all tasks when there are a large number of drones while the other approaches trend slightly upward with the M+ILS algorithm consistently well ahead of all other approaches.

3) Impact Of Floating Agents: We evaluated how the number of floating agents impacts performance by running our centralized solution on *Set 3*, where the number of drones and tasks are fixed at 15 and 3, respectively, but the number of floating agents varies from 0 up to 12 (80% of the drones). The top graph in Fig. 6 shows the average ratio of the found solution over the optimal solution found using the BnM algorithm. The plot shows that when all drones are required for a problem input (i.e. 0% floating agents) the M+ILS algorithm performs comparably to both the BM and VLSN methods. However, as the number of floating agents increases the BM and VLSN algorithms begin to struggle while the M+ILS improves and averages within 2.45% of the optimal over all inputs from data *Set 3*. The ILS algorithm initially performs very poorly compared to all other methods. Without any floating agents to switch between tasks, the ILS algorithm is essentially a greedy algorithm. However, the algorithm quickly trends towards the optimal as the percent of floating agents increases, beating all other baseline algorithms once 13% of the drones are floating agents and averages within 7.64% of the optimal solution after this point.

4) Finding Valid Solutions: We also used *Set 3* to evaluate how often each algorithm finds a valid solution. The bottom graph of Fig. 6 shows the percent of inputs where the algorithm fails to find a valid solution. The ILS algorithm slightly struggles when there are no floating agents available, failing on 2% of the inputs. This algorithm acts as a greedy solution when there are no floating agents available so we can anticipate that the algorithm will sometimes fail here. The MinDist algorithm struggles greatly at finding valid solutions, failing on 30.06% of the inputs. This is because of our constraint that no task will be considered when $p_{ij} < 0.5$, leading the algorithm to not consider valid solutions that might expose one or more of the drones to higher risk of failure. All other considered algorithms found valid solutions for all inputs and were omitted from the bottom graph of Fig. 6 to reduce clutter.

5) Computation Time (Centralized): Table III shows the computation time of the BnM algorithm compared to the classic branch-and-bound (BnB) approach that does not use the branch cutting techniques discussed in Section V-B. The classic BnB algorithm got stuck on inputs larger than 17 drones and 5 tasks. Our proposed BnM algorithm reached 19 drones before becoming too slow to compute solutions in a reasonable time. Table IV shows the average computation time of centralized versions of the heuristics-based algorithms as the number of drones grows. Our M+ILS algorithm handled large inputs quite well, averaging only 154 *ms* on inputs with 150 drones and 38 tasks. This was only topped by the MC algorithm, which averaged 118 *ms* on the 150 drone inputs. Notably, the VLSN algorithm struggled greatly as the number of drones increased.

6) Packet Count and Computation Time (Distributed): We used *Set 2* to evaluate the performance of the distributed implementation of the M+ILS algorithm and a distributed

TABLE III
COMPUTATION TIME (SECONDS) – EXACT METHODS

# Drones	9	11	13	15	17	19
BnB	$1.02{\cdot}10^3$	$7.68{\cdot}10^3$	0.740	1.23	582.3	-
BnM	$0.56{\cdot}10^3$	$7.90{\cdot}10^3$	0.183	0.921	26.1	541.4

TABLE IV
COMPUTATION TIME (SECONDS) – HEURISTICS-BASED (CENTRALIZED)

# Drones	30	50	70	90	110	130	150
BM	$1.1{\cdot}10^3$	$5.2{\cdot}10^3$	$1.8{\cdot}10^2$	$5.9{\cdot}10^2$	0.170	0.406	0.856
ILS	$3.9{\cdot}10^3$	$6.3{\cdot}10^3$	$1.2{\cdot}10^2$	$2.7{\cdot}10^2$	$4.8{\cdot}10^2$	$7.5{\cdot}10^2$	**0.118**
MinDis	$1.2{\cdot}10^3$	$4.3{\cdot}10^3$	$1.5{\cdot}10^2$	$5.2{\cdot}10^2$	0.139	0.339	0.713
VLSN	0.273	3.47	26.6	97.1	340.9	800.7	1461
M+ILS	$2.4{\cdot}10^3$	$6.3{\cdot}10^3$	$1.4{\cdot}10^2$	$3.1{\cdot}10^2$	$5.8{\cdot}10^2$	0.100	**0.154**

version of the ILS algorithm (dILS). We chose to only run the ILS baseline for this evaluation because it was shown to be notably better in solution performance and computation time compared to the other baseline approaches on *Set 1* (which has a similar setup to *Set 2*). Figure 7 shows the average number of packets sent over a socket (left y-axis in log scale) and the average computation time (right y-axis). The number of packets required for the dM+ILS algorithm grows much faster than the dILS algorithm, averaging 26,400 packets compared to only 219, respectively, when there are 40 drones. This leads to polynomial growth in computation time for the dM+ILS algorithm while the dILS algorithm grows near linearly with the number of drones.

7) Summary of Major Findings: Our simulation results show that the centralized version of our M+ILS algorithm performs very well across all considered scenarios (increasing number of drones, increasing percent of floating agents, finding valid solutions). The algorithm also runs very fast when implemented as a centralized solution. However, the distributed version of our algorithm (dM+ILS) requires a lot of message passing, leading to higher computation times when compared to the most competitive baseline approach. The computation time and message complexity of the algorithm could be improved by swapping out our chosen implementation of the distributed Hungarian method with the version presented in [16], which sacrifices capability privacy for algorithm efficiency. Changing the distributed Hungarian method implementation would not require structural change to our proposed algorithm.

D. Case Study Results

Returning to our case study in Fig. 1, let Tasks 1 and 3 be to monitor how the fire is moving along its boundaries while Task 2 is to provide personnel monitoring for firefighters close to the center of the fire. Task 4 is a preventative monitoring task for early fire detection near the giant sequoias. Tasks 1 and 3 require at least two drones with HD cameras for 10 minutes, Task 2 requires four drones equipped with both IR and HD cameras for 14 minutes, and Task 4 requires two drones with thermal cameras for 8 minutes. A total of 15 drones are to be launched from the two launch locations shown in Fig. 1. Five model 5 drones with older batteries (350 cycles) will be launched from location 1. Four model 1 (10 cycles), two model 2 (100 cycles), two model 3 (150 cycles), and two model 6 (250 cycles) drones will all be launched at location 2. The drones must fly from their respective launch points to their assigned tasks, then must fly out to the base station after completing their assigned tasks.

Fig. 8. The Raspberry Pi 3B+s used in physical testbed.

We ran our case study problem input on our two proposed algorithms and the ILS baseline. Our BnM algorithm found an optimal assignment with a 91.67% chance of completing all tasks while the M+T algorithm found a near optimal assignment with a 90.96% chance of completing all tasks. The BM, ILS, MinDist, and VLSN approaches all trailed behind, finding solutions of 76.28%, 84.88%, 19.32% and 76.28%, respectively. The BnM algorithm finished in 120 ms while the M+T, BM, ILS, and MinDist algorithms all completed in under 10 ms. The VLSN algorithm took 30 ms. These results follow the same pattern we found previously and support our result analysis from the numerical simulations.

VII. Performance Evaluation in the Field

To better understand the impact of increased message passing in a realistic distributed environment, we deployed both the dM+ILS and dILS algorithms on a physical testbed. We again chose to run only the dILS baseline on the test bed because of its superior performance compared to the other baseline methods. Our physical testbed consisted of an array of eight Raspberry Pi 3B+ (one mounted on a drone and the other seven free standing) that we deployed in a large courtyard area shown in Fig. 8. The Raspberry Pi were arranged throughout the courtyard, varying between 5 and 50 meters apart, and communicated via WiFi using Ad-Hoc mode. We used Raspberry Pi 3B+ because they are commonly used as an onboard computer for drones.

We ran inputs from the second test set (where the number of drones ranges from four up to eight) on the testbed. Table V shows the computation time as the number of drones increases while Table VI shows the average ratio of the found solution for each algorithm over the best found solution (a ratio of 1 means that the algorithm found the best known solution). We again see the same trend, where the dM+ILS algorithm outperforms the baseline but at the cost of computation time. The time delay becomes exacerbated on the physical test bed where data packets must be transmitted over WiFi. As discussed in Section VI-C, we could improve the performance

TABLE V
Computation Time (seconds) on Testbed

# Drones	4	5	6	7	8
dILS	0.142	0.206	0.552	0.211	0.244
dM+ILS	0.961	1.742	2.804	4.412	4.785

TABLE VI
Found solution over best found solution on Testbed

# Drones	4	5	6	7	8
dILS	0.998	0.991	0.981	0.945	0.956
dM+ILS	0.999	0.998	0.999	0.999	0.996

of the algorithm by implementing a more efficient version of the distributed Hungarian method at the cost of data privacy.

VIII. Conclusions

In this paper we formulated the Maximum Probability of Task Success (MPTS) problem, a task assignment problem that seeks to maximize the probability that all tasks are complete. We applied the problem to a wildfire monitoring scenario and proposed both a centralized, optimal solution (Branch-and-Match) and a distributed approach (M+ILS) that finds near-optimal solutions in polynomial time. We evaluated our two proposed algorithms in extensive simulations and ran our M+ILS algorithm on a physical testbed. Our Branch-and-Match algorithm is computationally faster than the classic branch-and-bound algorithm but does not scale well with the number of drones. Our M+ILS algorithm averages solutions within 2.68% of the optimal solution but at the cost of time and message complexity when compared to other alternative baseline methods.

Future work should focus on further developing and evaluating methods for modeling the probability that a drone can complete a task based on environmental factors and onboard energy. This will require extensive field experiments on physical hardware. Furthermore, we made the assumption that the probability of each drone contributing to a task is an independent event. Removing this assumption could lead to new variation of the MPTS problem and should be considered in future work. We also see area for expanding the considered application, such as including wait times when drones move from one task to another or extending the problem to include cooperative teams of both drones and ground robots.

References

[1] [Online]. Available: https://www.nps.gov/articles/000/yosemite-washburn-fire-suppression.htm

[2] M. L. Fisher, R. Jaikumar, and L. N. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management science*, vol. 32, no. 9, pp. 1095–1103, 1986.

[3] W. Feller, *An introduction to probability theory and its applications*. John Wiley & Sons, 1991.

[4] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[5] G. Notomista, S. Mayya, S. Hutchinson, and M. Egerstedt, "An optimal task allocation strategy for heterogeneous multi-robot systems," in *18th European Control Conference (ECC)*. IEEE, 2019, pp. 2071–2076.

[6] S. P. Lloyd and H. S. Witsenhausen, "Weapons allocation is np-complete." in *1986 summer computer simulation conference*, 1986, pp. 1054–1058.

[7] A. C. Andersen, K. Pavlikov, and T. A. Toffolo, "Weapon-target assignment problem: Exact and approximate solution algorithms," *Annals of Operations Research*, vol. 312, no. 2, pp. 581–606, 2022.

[8] K. A. Ravindra, K. Arvind, C. Krishna, and B. James, "Exact and heuristic algorithms for the weapon-target assignment problem," *Operations Research*, vol. 55, no. 6, pp. 1136–1146, 2007.

[9] X. Chang, J. Shi, Z. Luo, and Y. Liu, "Adaptive large neighborhood search algorithm for multi-stage weapon target assignment problem," *Computers & Industrial Engineering*, vol. 181, p. 109303, 2023.

[10] T. Wang, L. Fu, Z. Wei, Y. Zhou, and S. Gao, "Unmanned ground weapon target assignment based on deep q-learning network with an improved multi-objective artificial bee colony algorithm," *Engineering Applications of Artificial Intelligence*, vol. 117, p. 105612, 2023.

[11] L. Khalid and A. Anpalagan, "Adaptive assignment of heterogeneous users for group-based cooperative spectrum sensing," *Transactions on Wireless Communications*, vol. 15, no. 1, pp. 232–246, 2015.

[12] J. Buergin, P. Blaettchen, C. Qu, and G. Lanza, "Assignment of customer-specific orders to plants with mixed-model assembly lines in global production networks," *Procedia CIRP*, vol. 50, pp. 330–335, 2016.

[13] M. Khoo, T. A. Wood, C. Manzie, and I. Shames, "A distributed augmenting path approach for the bottleneck assignment problem," *Transactions on Automatic Control*, vol. 69, no. 2, pp. 1210–1217, 2023.

[14] S. Giordani, M. Lujak, and F. Martinelli, "A distributed algorithm for the multi-robot task allocation problem," in *Trends in Applied Intelligent Systems: 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2010, pp. 721–730.

[15] S. Chopra, G. Notarstefano, M. Rice, and M. Egerstedt, "A distributed version of the hungarian method for multirobot assignment," *Transactions on Robotics*, vol. 33, no. 4, pp. 932–947, 2017.

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

[17] S. Friedman and Q. Han, "Request and share then assign (rasta): Task assignment for networked multi-robot teams," in *17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2020, pp. 418–426.

[18] A. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics (ACL), 2017, pp. 440–445.

[19] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[20] B. Zheng, O. Liu, J. Li, Y. Lin, C. Chang, B. Li, T. Chen, and H. Peng, "Towards a distributed local-search approach for partitioning large-scale social networks," *Information Sciences*, vol. 508, pp. 200–213, 2020.

[21] A. J. Smith, G. Best, J. Yu, and G. A. Hollinger, "Real-time distributed non-myopic task selection for heterogeneous robotic teams," *Autonomous Robots*, vol. 43, pp. 789–811, 2019.

[22] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," *Handbook of metaheuristics*, pp. 129–168, 2019.

[23] O. Pisacane, A. Attanasio, F. Guerriero, and R. Musmanno, "A general distributed framework based on iterated local search," in *International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*. IEEE, 2009, pp. 333–338.

[24] M. Levorato, L. Drummond, R. Figueiredo, and Y. Frota, "A distributed gpu-based correlation clustering algorithm for large-scale signed social networks," in *Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD)*. SBC, 2017, pp. 280–291.

[25] Y. Zeng, J. Xu, and R. Zhang, "Energy minimization for wireless communication with rotary-wing uav," *transactions on wireless communications*, vol. 18, no. 4, pp. 2329–2345, 2019.

[26] M. Valenti, B. Bethke, J. P. How, D. P. de Farias, and J. Vian, "Embedding health management into mission tasking for uav teams," in *American Control Conference*. IEEE, 2007, pp. 5777–5783.

[27] J. Diller, P. Hall, and Q. Han, "Holistic path planning for multi-drone data collection," in *19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, 2023, pp. 222–226.

[28] R. Raj and C. Murray, "The multiple flying sidekicks traveling salesman problem with variable drone speeds," *Transportation Research Part C: Emerging Technologies*, vol. 120, p. 102813, 2020.

[29] J. Diller and Q. Han, "Energy-aware uav path planning with adaptive speed," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2023, pp. 923–931.

[30] C. Love, A. Gaskins, and N. R. L. W. DC, "Performance loss of lithium ion polymer batteries subjected to overcharge and overdischarge abuse," *Memorandum Report, NRL/MR/6110-12-9455*, p. 1, 2012.

[31] M. Fernández and S. Williams, "Closed-form expression for the poisson-binomial probability density function," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 46, no. 2, pp. 803–817, 2010.

[32] (2023) Doi uas fleet. US Department of Inteiror. [Online]. Available: https://www.doi.gov/aviation/uas/fleet

[33] A. B. Asghar, G. Shi, N. Karapetyan, J. Humann, J.-P. Reddinger, J. Dotterweich, and P. Tokekar, "Risk-aware recharging rendezvous for a collaborative team of uavs and ugvs," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5544–5550.