Characterization of Mixed Integer Linear Programming for Multi Robot Task Allocation

A Major Qualifying Project submitted to the Faculty of WORCESTER POLYTECHNIC INSTITUTE in partial fulfillment of the requirements for the degree of Bachelor of Science in Robotics Engineering and Computer Science

Prepared by Omri Green

Friday, August 9th 2024

Report Submitted to:

Professor Carlo Pinciroli

Abstract

In large-scale scenarios such as search and rescue, the complexity and scale often necessitate the deployment of multi-robot systems (MRS) to achieve efficient task completion. Unlike individual robots, MRS can operate in parallel, reduce task complexity, and provide redundancy, making them essential for tackling problems that exceed the capabilities of a single robot. The effective operation of an MRS hinges on the optimal allocation of tasks among the robots, a process known as multi-robot task allocation (MRTA). This research focuses on MRTA within a homogeneous swarm of robots, modeled as a multiple traveling salesman problem (mTSP). We aim to minimize the total distance traveled by the swarm while ensuring that all tasks are completed. To achieve this, we employ a mixed integer linear programming (MILP) solver, which guarantees the optimal solution for MRTA. Our analysis shows that the MILP solver's processing time exhibits a super linear relationship with the number of robots and tasks, providing a valuable benchmark for comparing future algorithms.

Contents

List of Figures

1 Introduction

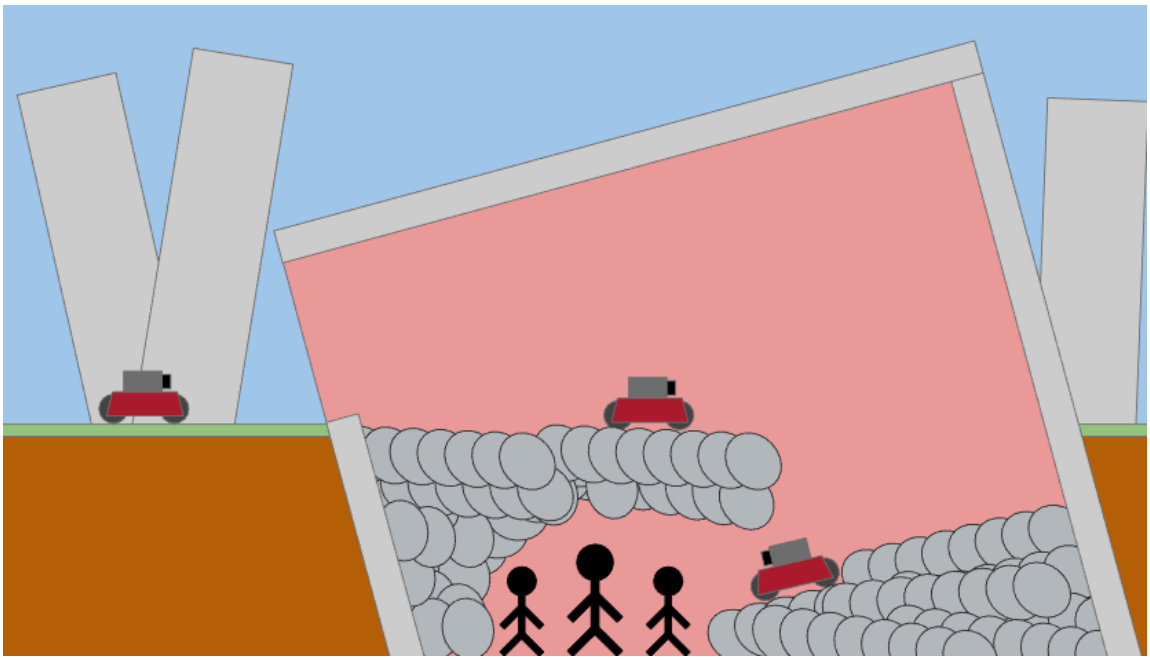


Figure 1.1: 3 Robots engaged in search and rescue in a collapsed building.

Robotic systems are used in solving a massive variety of real world problems at a large scale. Groups of robots have completed tasks ranging from building cars, inventorying warehouses, to even delivering packages to your door; however, for large scale scenarios such as search and rescue, using teams of robots is necessary to handle the scale and complexity of the scenario.

Consider a team of robots that is attempting to locate survivors in a collapsed building after an earthquake as depicted in figure 1.1. While in an ideal scenario one robot could do the entire task the area is too large to inspect using a single robot. The building's structure may be unstable, with debris and obstacles hindering movement, necessitating

the deployment of swarms of robots to cover different sections in a timely manner through multi-robot-task allocation (MRTA). This requires robots to use task scheduler to create the time-efficient schedules needed for operations.

1.1 Background

This scenario exemplifies a multi-robot system (MRS), commonly employed to tackle problems beyond the capabilities of individual robots [4]. MRS offers several advantages: components can work in parallel, drastically reducing task completion time; it provides redundancy, allowing tasks to be completed even if one robot fails; it reduces complexity, such as taking simultaneous pictures from different orientations; and it enables simpler design by distributing tasks among specialized robots [4]. However, for an MRS to be effective, each robot must be properly assigned tasks.

The process of assigning tasks within an MRS is known as multi-robot task allocation (MRTA) [2]. MRTA optimizes task assignment by maximizing or minimizing a specific objective function, ensuring efficient task distribution [2]. Often, MRTA is modeled as the multiple traveling salesman problem (mTSP), especially when the MRS is homogeneous and each task can be completed by any robot [4].

A common approach to solving the mTSP is through exact algorithms, which guarantee an optimal solution [6]. The mTSP is frequently modeled using mixed integer linear programming (MILP) [8]. An MILP solver will always find the optimal solution for MRTA if the problem can be represented as a convex graph. Given that MILP solvers are a standard method for MRTA, a wide range of software is available, making them an ideal benchmark for comparing other algorithms [8].

1.2 Problem Statement

In this research we address MRTA for a homogeneous swarm of k robots modeled as a mTSP. The swarm must complete t distinct tasks distributed across a $q \times q$ meter plane, starting from a designated point and ending at a fixed location. The goal is to minimize the total distance traveled by the swarm while ensuring all tasks are completed.

1.3 Contributions

To solve this problem, we employ a mixed integer linear programming (MILP) solver. This approach is particularly effective because the MILP solver will always find the most optimal solution.

We will characterize the MILP solver by the time it takes to compute a solution to the problem. This will allow us to characterize how long it takes for the swarm to calculate a solution to MRTA creating a benchmark for comparing other algorithms.

2 Related Works

In this chapter, we explore the foundational literature and methodologies essential for understanding the characterization of a mixed integer linear programming (MILP) scheduler within the framework of multi-robot task allocation (MRTA). MRTA is a critical area of research focused on the efficient distribution of tasks among multiple robots, making it a fundamental problem in robotics and autonomous systems. A key aspect of understanding MRTA involves studying the multiple traveling salesman problem (mTSP), a complex variant of the traveling salesman problem (TSP), which directly informs the strategies for task allocation in multi-robot systems.

This chapter will review the various approaches to solving the mTSP, including exact algorithms, heuristic methods, and problem transformations, as these solutions provide crucial insights into effective MRTA strategies. We will also delve into the principles of MILP, examining how it serves as a powerful optimization tool for solving the mTSP and, by extension, MRTA. This discussion will emphasize MILP's potential to deliver optimal solutions under the constraints typically encountered in multi-robot scenarios.

2.1 The Multiple Traveling Salesman Problem (mTSP)

The multiple traveling salesman problem (mTSP) is an extension of the traveling salesman problem (TSP), where multiple salesmen must start and end their routes at specified nodes while visiting every node on the map as efficiently as possible [1]. This problem has diverse applications, from print press scheduling to mission planning for autonomous vehicles [1]. Various approaches can be used to solve the mTSP.

One approach is using exact algorithms, which solve the mTSP directly[1]. Each decision made by a salesman incurs a fixed cost, activated when the decision is executed. These algorithms often utilize the branch-and-bound technique, allowing the program to evaluate all feasible solutions while adhering to the problem's constraints, ensuring that the optimal solution is found [1].

Another approach is to use heuristic algorithms, ranging from neural networks to parallel processing [1]. While exact algorithms guarantee the best solution for a specific mTSP, heuristic solutions are more generalized and can solve various mTSP problems with

comparable performance.

A third approach is transforming the mTSP into a standard TSP [1]. Given that TSP is one of the most studied problems in computer science, there are numerous algorithms available to solve it after transformation [6].

Compared to TSP, mTSP is better suited for modeling real-world scenarios such as multi-robot task allocation (MRTA) [1]. Depending on the situation, there may be fixed or variable starting and ending locations [1], time windows within which tasks must be completed, and various other constraints to accurately model a scenario [1].

When applying mTSP to MRTA, several common conditions and requirements are imposed on the robots and tasks. Some robots may only be capable of executing one task at a time, while others can handle multiple tasks simultaneously [2]. Tasks may require either a single robot or multiple robots for completion [2]. Additionally, tasks may need to be performed in a specific order, or the order may be irrelevant [2]. These conditions can be classified as follows [2]:

- No Dependencies (ND): The cost of a given robot-task pair is independent of all others.
- In-Schedule Dependencies (ID): The cost of a given robot-task pair depends on other tasks assigned to the same robot.
- Cross-Schedule Dependencies (XD): The cost of a given robot-task pair depends on tasks assigned to other robots.
- Complex Dependencies (CD): The cost of a robot-task pair depends on other schedules that have their own dependencies.

2.2 Mixed Integer Linear Programming (MILP)

Mixed integer linear programming (MILP) is a mathematical formalism for the expression of optimization problems composed of linear inequalities, or constraints, and a linear objective function to optimize with the restriction that some, but not all, of the variables must have integer values [6]. These variables can also be binary if the integer decision variable is limited to values 0 and 1[6]. A standard mixed-integer formula usually takes the form of an equation similar to equation 2.1 where z^* represents an ideal solution, c represents a cost vector, and x represents a decision vector[1];

$$z^* = min\ c^T \cdot x, x \in X \tag{2.1}$$

this equation is typically subject to linear constraints as seen in equation 2.2 where A is a matrix made up of equality or inequality rules and b is what $A \cdot x$ should be less than or equal to, equal to, or greater than or equal to [1];

$$A \cdot x(<, =, or >)b \tag{2.2}$$

there are also upper and lower bounds which limit the size of variables an example of which is shown in equation 2.3, where l represents the lower bound and u represents the upper bound that the decision vector x must be between [1].

$$l \leq x \leq u \tag{2.3}$$

These equations make an MILP problem find the best objective feasible solution called the optimal solution assuming the problem can be represented as a convex graph[6]; if the problem cannot be represented through a convex graph, it cannot always find the optimal solution, instead it will find the local maxima or minima which may or may not be the optimal solution. When applied to the mTSP this makes it an exact algorithm which will always find the best solution to the problem[1].

3 Methodology

In this paper, we will be minimizing the total distance traveled by the entire swarm of robots with the assumption that the robot swarm is homogeneous, all of the tasks only need to be visited once by a single robot, and that the cost for a robot taking a route between two tasks is equal to the distance between them. To determine how the amount of tasks and robots effect the time to process a solution to the mTSP using MILP we have created the formulation below.

3.1 Problem Formulation

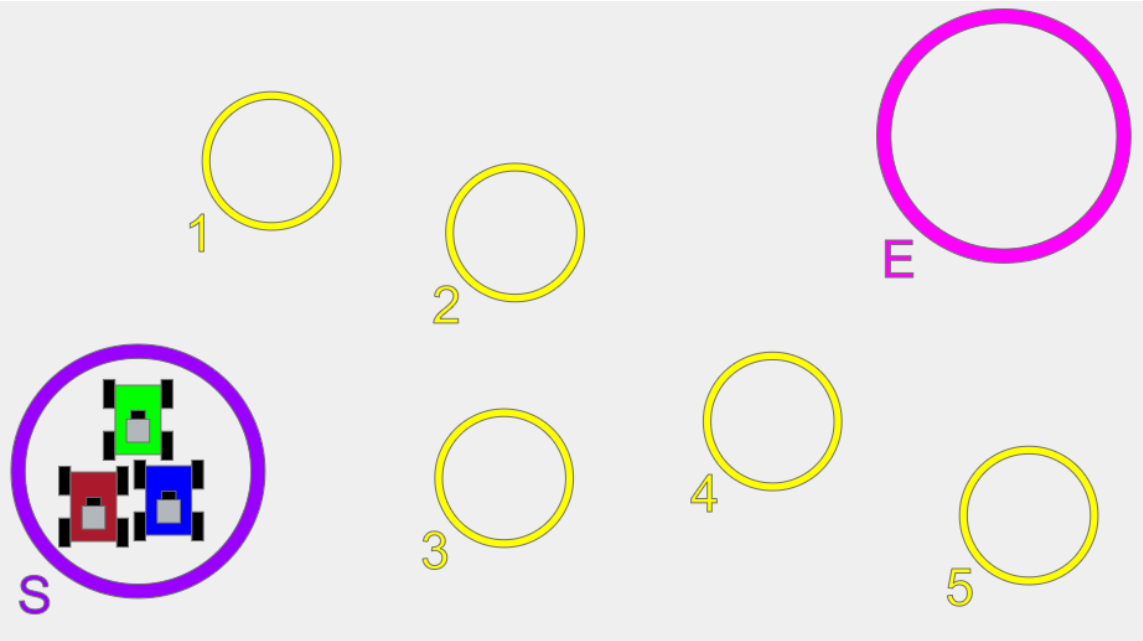


Figure 3.1: 3 robots in an unsolved environment with 5 tasks

To represent the environment in which the MILP algorithm operates in, we will be using a qxq meter plane. There will be an initial point from which the swarm starts, the start task, represented by s, there will be t tasks, which are each represented by a numeric id, that need to be completed by one robot, and finally when a robot finishes its assigned tasks it will then got to the end task which is represented by e. Additionally the number of robots is represented by k. An example environment is shown in figure 3.1.

The objective of the MILP algorithm is to assign a schedule to each robot that completes all the tasks with the minimum total movement. All robots must complete all tasks initially starting at the start task and stopping at the end task. This is shown in figure 3.2.

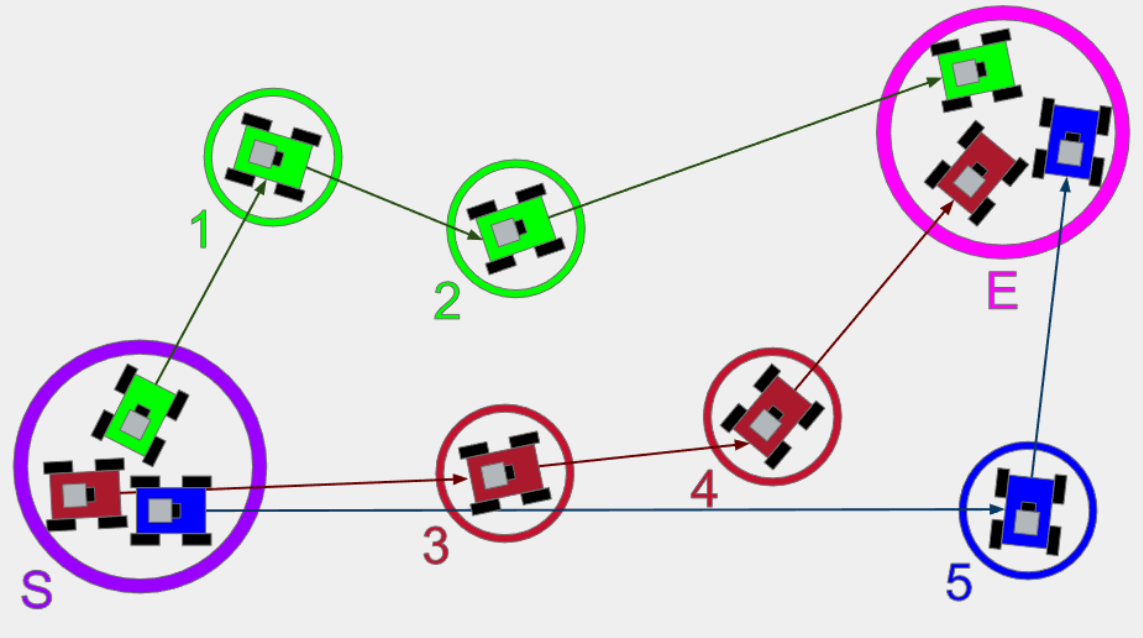


Figure 3.2: 3 robots in a solved environment with 5 tasks

3.2 Approach

3.2.1 Constraints

For a swarm of robots to complete all their assigned tasks, we create a cost tensor $\mathbf{C}$ consisting of k matrices where each matrix is assigned to an individual robot with ID i. Each of these matrices contains the distance between any two of the n tasks where j represents the task ID, j_1 represents the task a robot is leaving from, and j_2 represents the task the robot is going to. Since we are assuming symmetry in this problem, $c_{i,j_1,j_2} = c_{i,j_2,j_1}$. We are also assuming that there is a consistent start and end task which are represented by s and e respectively; additionally, the last non-consistent task is represented by $t = n - 2$. The set of tasks can be presented as the following $\{s, 1, \cdots, t, e\}$.

In order to represent the path each of the robots will take, we created the binary decision tensor $\mathbf{X}$ with each of the k matrices assigned to an individual robot with ID i. Each of these matrices contains the path its assigned robot travels. Just like in the cost tensor $\mathbf{C}$, travel between any of the n tasks is represented by the task it is leaving (j_1) and the task it is going to (j_2) with an identical set of tasks ($\{s, 1, \cdots, t, e\}$). If a robot has taken route $x_{i,j_1,j_2} = 1$; otherwise $x_{i,j_1,j_2} = 0$. To represent a valid path, all values in the decision tensor $\mathbf{X}$ must abide by the following constraints.

$$\text{Only exit a task once: } \sum_{j_2} x_{i,j_1,j_2} \leq 1, \forall i, j_1 \in \{s, 1, \cdots, t\} \tag{3.1}$$

$$\text{Only enter a task once: } \sum_{j_1} x_{i,j_1,j_2} \leq 1, \forall i, j_2 \in \{1, \cdots, t, e\} \tag{3.2}$$

$$\text{Leave } s \text{ and enter } e\text{: } \sum_{n} x_{i,e,n} + x_{i,n,s} \leq 1, \forall i \tag{3.3}$$

$$\text{Cannot enter } s\text{: } \sum_{j_1} x_{i,j_1,s} = 0, \forall i \tag{3.4}$$

$$\text{Cannot exit } e\text{: } \sum_{j_2} x_{i,e,j_2} = 0, \forall i \tag{3.5}$$

$$\text{Leave all entered tasks except } e\text{: } \sum_{n} x_{i,j,n} - x_{i,n,j} = 0, \forall i, j \in \{1, \cdots, t\} \tag{3.6}$$

$$\text{Cannot reenter tasks: } \sum_{j} x_{i,j,j} = 0, \forall i \tag{3.7}$$

To track which tasks robots have traveled through, we have created the binary tracking matrix $\mathbf{Y}$ consisting of k vectors where every vector is assigned to an individual robot with ID i. Each of the t values represents the completion status of every task ID with the exceptions of the start task (s) and the end task (e) since they are always visited. If a task with ID j has been visited by a robot with the ID i, $y_{i,j} = 1$; otherwise $y_{i,j} = 0$. A valid tracking matrix requires the following constraints to be fulfilled.

$$\text{One robot per task except } e \text{ and } s\text{: } \sum_{i} y_{i,j} = 1, j \in \{1, \cdots, t\} \tag{3.8}$$

$$\text{Track all visited tasks: } y_{i,j_2} = \sum_{j_1} x_{i,j_1,j_2}, \forall i, j_2 \in \{1, \cdots, t\} \tag{3.9}$$

To track the set of paths a swarm of robots will travel down, we created the tour matrix $\mathbf{Z}$ consisting of k arrays each assigned to a robot ID i. Each of these arrays has a length of n where every value represents a task ID (j) representing the task a robot could travel through. If a task is visited, $z_{i,j} \geq 1$; otherwise $z_{i,j} = 0$. In order to track the order in which a robot visits tasks and to avoid subtours, there is an energy flow, so if a robot goes from task j_1 to task j_2, $z_{i,j_1} - z_{i,j_2} = 1$. To create a valid tour matrix, the following constraints must be true.

$$Z_i \text{ doesn't track tasks robot i has not visited: } z_{i,j} - n * y_{i,j} \leq 0, \forall i, j \in \{1, \cdots, t\} \tag{3.10}$$

$$Z_i \text{ tracks the tasks robot i visited: } y_{i,j} - z_{i,j} \leq 0, \forall i, j \in \{1, \cdots, t\} \tag{3.11}$$

$$s\text{'s energy is the highest in } Z_i\text{: } z_{i,s} = n, \forall i \tag{3.12}$$

$$e\text{'s energy is depleted by visiting tasks in } Z_i\text{: } z_{i,e} = z_{i,s} - \sum_{j} y_{i,j}, \forall i \tag{3.13}$$

3.2.2 Implementation

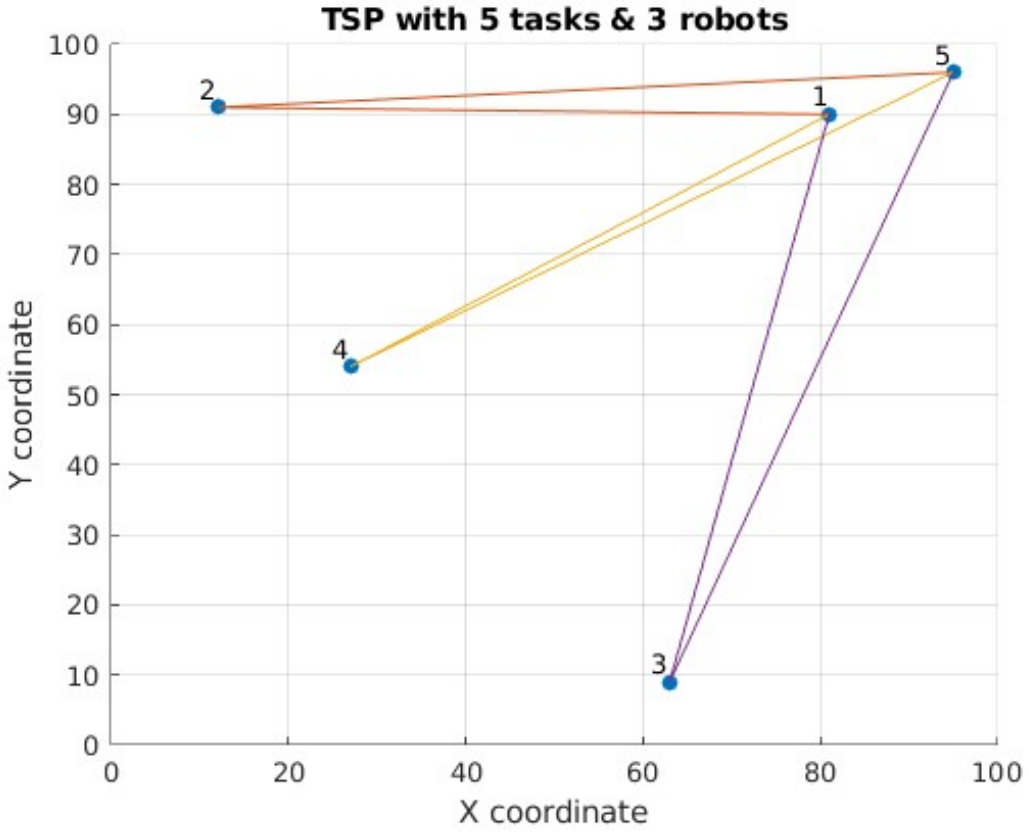


Figure 3.3: A solved mTSP problem with 5 total tasks including the start and end tasks with 3 robots.

The first step is vectorizing all of the tensors and matrices $(\mathbf{C}, \mathbf{X}, \mathbf{Y}, \mathbf{Z})$. Then the vectorized decision tensor $\vec{\mathbf{X}}$, tracking matrix $\vec{\mathbf{Y}}$, and tour matrix $\vec{\mathbf{Z}}$ is combined into the decision vector $\vec{\omega} = \{\vec{\mathbf{X}}, \vec{\mathbf{Y}}, \vec{\mathbf{Z}}\}$. Afterwards, the cost vector $\vec{c}$ will be created using the linearized cost tensor and then extended using 0s to reach the same dimensionality as the decision vector $(\vec{\omega})$. Therefore, $\vec{c} = \{\vec{\mathbf{C}}, 0, \cdots, 0\}$. Lastly, we solve for the optimal minimization of the total distance for the swarm to travel in the equation below.

$$\min_{\omega} c^T \omega \tag{3.14}$$

An example of a mTSP solved by the MILP algorithm 3.14 with the constraints in the constraints section is in Figure 3.3.

4 Experimental Evaluation

In this section, we will be going over how we determined the effect of the amount of tasks and robots on the time to process a solution to the mTSP using MILP; additionally, we will go over the metrics used for experimentation, the experimental setup, and we will finish this section by discussing the results.

4.1 Metrics

In order to evaluate the effect of the amount of tasks and robots on the time to process a solution to the mTSP using MILP, we will be using the inputs of Q, T, K, and R which respectively represent the height and width of the experimental plane, the total number of tasks not including the start and end task, the total number of robots, and the total amount of experiment repetitions for every combination of T and K which are represented by t and k respectively. The metrics used for this experiment are the following:

Average Calculation Time

- Measured in seconds
- Indicates the average calculation time across 10 unique tests with t tasks, and k robots.

Standard Deviation of Calculation Time

- Measured in seconds
- Indicates the standard deviation calculation time across 10 unique tests with t tasks, and k robots.

By utilizing these two metrics we can analyze the effect of the amount of tasks and robots on the time to process a solution to the mTSP using MILP, by using 2 heatmaps of size $t \times k$ to visually analyze the effect of the number of tasks and robots on the average calculation time for finding a solution to the mTSP using MILP.

4.2 Experimental Setup

All variations of the mTSP in this experiment were completed in MATLAB using the following parameters:

Experiment Plane Dimensions

- $Q = 50$
- All mTSP problems took place on a 50×50 meter plane

Maximum Number of Tasks

- $T = 10$
- All variations of the mTSP in the experiment will contain 1 to 10 different tasks outside of the start and end task
- The number of tasks in each mTSP variation is represented by t

Maximum Number of Robots

- $K = 10$
- All variations of the mTSP in the experiment will contain a homogeneous swarm of 1 to 10 robots
- The number of robots in each mTSP variation is represented by k

Number of Repetitions

- $R = 10$
- All variations of the mTSP are repeated 10 times with randomly generated environments

Using the parameters above it is possible to generate the data necessary to calculate the average calculation time and standard deviation of calculation time for all 100 potential combinations of t and k using the methods in data generation. To analyze this data it is turned into a variety of heatmaps that are explained in the data analysis section 4.2.2.

4.2.1 Data Generation

To generate the data needed for later analysis, the code was divided into two parts: environment generation and runtime measurement. Environment generation creates a random environment for every variation of t and k and for every repetition of the mTSP, while runtime measurement calculates the average calculation time and the standard deviation of calculation time for each variation of the mTSP.

Environment Generation

The reasoning behind the randomization for every single repetition is an attempt to reduce the significance of outliers. If there is a mTSP environment that is significantly easier or harder for the MILP solver to compute, it will not be repeated, reducing the effect of outliers. The algorithm used is in Algorithm 1:

Algorithm 1: Environment Generation

Input:
T: Number of Tasks not Including Start and End tasks
Q: Environment Dimensions
Output:
p: Task Array

$T \leftarrow T + 2$; `// total number of all tasks`
$p \leftarrow []$; `// task array initialization`
`// Generates Random Task Locations`
for $i \leftarrow 0$ *upto* t **do**
  $x \leftarrow U(0, Q)$; `// Generates a random x coordinate`
  $y \leftarrow U(0, Q)$; `// Generates a random y coordinate`
  p.append$((x, y))$; `// Add the (x, y) coordinates to the task array`
end

Metric Measurement

In order to measure the average calculation time and the standard deviation of calculation time for the MILP solver defined in equation 3.14 to solve the variations of mTSP defined above the same. Algorithm 2 is used to generate the 2 CSV files that are used for Data Analysis.

Algorithm 2: Metric Measurement

Input:

T: Number of Tasks not Including Start and End tasks
Q: Environment Dimensions
K: Total Number of Robots
R: Total Number of Repetitions

Output:

N/A: CSV Files of the data are exported

```
T = T + 2 ;                                   // total number of all tasks
stdevMatrix ← [[]] ;            // Matrix used to make the stdev CSV file
aveMatrix ← [[]] ;                // Matrix used to make the ave CSV file
// Goes Through Every Combination of t and k
for k ← 1 upto K do
    stdevList ← [];           // Stores stdev data for each variation of k
    aveList ← [];               // Stores ave data for each variation of k
    for t ← 1 upto T do
        data ← [];  // Used to calculate stdev and ave for each variation
        // Goes through repetitions to ensure generalization of results
        for t ← r upto R do
            taskList ← environmentGeneration(t, Q);      // Runs Algorithm 1
            cost ← calcCost(taskList);                // Calculates Cost Tensor
            startTime ← Current Time ;              // Current Time in Seconds
            solution ← milpSolver(cost, t, k) ;                 // Equation 3.14
            endTime ← Current Time ;                // Current Time in Seconds
            data.Append(startTime − endTime) ;     // Adds Runtime to data
        end
        stdevList.Append(stdev(data)) ; // Adds stdev of data to stdevList
        aveList.Append(ave(data)) ;  // Adds average of data to stdevList
    end
    stdevMatrix.Append(stdevList) ;           // Adds stdevList for each k
    aveMatrix.Append(aveList) ;                 // Adds aveList for each k
end
exportCSV(name = "stdev.csv", data = stdevMatrix) ;   // Exports CSV file
exportCSV(name = "ave.csv", data = aveMatrix) ;       // Exports CSV file
```

4.2.2 Data Analysis

To assess the impact of the number of tasks and robots on both the average calculation time and the standard deviation of calculation time for solving the mTSP, the data will be analyzed using heatmaps. Since the two exported CSV files are of identical size, their respective heatmaps can be generated using the same code. We will create six different heatmaps: three for the average calculation time and three for the standard deviation. These heatmaps will illustrate the overall effect, the effect of the number of tasks, and the effect of the number of robots on the calculation time for solving the mTSP with an MILP-based solver.

Overall Effect

To assess the overall effect the CSV data from both the standard deviation and average of the calculation time is simply input into a function in R.

Effect of the Number of Tasks

To find the effect of the number of tasks on the average and standard deviation of the calculation time on solving the mTSP with a MILP solver we have normalized the data for every single robot to eliminate the effect that the number of robots have on the data. This data is then put into a heatmap function.

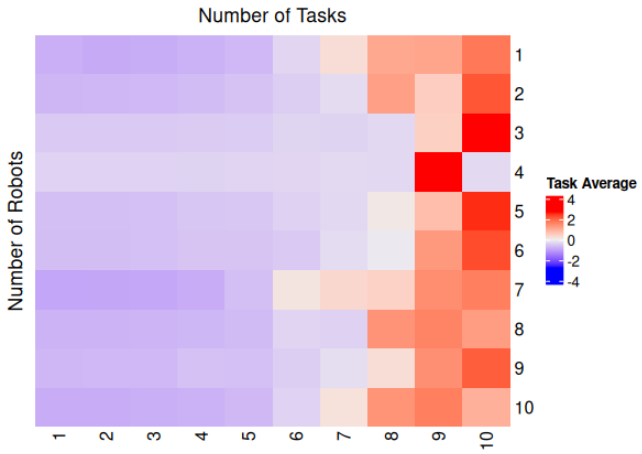
Effect of the Number of Robots

To find the effect of the number of robots on the average and standard deviation of the calculation time on solving the mTSP with a MILP solver we have normalized the data for every single task to eliminate the effect that the number of tasks have on the data. This data is then put into a heatmap function.

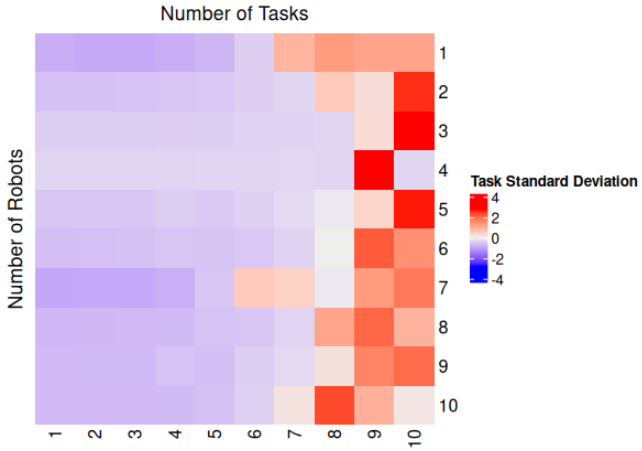
4.3 Discussion

In this section we will be discussing the results from our testing, we will be discussing the overall effect of the following on the calculation time of solving the mTSP with a MILP solver; the number of tasks, and the number of robots, and both.

4.3.1 Number of Tasks



(a) Effect of the number of tasks on the calculation time of solving the mTSP using a MILP solver



(b) Effect of the number of tasks on the calculation time of solving the mTSP using a MILP solver

Figure 4.1: Effect of the number of tasks on calculation time.

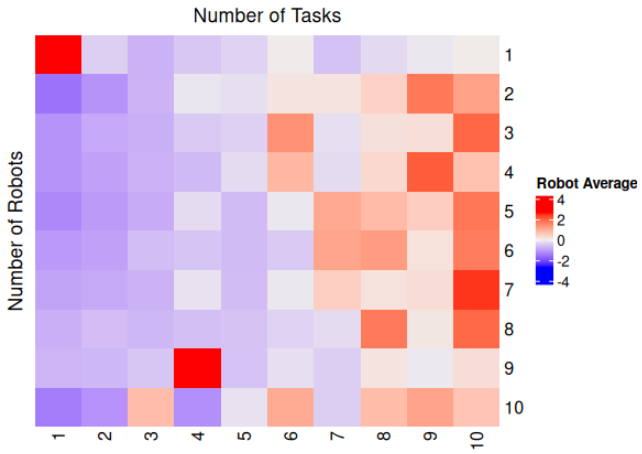
When analyzing the task average in Figure 4.1.a, a clear pattern emerges: as the number of tasks increases, the calculation time grows super linearly. Notably, in both MRTA scenarios with 1 and 10 robots, the normalized calculation time appears to be identical as the number of tasks increases. This similarity is likely due to the super linear increase in calculation complexity relative to the number of tasks. Additionally, an outlier is observed in the MRTA scenario with 4 robots and 9 tasks.

The analysis of the task standard deviation in Figure 4.1.b further reinforces the previous findings. As calculation time increases, so does the standard deviation. This

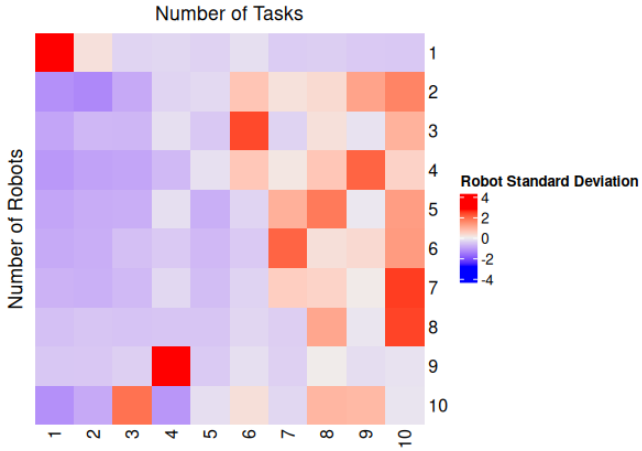
trend is underscored by the presence of an outlier in the mTSP scenario with 4 robots and 9 tasks. Similarly, in the mTSP scenarios with 1 and 10 robots, an outlier is observed in the scenario with 10 robots and 8 tasks. However, this appears to be an exception that confirms the general pattern: the standard deviation remains low until the number of tasks reaches 6, supporting the initial observation that as calculation time increases, the standard deviation follows suit.

The combined analysis of both heat maps suggests a super linear relationship between the number of tasks and the computation time required for the MILP solver to find the optimal solution for the mTSP.

4.3.2 Number of Robots



(a) Effect of the number of robots on the calculation time of solving the mTSP using a MILP solver



(b) Effect of the number of robots on the calculation time of solving the mTSP using a MILP solver

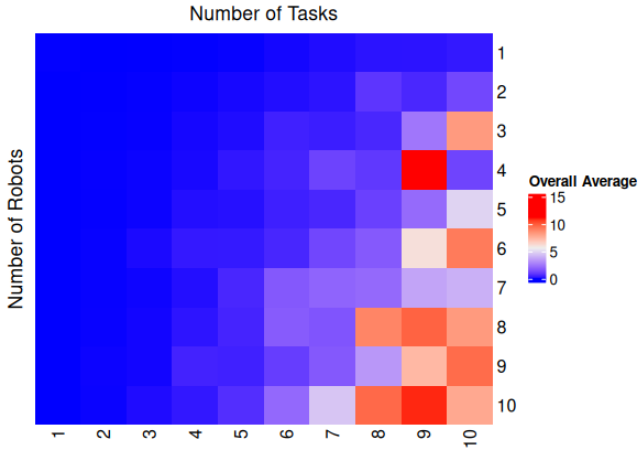
Figure 4.2: Effect of the number of robots on calculation time.

Figure 4.2.a shows a clear pattern as the number of robots increases there is a super linear increase in the average calculation time of solving the mTSP using an MILP solver. Unlike while analyzing tasks there doesn't seem to be any inflection point or anything with identical data. While there seems to be an outlier with 1 robot and one task, the rest of the data with 1 robot seems to confirm the rule.

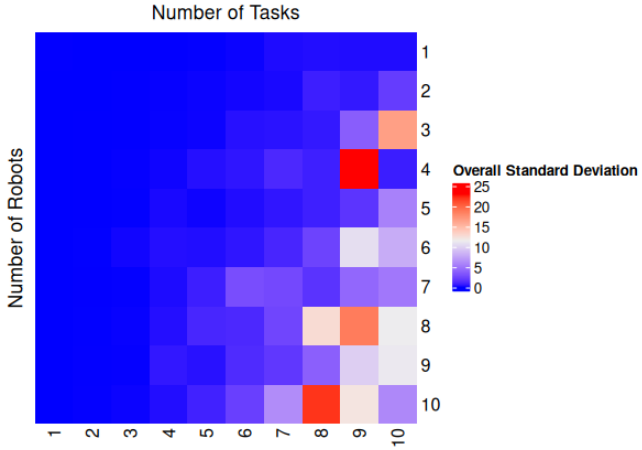
The data from figure 4.2.b seems to confirm the observations from the previous figure. As the number of robots increases super linearly so does the normalized standard deviation of the calculation time.

We believe that this is because as the number of robots increases, the amount of calculations needed for the MILP solver to find an ideal solution to the mTSP increases super linearly.

4.3.3 Number of Tasks and Robots



(a) Effect of the number of robots and tasks on the calculation time of solving the mTSP using a MILP solver



(b) Effect of the number of robots and tasks on the calculation time of solving the mTSP using a MILP solver

Figure 4.3: Effect of the number of robots on calculation time.

Figure 4.3.a shows that as the number of robots and tasks increase so does the average time it takes for the MILP solver to find the best solution for a mTSP for the number of robots and tasks. For individual robots the increase of calculation time seems to increase super linearly after an inflection point, with 10 robots it seems to be with 5 tasks, with 9 robots it seems to be at 4 tasks, and so on. Additionally, if we analyze the location of these "inflection points" as the number of robots seems to affect it super linearly.

These findings are reflected in figure 4.3.b, with similar patterns with "inflection points" having a location based on the number of robots in the mTSP problem. After each inflection point the standard deviation seems to increase super linearly as well.

Figures 4.3.a and 4.3.b indicate that the average calculation time for the MILP formulation to find an ideal solution to the mTSP seems to increase super linearly when compared to the volume of the Decision and Cost Tensors. This finding is reflected by the findings in sections 4.3.2 and 4.3.1 which find the effect of the number of robots and number of tasks on the average calculation time respectively.

5 Conclusions

5.1 Summary

This work presents an MILP based scheduler for homogeneous teams of robots of any size that are meant to complete an arbitrary number of tasks. The MILP scheduler minimizes the overall distance that the team travels to complete all tasks. The scheduler was characterized by the time it takes to complete a solution based on new data, simulating what a single robot would calculate upon encountering a new obstacle.

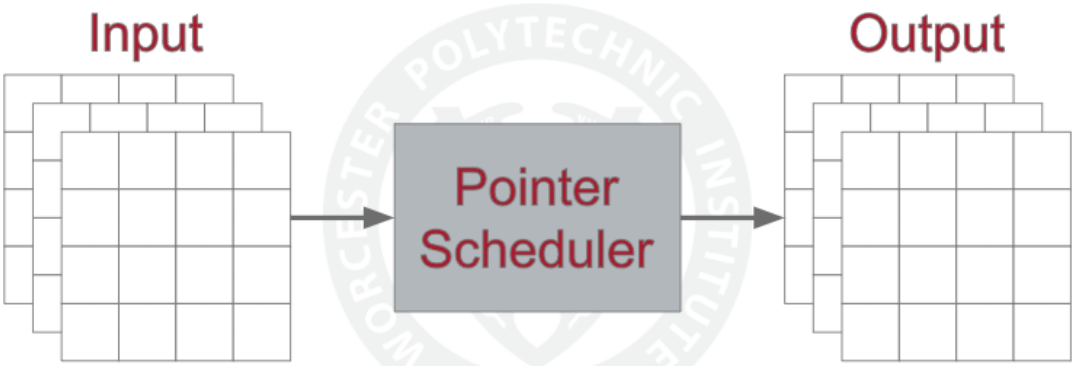
In regards to characterizing the MILP scheduler we utilized heatmaps to allow for a visual analysis of the data. Through this analysis we arrived at the conclusion that the time to complete a solution for a mTSP has a super linear relationship to both the number of robots and the number of tasks. Essentially, the MILP scheduler we have created does not scale up well, it gets far more complex as the number of tasks and robots involved increases.

5.2 Future Work

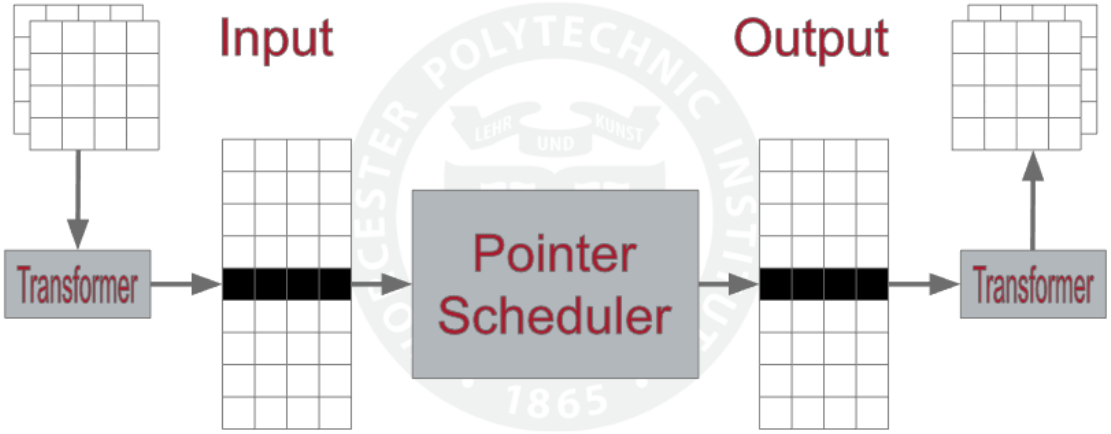
In this paper we have analyzed the MILP scheduler by creating a heatmap to determine an approximate relationship between the number of tasks and robots and the time to complete a solution to a mTSP. In future work we will instead determine an exact relationship between the number tasks, the number of robots, and the calculation time for the MILP scheduler to solve a mTSP problem. In order to do this we will plot the data as a three dimensional scatter plot, then we will create a numerical model of the relationship between the number tasks, the number of robots, and the calculation time for the MILP scheduler to solve a mTSP problem.

We will also implement a neural network based scheduling algorithm based off the pointer network architecture; the networks works by reordering discrete tokens corresponding to positions in an input sequence to a desired solution[7]; to use pointer networks for the mTSP we have come up with two different approaches to the network's architecture [7]; the first option is a multi-layer algorithm which uses a set of four dimensional tuples as inputs and outputs. These components are in order, the task ID, the robot ID, the X

coordinate, and the Y coordinate. This approach is shown visually in figure 5.1a. The other option is to use a transformer and what we call separator tuples, these values are meant to separate each robot's schedule in the output,which allows us to theoretically create a single layer algorithm similar to the one used in the original paper [7]. The way this method would work is to rely on the Pointer Networks inherent property of reordering its inputs, this allows us to separate the values. By using two transformers we can encode a four dimensional tuple as a three dimensional tuple and reverse the process as shown in figure 5.1b. To train a pointer network based scheduling algorithm you will also need a loss function, such as the traditional mean squared error loss function or a novel loss function called smart predict then optimize (SPO) [3].



(a) A graphical depiction of a multi-layer pointer-scheduling-algorithm



(b) A graphical depiction of a transformer based pointer-scheduling-algorithm

Figure 5.1: Pointer network based scheduling algorithm architecture proposals

Training a pointer network based scheduling algorithm using the SPO framework requires a defined SPO loss function, which quantifies the decision error induced by a prediction [3]. This function comprises two components: c, representing the actual cost of a feature, and $\hat{c}$, representing the predicted cost of a feature. A simplified representation of the SPO loss function is presented below where $\hat{c}$ represents the cost of a calculated solution and c represents the cost of an ideal solution [3].

$$l_{SPO}(\hat{c}, c) = \hat{c} - c \tag{5.1}$$

To compute the loss, each feature is denoted by a decision vector, whether it's the ideal decision vector $\vec{\omega_s}$ or the predicted decision vector $\hat{\vec{\omega}}$. The cost of a feature $c(\vec{\omega}, \vec{c})$ is determined by the dot product of the decision vector ($\vec{\omega_s}$ or $\hat{\vec{\omega}}$) and the cost vector $\vec{c_s}$ which is shown in equation 5.2 [3].

$$c(\vec{\omega}, \vec{c}) = \vec{\omega} \cdot \vec{c} \tag{5.2}$$

By substituting the cost calculation into l_{SPO}, we derive the following equation for SPO loss for a pointer network.

$$l_{SPO}(\vec{c_s}, \vec{\omega_s}, \hat{\vec{\omega}}) = c(\hat{\vec{\omega}}, \vec{c}) - c(\vec{\omega_s}, \vec{c}) \tag{5.3}$$

In future work we propose comparing the MILP scheduler, the two types of pointer network based scheduling algorithm architectures using a mean squared error loss function and one using a SPO based loss function. They will be compared by using the same measurement method we have proposed to further expand the analysis of the MILP scheduler. All algorithms will be tested on mTSP problems with a range in the number of tasks and robots that need to be solved and be evaluated based on the time it takes the algorithms to solve them.

5.3 Lessons Learned

While writing this paper we have learned multiple lessons that will help us in future work.

One key lesson was the importance of having a clear structure to base the paper off of. Creating an outline with defined sections and subsections helped us maintain our focus while writing the paper facilitating a more logical flow of ideas and ensuring that each section built on the previous one.

We have also learned of the importance of explaining previous works; Without proper explanation, readers may struggle to understand the behavior of the algorithms used in this work. By providing clear explanations, we have made our paper more accessible to a wider audience.

Finally, we have learned about the importance of time management and having realistic milestones. Writing a research paper is incredibly time extensive and requires careful planning and regular progress assessments. Establishing clear deadlines helped in managing the workload.

In summary, in the process of writing this paper we have gained valuable insights into the importance of structure, explaining previous works, and time management. These lessons will help us to improve the efficiency and quality of future research.

Bibliography

[1] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[2] Hamza Chakraa, François Guérin, Edouard Leclercq, and Dimitri Lefebvre. Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art. *Robotics and Autonomous Systems*, 168:104492, 2023.

[3] Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize", 2020.

[4] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative robots and sensor networks 2015*, pages 31–51, 2015.

[5] Jaynta Mandi, Emir Demirović, Peter. J Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems, 2019.

[6] Bochra Rabbouch, Hana Rabbouch, Foued Saâdaoui, and Rafaa Mraihi. Chapter 22 - foundations of combinatorial optimization, heuristics, and metaheuristics. In Seyedali Mirjalili and Amir H. Gandomi, editors, *Comprehensive Metaheuristics*, pages 407–438. Academic Press, 2023.

[7] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.

[8] Chaoqun Wang, Dan Mei, Yu Wang, Xiwen Yu, Wen Sun, Dong Wang, and Junquan Chen. Task allocation for multi-auv system: A review. *Ocean Engineering*, 266:112911, 2022.