ABSTRACT

Title of Thesis: DECENTRALIZED MULTIAGENT METAREASONING APPLICATIONS IN TASK ALLOCATION AND PATH FINDING

Samuel T. Langlois,
Master of Science, 2021

Thesis Directed By: Professor Jeffrey W. Herrmann
Department of Mechanical Engineering

Decentralized task allocation and path finding are two problems for multiagent systems where no single fixed algorithm provides the best solution in all environments. Past research has considered metareasoning approaches to these problems that take in map, multiagent system, or communication information. None of these papers address the application of metareasoning about individual agent state features which could decrease communication and increase performance for decentralized systems.

This thesis presents the application of a meta-level policy that is conducted offline using supervised learning through extreme gradient boosting. The multiagent system used here operates under full communication, and the system uses an independent multiagent metareasoning structure.

This thesis describes research that developed and evaluated metareasoning approaches for the multiagent task allocation problem and the multiagent path finding problem. For task allocation, the metareasoning policy determines when to run a task allocation algorithm. For multiagent path finding, the metareasoning policy determines which algorithm an agent should use.

The results of this comparative research suggest that this metareasoning approach can reduce communication and computational overhead without sacrificing performance.

DECENTRALIZED MULTIAGENT METAREASONING APPLICATIONS IN TASK ALLOCATION AND PATH FINDING

by

Samuel T. Langlois

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2021

Advisory Committee:
Professor Jeffrey Herrmann, Chair, Advisor
Assistant Professor Michael Otte
Assistant Professor Mumu Xu

Dedication

I would like to dedicate this work to my parents, Steven and Karla Langlois, their support and comfort are something I treasure. They have given so much of their time to me and for that I am grateful.

Acknowledgements

I could not have completed this thesis without the guidance and help of others. The assistance and support always encouraged my work and for that I am very grateful.

First, I would like to thank my advisor, Dr. Jeffrey W. Herrmann, for giving me the opportunity to work with him on a very interesting, complex topic. In a year where the world experienced a global pandemic, he adapted and guided me through the change in direction to online work. I don't think I could have successfully completed this without him. His comments and suggestions were vital to the understanding of the field and this thesis. It has been an honor and pleasure to work with Dr. Herrmann. Along with Dr. Hermann, I also want to thank Dr. Otte and Dr. Xu for being a part of the thesis committee.

My research colleagues, Oghenetekevwe Akoroda and Sharan Nayak, deserve an enormous thanks for help in understanding multiagent metareasoning and for the execution of the simulations used in this thesis.

My family has always been a source of inspiration and support. I especially want to acknowledge their encouragement to step out and follow my own path. I want too also acknowledge my confidante, Michelle Oller. Her compassion and understanding during this last year were important in aiding the thought process throughout this work.

Lastly, I would like to acknowledge the financial support from Northrup Grumman.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CBAA – Consensus Based Auction Algorithm

CBAAMM – Consensus Based Auction Algorithm Minimized Maximum Objective

DHBA – Decentralized Hungarian Based Algorithm

DHBAMM - Decentralized Hungarian Based Algorithm Minimized Maximum Objective

XGBoost – Extreme Gradient Boosting

*LRA** - Local Repair A*

*WHCA** - Windowed Hierarchical Cooperative A*

CBS – Conflict Based Search

MAPF – Multiagent Path Finding

*EPEA** - Enhanced Partial Expansion A*

ICTS – Increasing Cost Tree Search

MA-CBS – Meta-Agent Conflict Based Search

SWaP – Size, Weight, and Power

Chapter 1: Introduction

Rapid advancement of technologies in communications and robotics has increased the applicability of multiagent systems in industrial settings. Such applications operate through centralized systems or decentralized systems.

Centralized approaches have a single solver that sends the actions to the agents thereby reducing the complexity of the system. Decentralized approaches spread the computational efforts across an entire system, allowing agents to calculate their actions and communicate with each other to solve problems in a collaborative way. A decentralized system is more complex and robust than the centralized application because it does not contain a single point of failure. For example, decentralized systems provide benefits in low communication, partial information, dynamic, and unknown environments [3, 38].

As multiagent systems become more popular, they are being exposed to more complex environments. These environments result in a contradiction for the agents where they must increase the algorithmic complexity (computational time) to improve the solution quality. To combat this contradiction, researchers studied the application of metareasoning in multiagent settings. To date, multiagent metareasoning has been applied to instances of multiagent coordination, communication, and resource allocation to name a few [2, 45, 62].

This thesis describes the development and evaluation of a multiagent metareasoning method for distributed multiagent systems that is developed in an offline, supervised, machine learning approach that maintains performance while changing computational or communication attributes. For example, this method can maintain system performance in

a multiagent task allocation problem while reducing the number of recalculations and communication messages generated. The agent calculates the magnitude of each feature at a decision point and inputs these values into the machine learning model during the "monitoring" phase. The model utilizes these features to calculate whether the algorithm should be run or which algorithm should run during the "control" phase. The model is composed of state features whose values will be calculated for each agent at run time. Thus, each agent can independently monitor its environment and control its application of reasoning. At each decision point, the point at which the task allocation or path finding algorithms needs to be run, the agent will run its current state through the supervised learning model in the meta-level, the higher-level reasoning of the agent. The meta-level will either assign an algorithm to execute (path finding problem) or control whether the algorithm should be run (task allocation problem).

1.1 Motivation

Autonomous multiagent systems provide benefits in fields with increased safety risk or coverage over large distances. Examples of such applications include smart grids [61], search [63], tracking in underground mines [21], search and rescue [15, 20], agriculture [17, 41], and warehouses [5, 6, 28, 60]. In these applications, multiagent systems can accomplish the goal faster while keeping individuals safe. One of the best example of autonomous multiagent applications is in supply chain management and warehouse management. Led by companies like Kiva Technologies [60] and Symbotic [46], fulfillment centers and warehouses have reduced costs, increased scalability, and influenced sustainability. As the number of robotic agents in these systems increase, the amount of total computation required by the system grows. To solve this, a switch to

decentralized systems could be made, however these systems put a higher computational load on the agents completing the tasks. For example, in decentralized systems the agents must sense their environment and calculate the necessary motor functions to complete a task while simultaneously determining the task to be completed. This can be compared to a centralized system with one centralized controller determining which agent, focusing primarily on sensing and execution, receives a certain task. Since a decentralized system must take on the overhead of computation and communication of the centralized controller, the agents must attempt to fit more computation in the same hardware. This research focuses on the use of a decentralized multiagent system with metareasoning capability to decrease the computational overhead while maintaining or improving performance.

Two different approaches are used to solve the problem of reduced computation with increased performance. The first approach is to create better fixed algorithms. For multiagent path finding, some of the algorithms include *search-based algorithms*, *A* based algorithms*, *conflict-based search algorithms (CBS), and complete algorithms (M*)*. These algorithms are applied in bounded environments. They all can solve path finding problems, but none are the best fit for all situations. This fact leads to a second solution, the application of metareasoning. Most metareasoning approaches look to a machine learning model or neural network to use at the meta-level. The metareasoning applications have focused primarily on map features [53] or the features of the multiagent system [25]. This research seeks to understand the relationships of the agents at a deeper level to solve the problem with the metareasoning approach.

Like path finding, multiagent task allocation research has implemented single fixed algorithms, such as the Consensus-Based Auction Algorithm (CBAA), Decentralized Hungarian Based Algorithm (DHBA), and genetic algorithms (GA). This type of algorithm is applied in dynamic scenarios without predefined goal locations so the agents can determine which of the targets would be beneficial to the group. These algorithms have been improved over the years, but as algorithm complexity increases so does system overhead. Algorithms needing more data require larger amounts of computation and communication which leads to increases in these attributes. Metareasoning methods have been applied in some scenarios to find the greatest performance using neural networks [23] or communication availability monitoring and switching [12]. Breaking these systems down into parts of a whole and looking at the states of these parts may provide a better basis than the current metareasoning.

1.2 Contributions

This thesis introduces and evaluates a novel metareasoning approach where agents use individualistic values for state features in meta-level monitoring to determine their reasoning in multiagent task allocation and path finding scenarios. This approach attempts to solve the problem of excess computational effort in task allocation and algorithm selection in path finding. While this work is specific to these two problems, the overall concept of separating a multiagent system into its individual parts to understand and benefit from their relationships should be applicable in any multiagent situation. The overall approach takes the machine learning models generated from previous datasets and applies them to understand the interrelationships of the agents.

During run time, the agents collect the state information at each decision point and use them in combination with the machine learning model. The results of simulation experiments provide insights into the advantages and limitations of this metareasoning approach.

1.2.1 Decentralized Multiagent Task Allocation

The algorithms used to solve multiagent task allocation must be run multiple times during a simulation allowing agents to choose new tasks. Constant running of the algorithms permits the agents to communicate and make decisions collaboratively. For these scenarios, the agents will run the algorithms every 0.1 seconds in full communication resulting in excess communication and computation without added benefits. The approach used here is a classification of state features constructed during run time that the agents compute individually so the same values will not be shared. Thus, we can maintain the collaborative communication between agents, allowing them to achieve the overall goal of the system while acting on their own perception of the environment.

1.2.2 Decentralized Multiagent Path Finding

In this situation the algorithms provide different benefits in a range of environments. Much attention has been given to determining a single algorithm for the system. Kaduri *et al.* [12] and Sigurdson [23] both used learning approaches to solve this problem. This thesis considers state features more specific to an individual agent than the aforementioned papers to provide combinations of algorithms for the multiagent system.

All environments are the composition of smaller environments. In the case of videogame maps there may be sections of a specific environment that include bottlenecks while the other side of the map is completely open. In these situations, the application of a single fixed algorithm may be hindering, and different algorithms are needed. This approach attempts to use state features created using locations for tasks and distances to identify an algorithm that may provide the greatest benefit to the agent. Past research has shown that the map influences the path finding algorithm, so each map uses a different machine learning model in the meta-level.

1.2.3 Insights

Research into multiagent systems problems posed in Section 1.2.1 and 1.2.2 has focused on the system as a whole. When considering centralized approaches, the agent can understand the entirety of the system, but in a decentralized approach this may not be the case. This research looks to apply a metareasoning approach that increases the knowledge of how agents in a multiagent system interact when exposed to different environments and problems.

Experiments in this thesis provide two different perspectives for understanding the interrelationships of agents in a multiagent system. The task allocation problem identifies existing features in an agent in a multiagent system that can be used to gain an understanding of the computation benefits. Such a metareasoning policy also brings to light the meta-level effects on different sets of algorithms showing that some may benefit more than others.

The path finding problem helps us understand not all agents in a multiagent system are treated equally. When moving to a decentralized approach, the agents must do calculations on their own and some paths are more difficult to calculate than others. This research shows there exists a combination of algorithms in a multiagent system that may provide added benefit.

Comparing these two problems allows us to identify if this approach can scale between the applications of a multiagent system. The approach is applied to two different modes of multiagent metareasoning showing it can be applied in multiple ways to different systems.

1.3 Overview

Chapter 2 of this thesis outlines the work relevant to this research. Chapter 3 discusses the multiagent task allocation problem studied here and describes the overall metareasoning approach. It presents a detailed, step-by-step guide explaining how metareasoning was applied to the problem. Finally, the results of the simulation study and a summary of the findings are discussed.

Chapter 4 explains the multiagent path finding problem and provides an example of it being used to address a real-life problem in a warehouse. It then presents the overall approach and a step-by-step guide to solve this problem. It ends with the results and summary of the work.

Chapter 5 discusses the findings and insights gained from the research results and suggests future work that can be done in this area.

Chapter 2: Related Work

Multiagent task allocation and multiagent path finding are two common problems in the multiagent domain. Each problem has witnessed numerous approaches such as single fixed algorithms and meta-level control and monitoring.

2.1 Multiagent Task Allocation

One approach to the multiagent task allocation problem is to use a single fixed algorithm with a consensus phase [14, 29, 30, 31, 59]. Another approach by Ismail [24] includes the use of cost matrices to determine the optimal solution. Bapat [3] conducted research on applications for low communication environments. No one algorithm has provided the best solution in every environment, and differences in the algorithm performance were studied in [48].

When there exists a set of fixed algorithms, where no algorithm dominates all environments, a metareasoning application could be used as a solution to the algorithm selection problem. Carrillo [12] used a subset of task allocation algorithms in a metareasoning policy where a conditional rule set was constructed based on the communication quality of the system. In this approach, when the communication quality rose above or dropped below different thresholds, the agents would switch task allocation algorithms. The results showed Carrillo's approach could improve or maintain performance compared to the fixed algorithms. Herrmann [23] used a neural network to determine performance functions for a subset of task allocation algorithms. This allowed the agents to identify the preferred algorithm in a current state. In the scenarios that were

studied, the results of Herrmann's experiment proved there existed a metareasoning approach that outperformed the fixed algorithms.

2.2 Multiagent Path Finding (MAPF)

The multiagent path finding problem or MAPF, has been studied for over two decades. Sturtevant [57] created a benchmark set of acceptable maps for the MAPF problem. Stern [56] built upon these maps by generating scenarios and providing terminology. Optimal multiagent path finding algorithms can be divided into 7 different categories: suboptimal solvers, reduction-based optimal solvers, A*-based optimal solvers, increasing cost tree search (ICTS), conflict-based search (CBS), and sum-of-costs SAT solver. Fixed algorithms used for this problem include: D* [55], different variations of A* [54], M*[58], ICTS [52], CBS [51], and its different variations. Felner [19] demonstrated that there is no one-size-fits-all-algorithm for all applications.

Sigurdson [53] provided a deep learning method to construct a metareasoning approach for MAPF. A convolutional neural network was used for image processing to identify the best algorithm for a given MAPF problem instance. The instances were identified by their map topologies, distributions of the agents, and other characteristics. The distance traveled and goal achievement time were used as metrics and the set of algorithms included: Windowed Hierarchical Cooperative A* (WHCA*), flow annotation replanning, and bounded rationality A*. Kaduri [25] used a similar approach and compared the convolutional neural network image processing to an extreme gradient boosting (XGBoost) supervised learning approach that used MAPF features. The learning approaches were used in combination with algorithms that include EPEA*, A*,

ICTS, CBS, MA-CBS, and a heuristic version of CBS. The results of this paper show that using an algorithm selection model resulted in the solution of more problems and a shorter runtime compared to the fixed algorithms.

Different variations of MAPF have been tested in realistic environments that extend beyond the discrete MAPF benchmarks. Li *et al.* [33] studied a variant of the MAPF problem that incorporates multiple tasks per agent. The study demonstrated the use of a windowed low-level solver approach, allowing an agent's plan to be more pliable to adapt to online settings while avoiding waste of computations for the distant future. This method was scalable up to 1,000 agents. This lifelong variation, usually seen in large warehouse settings, was first studied by Ma *et al.* [34, 35]. Håkansson [22] studied a similar problem to MAPF and used metareasoning to solve a Traveling Salesman Problem (TSP) in a multiagent setting. This approach used multiple meta-agents to monitor static and dynamic characteristics of the environment and determined the fastest route possible to solve the problem which included a network structure like the MAPF implementation. It included static variables like speed and distance and dynamic variables like weather.

2. 3 Multiagent Metareasoning

Metareasoning for an intelligent agent is the application of a three-level reasoning structure first depicted in Cox and Raja [16] and shown in Figure 2.1. The first level, or ground level, represents the computations done by an agent that represent movement or sensing. The second level, or object level, represents the computations done by an agent that contribute to the decision-making process of an agent. The outputs of this level are

the inputs to the ground level action computations. The third level, or meta-level, represents the agent's internal understanding of the benefits of the object level.

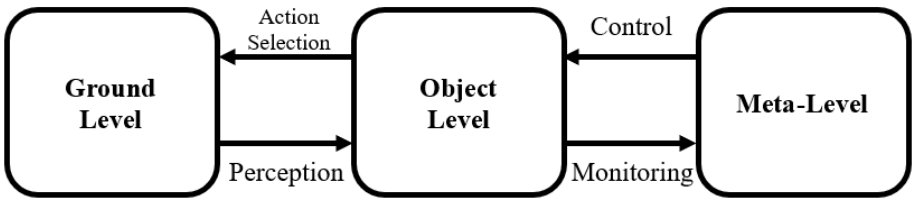


Figure 2.1: Three level structure of metareasoning

Depending on the multiagent system used, the application of metareasoning for these agents' changes. In centralized systems the structure is the same as in Figure 2.1 because there exists only one object level. In our metareasoning approach each of the agents has its own object level that is monitored and controlled by an agent's meta-level. This can be referred to as an independent metareasoning structure [32]. The agents operate with full communication to execute the object level while they communicate path and location information. The structure for the application used in this paper can be seen below in Figure 2.2.

There exists other multiagent metareasoning modes that have not been applied to the problems in Sections 2.1 and 2.2. The set of modes for multiagent metareasoning includes stopping an algorithm, modifying parameter values, modifying reasoning rules, authorizing communication, sharing information, designing coordination, and redefining relationships. Langlois *et al.* [32] surveyed these multiagent metreasoning modes as well as the structures and problems.

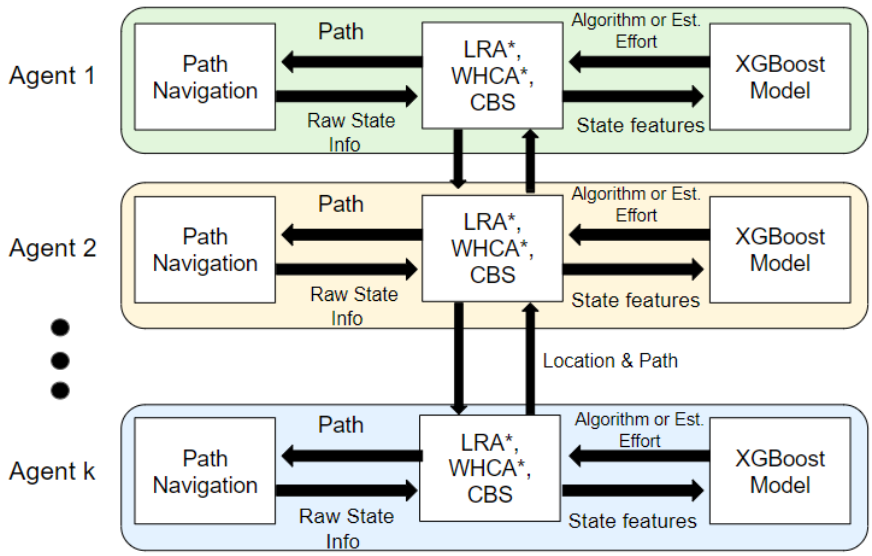


Figure 2.2: Independent metareasoning structure for state feature monitoring and control (based on a figure in Langlois et al. [32]*)*

The algorithm-stopping approach has a meta-level that determines when the object-level computation should stop to allow for other computations and lower-level actions. Zilberstein and Carlin [11, 66] demonstrated this approach using a meta-level Markov Decision Process (MDP) as a probabilistic performance profile to monitor and control an anytime algorithm in a multiagent collaborative decision-making scenario. The MDP determined when an agent should stop running the anytime algorithm to provide the greatest utility to the agent.

The parameter modification approach contains reasoning algorithms composed of parameter values. The parameters can be modified when conditions change to improve performance. Pinyol and Sabater-Mir [43] demonstrated this approach in a marketplace environment where buyer agents were given false or true information. In instances where the agents were able to modify their parameters based on reputation information, the buyer agents had increased performance. Rubenstein *et al.* [47] demonstrated this

approach on search algorithms. This change affected the computational resources as well as the utility of the solution. Noda and Ohta [39] modified learning parameters based on performance to encourage either exploration or exploitation.

The rule modification approach changes the rules by which the multiagent system must interact, usually a priori determined by the system designers. Artikis [2] proposed a framework where the agents were able to vote on their own interest so that the rules of the system would benefit the majority.

The communication authorization approach occurs when agents must use cost or benefit computations to determine whether communication is needed. Xuan et al. [62] used different heuristics for the meta-level control policies to determine the value of the communication. In this work, the policies that incorporated communication cost had the greatest performance. Becker *et al.* [4] proposed an offline policy to calculate the value of communication based on its beliefs of the other agents. If the value was positive, the agents would communicate.

The information-sharing approach contains a meta-level that uses deduction to reason about the object-level of other agents. The meta-level in this case provides controls without using communication instead using its knowledge of the allowable actions for the agents to understand what will occur [40].

The coordination design approach has the agents break down the MAS into smaller groups to reduce the cost of communication in the system. Zhang and Lesser [65] had the agents calculate the performance costs for group combinations. The smallest group for each agent that provided an acceptable performance was chosen. Brueckner [8]

used meta-agents to cluster groups which demonstrated increased effectiveness and reduced metareasoning overhead.

The relationship-redefinition approach focuses on the interrelationships of the agents inside the MAS. Changing these relationships changes the reasoning of the agents. Kota *et al.* [26] studied a meta-level that determined when agents should redefine their relationships. The study determined that a poorly performing agent will prefer to change the relationship despite the reorganization cost. Ahmadi and Allan [1] studied how limiting the amount of relationships to redefine can reduce the cost of reorganization.

2.4 Discussion

Recently the focus on the multiagent task allocation problem has been based on minimizing the total distance traveled [23] or creating robustness in low or varying communication [3, 12]. As research moves toward creating higher quality algorithms, the level of complexity and communication increases as well. To be effective, the task allocation algorithms must be run constantly throughout the agents' mission to maintain system performance due to the completion of tasks or the identification of new tasks. This results in the algorithm being run even when the results will not change the performance. Such overhead requires computational resources that could be diverted to other aspects of the agent. This issue is important because real-world constraints such as SWaP, (size, weight, and power) and cost may limit the computational hardware that can be applied to a multiagent system. These values constrain the agents in a multiagent structure to hardware/specific computational ability. As the size and weight of the agents

decrease, the computational abilities of the agents decrease too, therefore the method of computation the agents use becomes critical. An example of this can be seen in the comparison between reinforcement learning and supervised learning. Reinforcement learning allows for adaptivity throughout the mission but the amount of computation that must be undergone at each step provides a significant overhead [10, 27]. Current research has not focused on the specific task of reducing the amount of computation in task allocation algorithms or the effects on the multiagent system.

The multiagent path finding problem has been studied mainly from the view of fixed algorithms like WHCA*, CBS, and ICTS. The current state of research in this field has not identified a single algorithm that performs best in every scenario which has encouraged the use of metareasoning to solve this problem. The metareasoning approaches used to solve the algorithm selection problem in MAPF include machine learning applied to map image processing and map features to identify the best algorithms for a particular map. These metareasoning approaches consider the map attributes only and they apply one algorithm per map for the multiagent system. In Chapter 4, this thesis describes research using a metareasoning approach that allows each agent to select the path finding algorithm best suited to its own current state. This approach allows for the agents to flexibly determine which algorithm works best for their situation instead of the entirety of the system. The CBS, WHCA*, and LRA* algorithms were included in this approach.

Chapter 3: Decentralized Multiagent Task Allocation

This chapter describes a multiagent task allocation problem and the development and testing of a metareasoning approach that determines when an agent should run a task allocation algorithm. The chapter includes a problem description, overall approach explanation, detailed steps, results, and discussion of implications.

3.1 Specific Research Gaps

As stated in Section 2.4, this chapter focuses on the reduction of computation in task allocation algorithms and its effects in a decentralized multiagent system. This chapter uses state features based on distances to provide individualized states for each agent. This application has never been done in multiagent task allocation. This method allows the agents to use information from past missions to understand its current computational benefits at each point in time.

3.2 Problem Description

This chapter investigates two scenarios for multiagent task allocation: (1) collaborative visit and (2) collaborative search and visit. Both scenarios occur in a two-dimensional square workspace. The collaborative visit scenario has stationary target set $U = \{u_1, \dots, u_n\}$, which the agents know a priori. The agents will be assigned to a set of tasks $T = \{t_1, \dots, t_m\}$. In this scenario $T = U$ throughout the agents' mission. A target is considered visited if an agent moves within the threshold distance δd_T of the target's location. The mission ends when every target has been visited at least once.

The collaborative search and visit scenario has a stationary target set $U = \{u_1, \ldots, u_n\}$, but these are initially unknown. These targets are in a square workspace that is separated into non-overlapping grid cells $G = \{g_1, \ldots, g_r\}$ that are known a priori. Initially U is empty and $T = G$, but, as the agents move around the workspace, they use their sensors (which have a detection radius of R_v) to detect targets. A cell is considered visited when an agent reaches the center of the cell; at this point the agent's sensor radius covers the entire cell, and, if any targets are in this cell, the agent detects these targets and shares their locations with the other agents. The set U of known targets thus grows as the mission progresses, and $T = G \cup U$. After all the cells and targets have been visited at least once, the mission is complete.

3.3 Overall Approach

This metareasoning approach is applied to two coordination algorithms: Consensus Based Auction Algorithm (CBAA) and Decentralized Hungarian algorithm (DHBA). CBAA [9] and DHBA [24] are single task allocation algorithms that assign one task per algorithm cycle. CBAA uses auctions to determine which agent will be visiting which target. DHBA uses a cost matrix to identify which target an agent is assigned. For this research we will be using the min max variations of these algorithms (CBAAMM and DHBAMM). The two algorithms are variations of CBAA and DHBA that use the min-max distance travelled objective. These variations use the current distance travelled by the agents as well as the distance to the target to create their bid evaluations. Both algorithms require communication with the rest of the system to exchange their bid or cost information at each time step resulting in the appropriate

task. In the scenarios considered in this chapter, communication is perfectly reliable (no communication packets are dropped), and every agent can communicate with every other agent. The agents communicate their current locations at every time step, and the agents communicate their task allocations whenever they run the task coordination algorithm.

A multiagent simulation model was used to evaluate the performance of the task coordination algorithms. The simulation software used for this research was described by Nayak *et al.* [48]. We generated multiple instances by randomly selecting target locations.

During the simulated mission, each agent runs the task coordination algorithm (CBAAMM or DHBAMM) every 0.1 seconds. Rerunning the algorithm increases the number of calculations and messages. Due to the static nature of the targets and the logic of the algorithms, after an agent has selected a task, the only reason to switch tasks is due to another agent being assigned to that task and closer to it. In any other instance the agent is rerunning the algorithm and continuously sending messages without new useful information. This unnecessary computation and communication are the motivation for using metareasoning to monitor and control the decision making so that computational and communication resources are used efficiently.

The concept behind the metareasoning policy was to run the algorithm only when it might provide benefit. If the algorithm did not change the current task assignment, then running it at that time was not valuable.

To implement this concept, we developed a metareasoning policy in which the agent's meta-level (which is monitoring and controlling the agent's decision-making

process) determines, based on the current state, whether to run the algorithm. This was implemented as a sequence of decision trees that were learned using XGBoost [13, 36, 37]. XGBoost was chosen because it performs well when data is scarce.

This approach uses four steps: (1) run simulations of the scenarios to determine the results of the algorithms during these missions, (2) run each algorithm offline to construct optimal task assignment, which determines whether running it is needed, (3) use extreme gradient boosting to learn a classification model, and (4) run additional simulations of the scenarios to evaluate the impact of the learned metareasoning policy (and other benchmark policies).

3.4 Explanation of Steps

Figure 1 depicts some details of this approach. The process begins with simulating 240 randomly generated instances for each scenario-algorithm combination (listed in Table 1) and ends with a classification model made of a sequence of decision trees that can be used by each agent. Each scenario-algorithm combination has its own model. (That is, we performed this process four times; once for each combination of scenario and algorithm.) The following subsections describe the steps in this process. The simulation, algorithms, and logging in this chapter was developed by Nayak et al. [48].

3.4.1 Online Simulation

This research generated 240 instances of target locations and used these for each algorithm-scenario combination. The workspace was a square, and the length of each

side was 100 units. Three values were considered for the number of targets (15, 20, and 25), four values for the number of circular "clusters" (1, 2, 3, and 4), and 20 values for the cluster radius (5, 6, …, 24). For each combination of these values, an instance was generated. In an instance with more and larger clusters, the target locations were spread over the entire workspace. In an instance with only one small cluster, the target locations were grouped closer to each other. To create an instance, cluster centers were randomly selected so that the entire cluster was inside the workspace. Then, for each target, the cluster and the "offset" between the target and the cluster center were randomly generated, which determined the target's location.

In the search-and-visit scenario, the grid cells were squares, and the length of each side was 20 units. In both scenarios, the agents moved at a constant speed of 8 units per second.

Next each instance was simulated, in which each agent ran the task coordination algorithm every 0.1 seconds and recorded what happened as the agents completed their mission, which yielded the log files.

3.4.2 Log File

During the simulated mission, at each time step (0.1 seconds), each agent recorded its state information in a log file. At each time step, the log file lists all known target locations (x, y), all agent locations (x, y), the current assigned target of each agent, the number of times the algorithm has run, and the current time step of the simulation. Every agent has its own log file. For each instance, the log files for those

agents were parsed and combined into a new file with comma-separated values (CSV). This new file is called the Historical CSV.

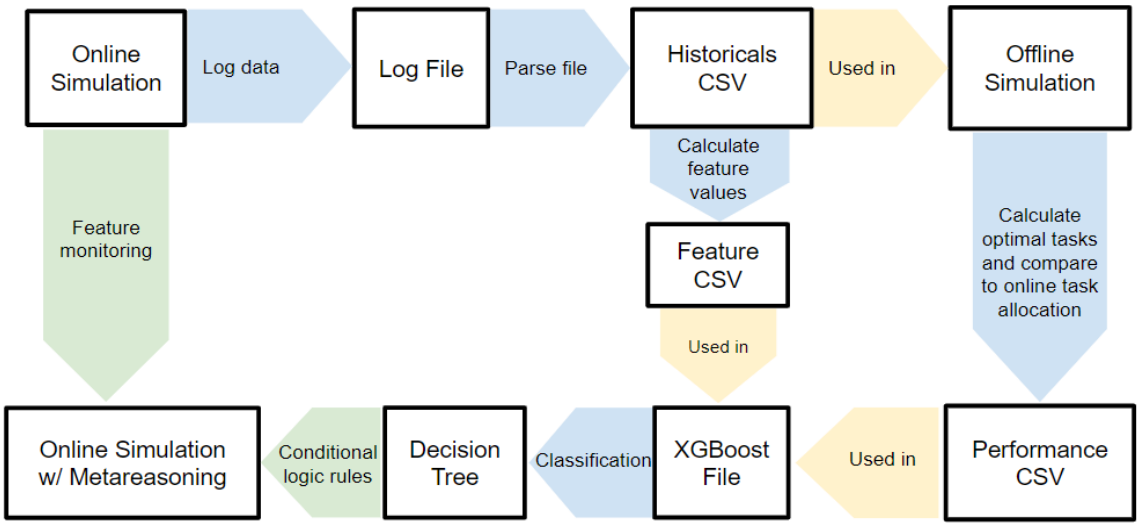


Figure 3.1: Path for determining metareasoning policies. White boxes represent files used to store of process data. Blue arrows represent an action, yellow arrows represent imports, and green arrows identify added functionalities.

Table 3.1: Instance Characteristics

Task Allocation Algorithm	**Scenario**	**Number of agents**	**Number of targets**	**Number of clusters**	**Cluster radius**
CBAAMM	Visit	5	15, 20, 25	1, 2, 3, 4	5, 6, …, 24
CBAAMM	Search and Visit	5	15, 20, 25	1, 2, 3, 4	5, 6, …, 24
DHBAMM	Visit	5	15, 20, 25	1, 2, 3, 4	5, 6, …, 24
DHBAMM	Search and Visit	5	15, 20, 25	1, 2, 3, 4	5, 6, …, 24

3.4.3 Historical CSV

For each instance, the historical CSV file contains the locations of every known target and agent in the simulation at each time step during that mission. (In the search-

and-visit scenario, the locations of unknown targets are only populated in the CSV once they have been found by an agent.) It also records the current task assignments.

Once a simulation has completed the historical CSV is added to a larger cumulative historical file that contains the data from all previous simulations. The array that results from the combination of all the historical CSVs is sparse. This is due to the timestep after an agent completes a task where it is reporting the completed task and running the algorithm. These actions are completed in this timestep and the assignment of a new task is not given. This does not occur when an agent's task is completed by another agent because the agent is not responsible for reporting the completion.

3.4.4 Offline Algorithm

Offline the historical data can be used to identify the closest task for each agent without tasking more than one agent to the same target. The task assignment occurs through the task selection algorithm, but it does not allow two agents to target the same task at once. The online algorithm has specific instances where an agent is outbid after it has already committed to a target and it continues until the target until it has been completed no matter if another agent has been assigned. The offline algorithm results in the agents only assigning tasks based on the distance metric during the current time step. The pseudocode for this algorithm can be found in Algorithm 1.

The offline algorithm assigns a task to each agent; this task may be the same as the running task (the task that was determined by the agent online that was stored in the historical CSV), or it may be a different task. Moments where the task assignment is different include timesteps where the agent's running task is outbid by another agent.

Here the agent should select a new task even though its running task has not been completed. In the online simulation an agent may select a task because all the other agents are currently tasked to different targets. Later, when one of the other agents completes its task it may be the closest to the running task of another agent. This results in two agents tasked to the same target. It is this instance where an agent would benefit in running the algorithm.

When the offline task assignment is the same as the online task for a timestep the agent is tasked with its closest target and no other agent is closer and tasked to that target. In this situation the agent is using computational resources to determine the most optimal task with no change in tasking. These are the instances where the agent is expending resources that result in the same optimal conclusions. The decision-making algorithms (CBAAMM and DHBAMM) also include the expense of resources to communicate during each run so one run expends communication and computational resources.

At each time step, the offline algorithm uses the data in the Historical CSV file to assign targets to the agents. It then compares its assignment to the assignment that occurred during the simulation (recorded in the Historical CSV file) to generate the Performance CSV file.

3.4.5 Performance and Features CSV

For each time step in the instance, the performance CSV file indicates, for each agent, whether the target that the offline algorithm assigned is the same as the target that the task coordination algorithm (CBAAMM or DHBAMM) assigned. This is called a "supervisory signal." Let s_{it} be the supervisory signal for agent i at time step t. If these

targets are different, then $s_{it} = 1$. If these targets are the same, then $s_{it} = 0$. If an agent is in operation and the task coordination algorithm returns a *Nonvalue* and the offline

Algorithm 1 Offline CBAAMM

```
N → Number of agents
G → Number of targets
T → Number of time steps
Data → [Agent1...AgentN, Target1...TargetG, Baseline1...BaselineN] * T
Iterations → 2
Threshold → 0.25
goal[Target1, ..., TargetG] → Locations of the targets
robot[Agent1, ..., AgentN] → Locations of agents at every time step
Baseline → Task assignment for every agent at every time step
Initialize completedTasks and winningBids
completedTasks = [ ]
winningBids → ["inf"] * G
localBids[Agent1 ... AgentN] → [None] * G
tasks[Agent1 ... AgentN] → [None] * T
currentcost[Agent1 ... AgentN] → "inf"
for t = 1, 2, ..., T do
    for iterations = 1, 2 do
        for target = 1, 2, ..., G do
            for agent = 1, 2, ..., N do
                Calculate the Euclidean distance d from robot(agent, t) to goal(target)
                if target in completedTasks or d > winningBids(target) then
                    localBid(agent, target) = inf
                else
                    Set localBid(agent, target) = d
                    if d < agent.currentcost then
                        agent.task = target
                        agent.newtask = True
                        agent.currentcost = d
                    end if
                end if
            end for
        end for
        for target = 1, 2, ..., G do
            for agent = 1, 2, ..., N do
                if agent.newtask == True then
                    if agent.currentcost ≤ winningBid(target) then
                        winningBid(target) = agent.currentcost
                    else
                        agent.currentcost = inf
                        agent.task = None
                    end if
                end if
                if winningBid(target) ≤ and target not in completedTasks then
                    completedTasks.append(target)
                    agent.currentCost = inf
                end if
                tasks(agent, t) = agent.task
            end for
        end for
    end for
end for
```

Algorithm 2 Performance Comparison

```
for agent = 1, 2, ..., N do
    for t = 1, 2, ..., T do
        if Baseline(agent, t) == None then
            Performance(agent, t) = 0
        else
            if tasks(agent, t) == "" then
                Performance(agent, t) = 1
            else if tasks(agent, t) ≠ Baseline(agent, t) then
                Performance(agent, t) = 1
            else
                Performance(agent, t) = 0
            end if
        end if
    end for
end for
```

algorithm also returns a *Nonvalue*, then $s_{it} = 0$. If the agent stops operating (ends before the simulation ends) it will still report a *Nonvalue*. This occurs when the number of remaining tasks is less than the number of agents and an agent loses the bid for all the tasks to another agent. The agent does not record a task for this timestep and moves to the next timestep. The result is a sparse dataset with missing task data. Algorithm 2 contains the pseudo code used to calculate the values for the Performance CSV. We used this data with the feature data (in the Features CSV file) to learn the classification model that was used as the metareasoning policy.

The features used in this study were based solely from features that are independent of time. The features were represented with different measures of distance available in the simulation. The features used in this research include Target Distance, Closest Agent, and Cumulative Distance. These features are calculated for each agent using its position, the positions of the other agents, and the target's position. Let n be the

number of agents. Let $(u_{\alpha x}, u_{\alpha y})$ be the current position of agent u_α. Let (u_{ix}, u_{iy}) be the current position of agent u_i. Let $(t_{\alpha x}, t_{\alpha y})$ be the position of the target that has been assigned to agent u_a. Let TD_α, the target distance feature, be the distance from the agent to its assigned target. Let $d_{\alpha i}$ be the distance from agent u_α to agent u_i. Let CA_α, the closest agent feature, be the distance to the closest agent. Let CD_α, the cumulative distance feature, be the sum of the distances from the agent to all other agents. The features are calculated as follows:

$$TD_\alpha = \sqrt{(u_{\alpha x} - t_{\alpha x})^2 + (u_{\alpha y} - t_{\alpha y})^2}$$

$$d_{\alpha i} = \sqrt{(u_{\alpha x} - u_{ix})^2 + (u_{\alpha y} - u_{iy})^2}$$

$$CA_\alpha = min\{d_{\alpha i} : i \neq \alpha\}$$

$$CD_\alpha = \sum_{i=1}^{n} d_{\alpha i}$$

The features were calculated for each agent at every time step. Since the agents are homogeneous, operate under the same reasoning and have the same physical characteristics, each time step results in *n* data points.

3.4.6 Extreme Gradient Boosting

After the four algorithm-scenario combinations and 240 instances were run for each combination, the simulations produced 648,839 data points. (One data point contained the features for one-time step and the supervisory signals for a single agent.)

We then created a training dataset with 80% of these data points and a test dataset with the other 20%.

For each algorithm-scenario combination, we used XGBoost (https://github.com/dmlc/xgboost) to create a classification model for each state. We used gamma regularization so that the decision tree was not overfit to the training data. XGBoost handles sparse data internally. A description of how XGBoost handles sparse matrices can be found in Mitchell *et al.* [36]. From the Performance CSV file and the Feature CSV file, XGBoost created a sequence of decision trees whose branches split based on a logical operator and a feature value and associated with each leaf node with a $log\ (odds)$ value for the supervisory signal s_{it}. Thus, the sequence of trees classifies a given state through a summation of the $log\ (odds)$ values for each tree in the sequence. Using the logistic function to interpret this value we get a probability that determines whether the agent should run the task coordination algorithm.

The values for the leaves are equal to the similarity score which is calculated using the residuals of the performance value for the data points classified to the leaf (k) to the current probability (R) and the previous probability that the agent will need to run the algorithm (P).

$$Similarity\ Score = \frac{\sum_{i=1}^{k} R_i}{\sum_{i=1}^{k} P_i(1 - P_i)}$$

We used cross validation to tune the XGBoost parameters. To maximize classification accuracy, we used the *multi:softprob* multiclass classification approach, with a learning rate (eta) of 0.1. Table 2 includes the ranges of max depth, minimum

child weight, gamma regularization, and column sample by tree used to determine the best combination through cross validation.

Table 3.2: Parameter descriptions and values used in task allocation cross validation

Parameter	**Description**	**Possible Values**
Number of classes	This is the number of classes that machine learning must classify	2
Max depth	This is the depth to which the tree should be split	3 - 6
Eta	The learning rate	0.1
Minimum child weight	This is the sum of the instance weight. If a leaf node is less than the minimum child weight it is pruned	1, 3, 5, 7, 9, 13
Gamma regularization	Minimum loss reduction required to make a further partition on a leaf node of the tree	0 - 10
Column sample by tree	Subsample of columns when constructing each tree	0.3, 0.5, 0.7, 0.9

3.4.7 Online Algorithm with Metareasoning

All 240 instances were simulated for each algorithm-scenario combination under three new meta-level policies: (1) metareasoning, (2) random, and (3) necessary.

When using the metareasoning policy, an agent uses the appropriate decision tree (the one for that algorithm-scenario combination) to determine whether to run the task coordination algorithm at each time step. It should be noted that, to use this policy, every agent must communicate to every other agent its current location at every time step. This is part of the overhead of the metareasoning policy.

When using the random policy, an agent will use the task coordination algorithm at a given time step if a random value between 0 and 1 is less than or equal to 0.5. That is, the agent uses the algorithm 50% of the time.

When using the necessary policy, an agent uses the task coordination algorithm only when it has completed its task (visited a target or cell) and needs a new target or cell to visit.

The original simulation results, when the agents used no meta-level policy, was denoted as the "control" policy. The random and necessary policies were used as benchmarks for assessing the performance of the metareasoning policy.

3.5 Results

This section contains the results of the machine learning cross validation tests, the feature importance results, and the results for all the algorithm-scenario combinations.

3.5.1 Cross Validation and Importance Results

Before running the cross validation, I determined which of the two classification methods I should use: *multi:softprob* or *multi:softmax*. Softmax outputs a class output (l or 0 in this case); softprob outputs a vector of probabilities for each of the classes (Ex. 0.66 and 0.34). Using the same number of classes ($number\ of\ classes = 2$), the max depth ($max\ depth = 3$), and learning rate ($\varepsilon = 0.1$) for both classifications I calculated the accuracy for classifying the CBAAMM Search and Visit data. The accuracy when using *multi:softprob* was 91.9%; the accuracy when using *multi:softmax* was 83.7%.

The cross validation, using *multi:softprob* classification, was tested on the CBAAMM Search and Visit combination. This resulted in the values from Table 3.3 with a gamma value of 0, with an accuracy of 87.8%.

Table 3.3: Extreme gradient boosting tuned parameters from CBAAMM, Search and Visit

Parameter	**Value**
Number of classes	2
Max depth	6
Eta	0.1
Minimum child weight	1
Gamma (regularization)	80
Column sample by tree	0.7

Although it was the best solution, this parameter set was too complex to implement due to its zero-gamma value because this resulted in a total of 64 leaves. To simplify the decision trees a greater gamma value was needed. Multiple gamma values were tested while keeping the other parameters constant. The goal of this test was to identify the trade-off of percent accuracy compared to the simplification of the decision tree. At a gamma value of 0 the decision tree had 64 leaves and 87.8% accuracy. This would mean that there would be a minimum of 32 conditional statements for each tree in the classification model. The larger the tree the larger the computational overhead so the goal would be to have less than 10 leaves in each tree. At a gamma value of 80 the decision trees had about 10 leaves with 87.3%accuracy. Due to the small change in accuracy and low complexity the gamma parameter was chosen to be 80. The parameters determined in this cross-validation assessment were used in the extreme gradient boosting for all the other algorithm-scenario combinations. They can be seen in Table 3. The accuracy results for the CBAAMM Visit, DHBAMM Search and Visit, and the DHBAMM Visit scenario were 78.7%, 86.3%, and 78.4% respectively using these parameters.

The supervised learning also results in the identification of the feature importance in each of the combinations. The features are given an F score when the tree is constructed; a greater F score indicates that the feature has more influence on the tree. The feature importance values can be seen in Table 4. These suggest that the TD_α feature is more influential than the other features.

Table 3.4: F Score (Features Importance) values for all algorithm scenario combinations

Features	CBAAMM Visit	CBAAMM S&V	DHBAMM Visit	DHBAMM S&V
TD_α	160	162	100	117
CA_α	105	127	62	84
CD_α	77	101	40	42

3.5.2 Simulation Results

These simulations represented the control for the experiments. Once the meta-level controls were identified from each combination another 240 simulations were run with inclusion of the meta-level. Simulations with a 50% random policy and task completion policy were run for comparison. The random policy is like running the decision-making algorithm every 0.2 seconds. The task completion policy runs the algorithm only when the task an agent has been assigned to has been completed.

For each algorithm-scenario combination, meta-level policy (control, metareasoning, random, and necessary), and instance, we assessed performance using three metrics: distance travelled, total messages exchanged, and total runs. The distance travelled metric is the total distance that all the agents traveled during the mission. The total messages exchanged metric is the total number of messages sent by all agents during the mission. The total runs metric is the total number of times that the agents ran the task

coordination algorithm during the mission. Figures 3.2, 3.3, and 3.4 show the distributions of these metrics for each policy in each algorithm-scenario combination.

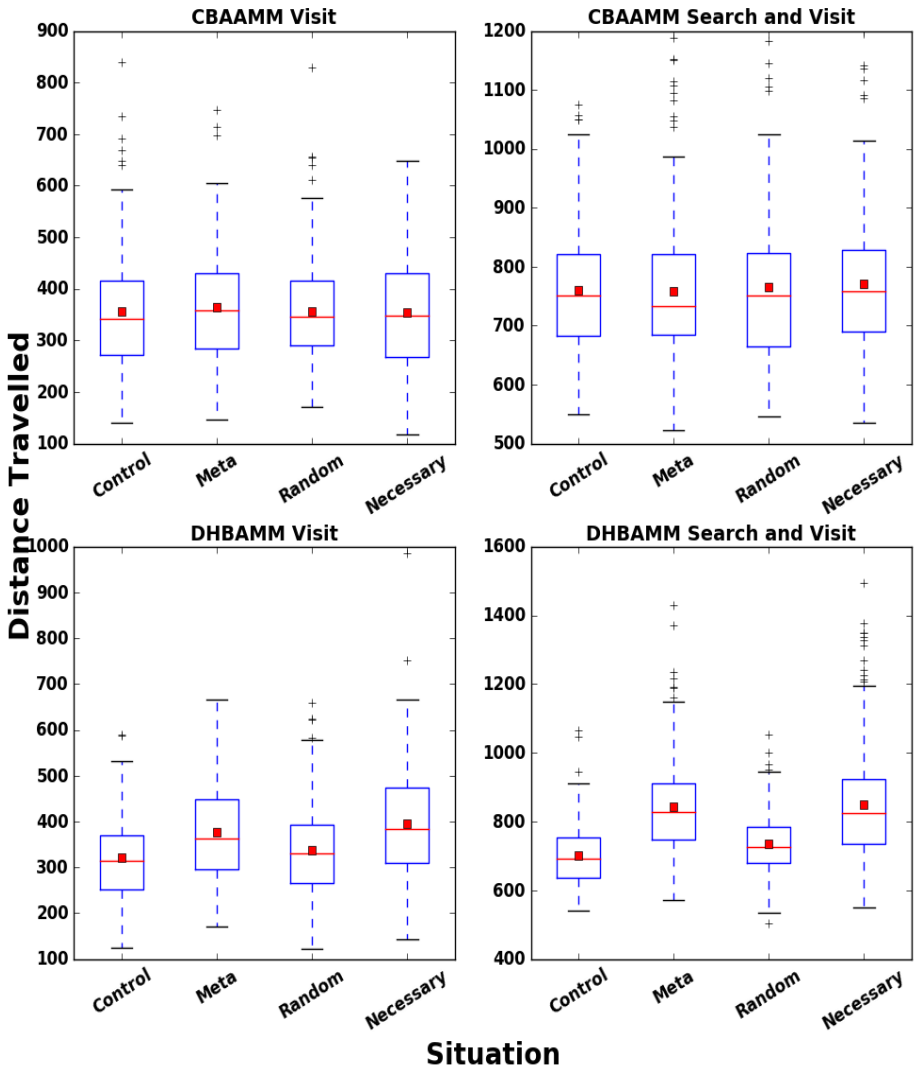


Figure 3.2: Distance travelled for each algorithm scenario combination. Red dots represent averages, the red lines represent the medians, and the box represents the middle two quartiles.

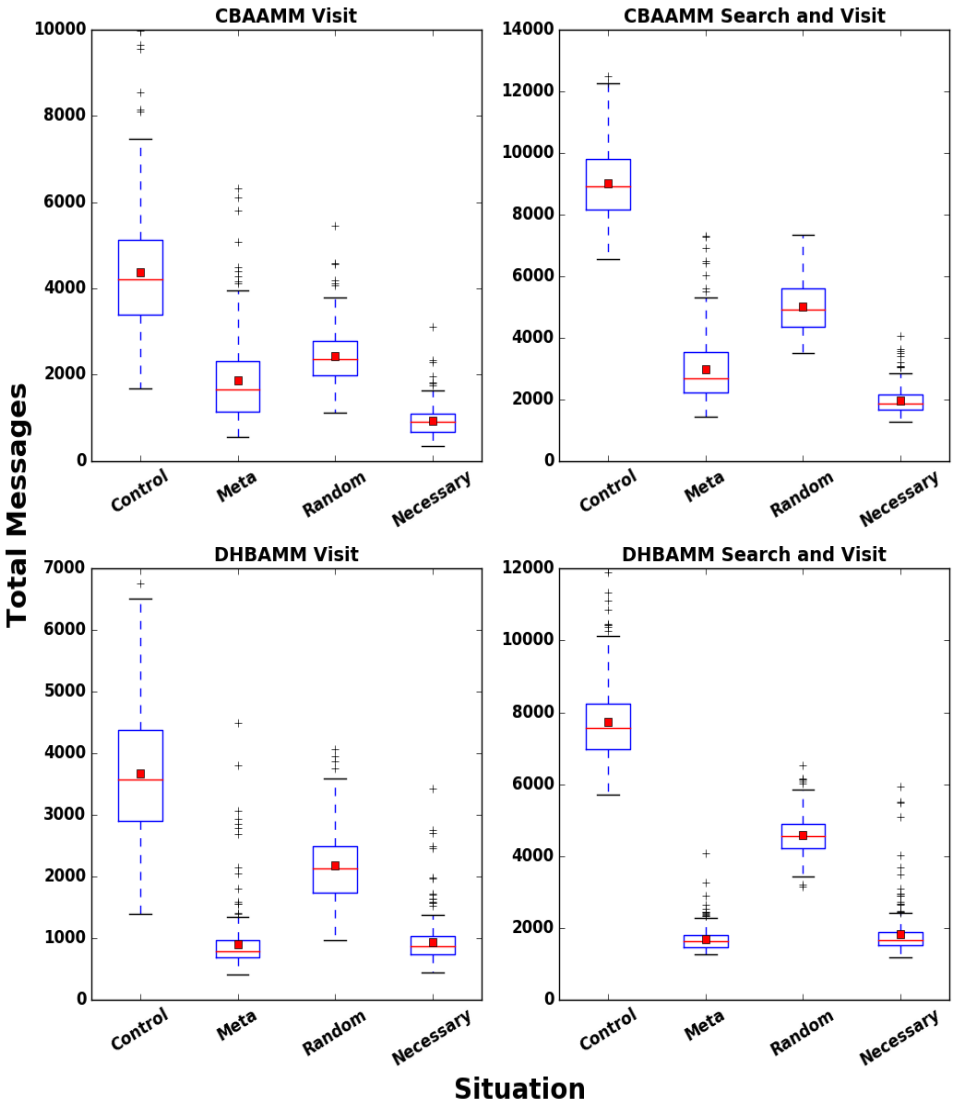


Figure 3.3: Messages exchanged for each algorithm scenario combination

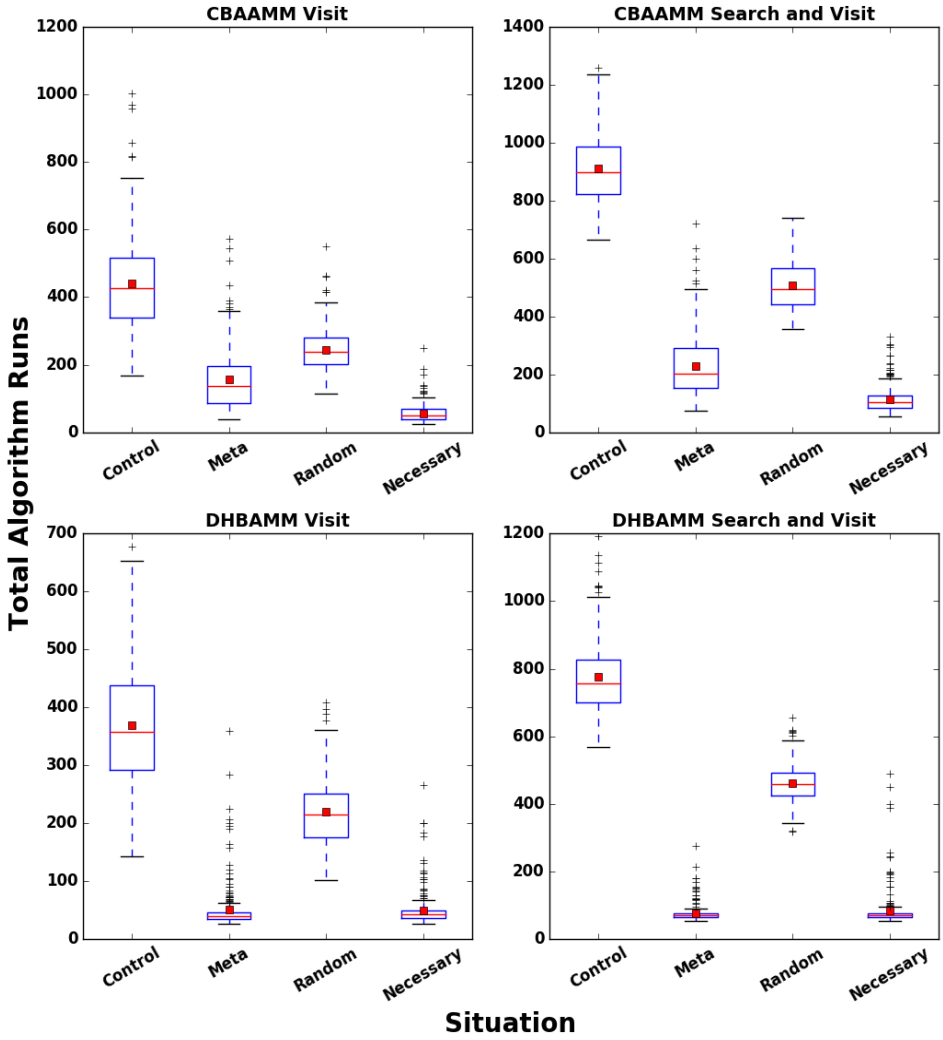


Figure 3.4: Total runs for each algorithm scenario combination

3.6 Discussion

The baseline policy, in which the agents ran the task coordination algorithm every time step (labelled "Control" in Figures 3.2, 3.3, and 3.4), yielded lower mean and median distance values than the metareasoning policy in three of the four algorithm-scenario combinations. These values were smaller in the CBAAMM scenarios than the DHBAMM scenarios. The baseline policy yielded much larger values for the messages

exchanged and total runs metrics, however. Thus, it appears that the metareasoning policy can reduce computational effort and communication costs without degrading system performance (total distance); indeed, in one case (using CBAAMM in the search-and-visit scenario), the system performance was not worse. Although the necessary policy improved the messages exchanged and total runs metrics even more than the metareasoning policy, it also degraded system performance more.

The baseline policy (which runs the algorithm every time step) serves as an upper bound on the messages exchanged and total runs metrics. The random policy runs the algorithm half as often as the control. In all the combinations the metareasoning policy yields values less than the random policy and in the case of DHBAMM it is similar to the messages exchanged and total runs of the necessary policy.

The control policy exchanged the most messages and performed the algorithm significantly more than any of the other methods. As shown in Figures 3.3 and 3.4, running the algorithm constantly acts as an upper bound for the number of messages transmitted and algorithmic runs per simulation. This is a result of the policy running every 0.1 seconds; although the other policies determine whether the algorithm will run, the control automatically runs the algorithm. Running the algorithm results in a sending of bids messages to the other agents. Running the algorithm in the necessary situations results in the least number of algorithmic runs to complete and therefore the least messages. The performance of the random policy, which ran the task allocation algorithm 50% of the time, was near the average of the performance of the control and necessary policies, as shown in Figures 3.3 and 3.4. The metareasoning policy does not follow the same trend of Figure 3.2. The metareasoning provides a reduction in the

overall messages passed and a reduction in algorithmic runs. The metareasoning is consistently lower than the random policy in all algorithm scenario combinations and in the CBAAMM instances it completes the simulation with less distance.

Not all the features in the supervised learning were used to construct the classification model. The cumulative feature has no logical expressions in any of the scenarios. Given the two features used for these decision trees, XGBoost was able to obtain a decision tree that provided a structure that decreased the messages and the computational overhead of the decision-making process while minimally affecting the performance of the system. The decision trees used in this study were all less than 90% accurate. Increasing the classification model's accuracy would improve an agent's ability to detect instances when it would otherwise be over computing. Since the occurrences of multiple agents on a single task is less likely than the agents going to different tasks this would reduce the algorithmic runs further as well as the communication of the system. However, this increased accuracy will also increase the model complexity and a larger computational overhead for the model.

Overall, these results indicate that a metareasoning policy that is trained using supervised learning can decrease overhead while maintaining performance. The approach was also able to scale between the different algorithms and environment complexities while producing similar results. The benefits of the metareasoning can differ based on these algorithms and scenarios.

Chapter 4: Decentralized Multiagent Path Finding

This chapter describes the multiagent path finding problem and the development and testing of a metareasoning approach that determines out of a set of path finding algorithms which one should be executed for each agent. The approach is implemented on a decentralized system using optimal and suboptimal centralized path planning algorithms that have been converted into their decentralized logical equivalents. The chapter includes a problem description, overall approach explanation, detailed steps, results, and discussion of implications.

4.1 Specific Research Question

The lack of a single dominating fixed algorithm for the MAPF has motivated research on the relevant algorithm selection problem. Current metareasoning research used to solve this problem has utilized map features to determine which of the multiagent path finding algorithms are best suited for a specified map [25, 53]. Although this approach has provided positive results, it has only focused on the problem from a centralized system viewpoint. This chapter attempts to identify whether using the individualized state features of an agent in a decentralized multiagent system to solve the algorithm selection problem for MAPF will provide better results than the individual fixed optimal and suboptimal algorithms. The algorithms used in this section include LRA*, WHCA*, and CBS.

4.2 Problem Description

The *classical multi-agent pathfinding* (MAPF) problem with *k* agents is defined by a tuple $\{G, s, z\}$ where $G = (V, E)$ is an undirected graph, $s : [1, \dots, k] \rightarrow V$ maps an agent to its source vertex, and $z : [1, \dots, k] \rightarrow V$ maps an agent to its target vertex [56]. The timeline is a discrete sequence that begins at t_0, and agent a_i is at s_i at time t_0. During each time step an agent can complete a single action. The agent can complete a wait or move action at each time step. A wait action is one where the result of the action is the current vertex. A move action is one where the agent moves to a vertex that is adjacent to the current position. During a move action the agent must move along an edge denoted by $(v, v') \in E$. A solution to the classical MAPF is a set $p = \{p_1, \dots, p_k\}$ of paths, one for each agent, where a path is a sequence of wait and move actions where the agents can move from source to target without conflicts. In this research we consider two different types of conflicts: swapping conflicts and vertex conflicts. We will refer to these as edge collisions and node collisions respectively in this paper. An edge collision occurs when two agents switch vertices in a time step resulting in them traveling on the same edge. A node collision occurs when two agents arrive at the same node at the same time step. These collisions are used with the algorithms to help agents avoid following faulty paths.

Sturtevant [57] proposed a set of benchmark maps M that includes warehouses, mazes, video game maps, and random environments. This study used three maps, each of a different type (warehouse, videogame, and random) that includes fewer than 12,000 vertices. When acting the agents will only be able to move to four adjacent vertices other

than its current vertex, otherwise known as a 2^n neighborhood grid where n is equal to 2. This means agents may only move in the x and y direction, and they may not move on a diagonal. We studied a decentralized multiagent system using a decoupled path finding approach, while trying to minimize the sum-of-costs (the sum of all individual path costs) and minimize the sum of the computational time for all agents. The source and goal nodes were randomly generated, and the maps chosen will not have any isolated portions that would otherwise lead to the inability for an agent to determine a path.

4.3 Overall Approach

The algorithms used in this study are LRA* [64], WHCA* [54], and CBS [51]. These three algorithms were chosen due to their use of the A* algorithm, their ability to be converted into a decentralized form in full communication, and their temporal relationship to collision checking. A description of these algorithms can be found in Section 4.4. Other algorithms like the complete M* were not included because they do not enhance the testing of different collision checking times.

The metareasoning approach used in this research is Selecting a Reasoning Algorithm. That is, each agent's meta-level chooses the algorithm that it will use to find its path. No matter the map or the agent, the decision is based on the understanding of the state features. It will then choose which of the algorithms will provide the most benefit.

As explained in Section 4.5, we first ran the fixed algorithms on each of the maps to collect state and performance data. At the start of each run distance-based state features are used to gain an understanding of the current situation for each agent. These

features do not include any information about the environment, instead they only consider the magnitude of the distances to the other agents as well as the magnitude of the distance to the targets. They also use the average of the distances between the agents and the source to target distances. Once all the runs were complete the information was processed using XGBoost. Both classification and regression were used.

Once the models were constructed a subsection of the runs were conducted a second time. During a run, each agent used the classification or regression models to determine which of the algorithms was best suited for the state at t_0 depending on the metric used to create the model. In both cases the model is trying to minimize the metric. The metrics used were the sum-of-costs and the sum of computational time.

4.4 MAPF Algorithms Used

Local Repair A* (LRA*) is a search-based solver [54] which comes from the brute force planner developed in [63]. It allows the agents to calculate their optimal paths excluding the other agents. The agents then begin to follow their paths until a collision is imminent. Before an agent moves into a position that would result in a collision (edge collision or node collision) the agent recalculates the rest of its path. During the recalculation, the agent considers the collision resulting node as an obstacle for the next time step, the time step at which the collision would have occurred. This collision check happens three times so that new paths do not result in new collisions. To implement this algorithm, the agents must have the ability to perceive two nodes in each direction from the current location. These responsive traits tend to result in a difficulty in bottleneck environments where there are many agents. Silver [54] noted that this type of a situation

causes the algorithm to rerun at every time step causing the agent to cycle between two adjacent vertices.

Windowed Hierarchical Cooperative A*(WHCA*) is also a search-based solver [54]. WHCA* uses A* initially to compute the optimal path of an agent however after an agent has computed the optimal path the agent must check a small portion of it before it is accepted. The agents must check their paths with a global reservation table for a specified window, which is a fixed number of future time steps. After completing the checks if an agent does not find any conflict the agent may proceed and update the reservation table. If it does find a conflict, the agent must run the algorithm again using the current reservation table as a set of constraints in determining the new path. The agents do these windowed checks synchronously when they reach the midpoint of the reservation table. For example, if the window is set to be sixteen when the agents reach the eighth vertex in the table it will be shifted forward. The current vertex becomes the first vertex in the reservation table and the following vertices are populated. This algorithm is known in research to be centralized but the application in this research allows the agents to share the current reservation table with the next agent in the hierarchy. By following this process, the structure is maintained, and each agent can calculate the path with A* and the reservation table constraints.

The Conflict Based Search (CBS) algorithm is a two-level solver that uses A* in the low-level and a constraint tree in high-level [51]. Initially A* is used to determine the optimal paths for each of the agents. A single agent is then assigned by design to check if any conflicts exist in the paths. If no conflicts exist, the initial paths are collision free and accepted. If a conflict does exist two nodes are created with the paths involved, agents

involved, and time of occurrence. The agent that discovered the conflict must run the A* algorithm in each node using the conflict vertex and time as a constraint. If the agent determines nonconflicting paths in one node while the other node contains a conflict the non-conflicting paths are returned. If both nodes have non-conflicting paths the sum-of-costs of the paths is taken and the one with the lesser value is returned. The node with the lowest sum-of-costs becomes the paths and constraints used in the next iteration. CBS is known to be a centralized algorithm, but Sharon [51] described that if a system had full communication and was fully cooperative it would be logically equivalent to a decentralized system. In this research CBS has been constructed in this decentralized manner. When the branch of a constraint tree is split into two nodes one agent out of the set of agents replans its path. If this new node is the lowest cost out of all nodes than the agent whose path was replanned is responsible for the next iteration of conflict checks and replanning. This is made possible because the new plans are communicated to all agents and the algorithm runs through any agent that has needed to replan its path.

The metareasoning approaches allow for multiple algorithms to be used in the system which provides an issue of conflict resolution. To solve this problem each agent uses the LRA* algorithm to resolve conflicts that wouldn't occur in the single fixed algorithm implementations. The WCHA* and CBS algorithms don't include agents in their solutions that do not run the same algorithm. For example, the CBS algorithm only runs the conflict checking and replanning for the set of agents that are running the CBS algorithm.

4.5 Explanation of Steps

This approach relies on data from initial experiments that can be used for training of a supervised machine learning model. The initial experiments include testing on three MAPF benchmark maps [56], three fixed path finding algorithms, and two metareasoning approaches trained using XGBoost. XGBoost was chosen because of successful metareasoning application in Kaduri et al. [25].

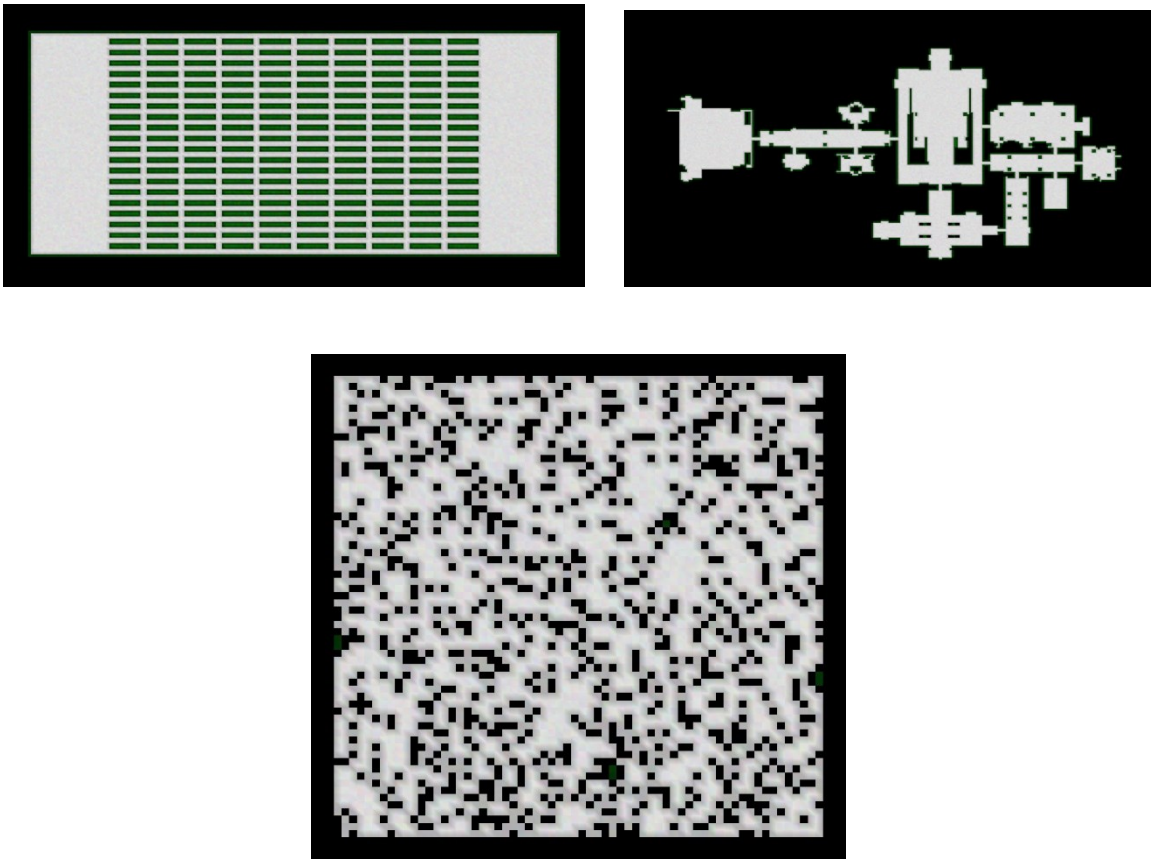
4.5.1 Scenario Generation

Each fixed algorithm will be run with sizes of multiagent systems ranging from 10 to 70. As the size of the multiagent system increases the number of runs decreases. This is done because the agents log their state features at the beginning of the simulation thus the number of data points in a single run is dependent on the size of the multiagent system. There will be 500 runs per algorithm for each of the maps, this is a total of 1,500 initial runs. This will result in 16,900 data points for each of the maps per algorithm.

The maps chosen for this experiment include three types: warehouse, video game, and random. Due to the number of runs needed per map we were unable to use all the maps from the benchmark set. Each of the maps chosen were selected based on the number of nodes in the map. Due to the application of this approach in MATLAB, medium size maps (about 12,000 nodes or less) were chosen. The specific maps chosen were the warehouse 10-20-10-2-2 map, the lt_gallowstemplar_n map, and the random 64-64-20 map. These maps can be found at https://movingai.com/benchmarks/mapf/index.html [57], and they will be referred to as warehouse, gallows, and random respectively and

they can be seen in Figure 4.1. Table 4.1 contains the design of experiments for the initial tests.

(a) (b)



(c)

Figure 4.1: Maps used in this research: (a) warehouse, (b) gallows, (c) random.

These algorithms were also tested using the 128 by 128 maze with a corridor width of two, but it was excluded from this study because the algorithms were unable to solve this problem in the larger system sizes without collisions.

Table 4.1: Initial data for all algorithms and maps

Algorithm	**Map Type**	**Number of Agents**	**Number of Runs**
A*	{warehouse, gallows, random}	{10, 20, 30, 40, 50, 60, 70}	{110, 90, 80, 70, 60, 50, 40}
WHCA*	{warehouse, gallows, random}	{10, 20, 30, 40, 50, 60, 70}	{110, 90, 80, 70, 60, 50, 40}
CBS	{warehouse, gallows, random}	{10, 20, 30, 40, 50, 60, 70}	{110, 90, 80, 70, 60, 50, 40}

In Table 4.1, the number of agents is related to the number of runs respectively. This means that the system size of 10 agents had 110 runs, 20 agents had 90 runs, and 30 agents had 80 runs. This tactic was used so that the smaller system sizes would have enough data points for the machine learning.

4.5.2 Feature and Performance Logging

The feature attributes were set so that no agent would share the same value in the same run. This means that none of the features are related to the map or the values of the system as a whole and instead their magnitudes change based on which agent is completing the computation. To accomplish this, we avoided using temporal attributes, and we used only attributes that could be calculated using the data pertaining to agent locations, target locations, and the number of agents. These factors were based on the understanding that if the agents must travel during the simulation the farther the agent is from another agent the lower the likelihood that the agents will receive a conflict. From the target side, the farther the target location is from the source the higher the likelihood the agent will experience a conflict. Given these assumptions there may be factors based

on distance that may classify states where agents would benefit from one algorithm over another.

The features used in this research include Number of Agents (NA), Closest Agent (CA), Farthest Agent (FA), Average Agent Distance (AAD), Target Distance (TD), Ratio of Closest Agent to Average Agent Distance (RCA), and Ratio of Task Distance to Average Task Distance (RTD). Let k be the number of agents. Let $(u_{\alpha x}, u_{\alpha y})$ be the current position of agent u_α. Let (u_{ix}, u_{iy}) be the current position of agent u_i. Let $d_{\alpha i}$ be the manhattan distance from agent u_α to agent u_i. Let CA_α, the closest agent feature, be the distance to the closest agent. Let FA_α, the farthest agent feature, be the distance to the farthest agent. Let AAD_α, the average agent distance feature, be the average distance to the agents. Let $(t_{\alpha x}, t_{\alpha y})$ be the position of the target that has been assigned to agent u_a. Let TD_α, the target distance feature, be the Manhattan distance from the agent to its assigned target. Let RCA_α, the ratio of the task distance to the closest agent feature, be the ratio of TD_α divided by CA_α. Let RTD_α, the ratio of task distance to average task distance feature, be the ratio of TD_α divided by the average of all task distances. The features are calculated as follows:

$$d_{\alpha i} = |u_{\alpha x} - u_{ix}| + |u_{\alpha y} - u_{iy}|$$

$$CA_\alpha = \min\{d_{\alpha i} : i \neq \alpha\}$$

$$FA_\alpha = \max\{d_{\alpha i} : i \neq \alpha\}$$

$$AAD_\alpha = \frac{\sum_{i=1}^{k} d_{\alpha i}}{k-1} : i \neq \alpha$$

$$TD_\alpha = |u_{\alpha x} - t_{\alpha x}| + |u_{\alpha y} - t_{\alpha y}|$$

$$RCA_\alpha = \frac{TD_\alpha}{CA_\alpha}$$

$$RTD_\alpha = \frac{TD_\alpha}{\left(\frac{\sum_{i=1}^{k} TD_i}{k-1}\right)} : i \neq \alpha$$

The performance metrics measured in these experiments includes the sum-of-costs and the sum of computational time. Let $C = \{c_1, \dots, c_k\}$ be the set of costs for each agent in a single run. The cost c_i equals the number of move and wait actions of agent i in a run. Let $J = \{j_1, \dots, j_k\}$ be the set of computational times for each agent in a single scenario. The computation time j_i equals the total time that an agent takes running the algorithm for a single scenario.

4.5.3 Extreme Gradient Boosting

We implemented XGBoost on the initial test data. In this experiment we calculated two different machine learning models for each of the maps. The first approach used C with classification to determine which algorithms should be run in each state. We used the *multi:softprob* classification approach, which is a multiclass approach that creates a vector of probabilities for each of the classes.

The second approach used J with regression. This resulted a model that can be used to estimate the predicted computational effort for each algorithm at the beginning of

a run. In this approach, each agent chooses the algorithm that will require the least computational effort.

Before implementing the machine learning approaches, we conducted a cross validation test to determine the parameters for the machine learning application. We conducted two cross-validation checks for each map, one for classification and one for regression. Table 4.2 and Table 4.3 show the inputs to the cross-validation experiments. The results of the cross-validation experiments can be found in Section 4.6.

Table 4.2: Cross Validation for Extreme Gradient Boost Classification

Parameter	**Description**	**Possible Values**
Number of classes	This is the number of classes that machine learning must classify	3
Max depth	This is the depth to which the tree should be split	5 - 8
Eta	The learning rate	0.1, 0.2, 0.3
Minimum child weight	This is the sum of the instance weight. If a leaf node is less than the minimum child weight it is pruned	1, 3, 5
Gamma regularization	Minimum loss reduction required to make a further partition on a leaf node of the tree	0 - 10
Column sample by tree	Subsample of columns when constructing each tree	0.5, 0.75, 1

The tuned parameters were input into the machine learning approach. This resulted in a model saved as a binary file. When a run begins the agent calculates the state features in the initial step. Then, the model was loaded into the MATLAB simulation. Using these state features the model chooses one of the three algorithms and gives the result to the agent.

Table 4.3: Cross Validation for Extreme Gradient Boost Regression

Parameter	**Description**	**Possible Values**
Eta	The learning rate	0.1, 0.2, 0.3
Max depth	This is the depth to which the tree should be split	5 – 8
Minimum child weight	This is the sum of the instance weight. If a leaf node is less than the minimum child weight it is pruned	1, 3, 5
Gamma regularization	Minimum loss reduction required to make a further partition on a leaf node of the tree	0 - 10
Subsample	Fraction of observations to be randomly sampled (total training data selection)	0.5, 0.75, 1

4.5.4 Meta-Level Reasoner

At run time either the classification model or the regression model will be used as the meta-level reasoner. Each agent will calculate the set of state features at the start of each scenario. These features will then be passed into the model during the meta-level monitoring process. In this case the model will run a python function in MATLAB to determine the best algorithm out of the three. In classification, the extreme gradient boosting model goes through a sequence of boosted trees resulting in a set of leaves containing similarity scores. If the magnitude of the similarity scores is low, the group of data separated into the leaf have different classifications and therefore the separation is a poor one. If the magnitude of the similarity scores is high, the group of data are similar and therefore the separation, or classification, is good. The equation for the calculation of the similarity score can be seen below:

$$Similarity\ Score = \frac{\sum_{i=1}^{m} R_i}{\sum_{i=1}^{m} P_i(1 - P_i)}$$

Where R is a set of residuals and P is the previous probability. The construction of trees with different branches and leaves occurs many times each tree constructing new branches and leaves based on the probability, P, calculated by the previous tree. The result is a sequence of decision trees that use the feature data to move through each tree resulting in a leaf. The resulting set of leaf values is then summed to calculate a *log(odds)* value. Each of the algorithms receives its own *log(odds)* value. A probability is then calculated using the logistic equation below:

$$Probability = \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

The model then returns whichever of the algorithms has the greatest probability, in this case the algorithm that will most likely result in the lowest cost. This determination will become the meta-level control used by the agent to select which of the algorithms it will use in the object-level. The result of this testing on each agent will be a non-homogenous set of algorithms, $A = \{a_1, \dots, a_n\}$, where each $a_i \in \{LRA^*, WHCA^*, CBS\}$.

Using extreme gradient boosting with regression will produce three times the machine learning models compared to classification, one for each of the fixed algorithms. These models will be used to estimate the computational time an algorithm will take given the state. An average of all the computational times is taken to determine the original leaf average (φ). The data is then split into branches of a decision tree like the classification approach. The residuals for each data point in a branch are taken in respect to φ. If the group has a high magnitude of similarity score the values in this branch are similar. Once a tree has been completed the predicted value for the data point is adding

the φ to the average residual (R_i) of the leaf times the learning rate (ε). This occurs for a sequence of *n* trees. The equation for this calculation can be seen below:

$$Prediction = \varphi + \sum_{i=1}^{n} \varepsilon R_i$$

When new state features are provided to the model the trees are implemented in their sequence. The φ is always the same but as the conditional values of the tree evaluate the state features different residuals are determined based on the resulting leaves. At the completion of the model there exists a set of residuals that when summed together and multiplied by the learning rate result in some estimation of computational time. Since there are three models, one for each of the algorithms, the meta-level chooses the algorithm with the minimal computational time, $\min\{J\} = \min\{j_{A*}, j_{WHCA*}, j_{CBS}\}$. The algorithm with the minimal computational time will be used by the agent.

4.6 Results

This section contains the results of the machine learning cross validation tests, the feature importance results, the distance and computation results for the algorithms, and the percent of each algorithm use in the metareasoning for the differing maps.

4.6.1 Cross Validation and Importance Results

The cross validation for the classification model was conducted using the *multi:softprob* objective. This objective returns a vector of probabilities in relation to the algorithms, the algorithm with the highest probability will be run in the object-level by an

agent. The *number of rounds* was set to 25 to reduce the amount of decision trees in the sequence. The *evaluation metric* used was the multiclass negative log-likelihood. This metric was used to validate the data. The *number of classes* was equal to three since there are three algorithms to classify. The results for this cross validation can be seen in Table 4.4.

When classifying the algorithms, the algorithm with the smallest distance is chosen. When a tie occurs, a tie breaker is used to choose the algorithm. If LRA* is tied for the lowest distance LRA* is returned. If WHCA* and CBS tie than WHCA* is returned.

Table 4.4: Tuned Parameters from the classification cross validation

Parameter	**Warehouse**	**Random**	**Gallows**
Eta	0.1	0.3	0.1
Max depth	7	6	5
Minimum child weight	5	5	5
Gamma	4	3	2
Column sample by tree	0.5	0.5	0.5
Accuracy	63%	59%	63%

The regression cross validation used the *reg:squarederror* objective. This is the default objective. The evaluation metric used to validate the data was the *root mean squared error.* The *number of estimators* (number of trees in sequence) used for this model was set to 100 to provide a mid-sized model with moderate accuracy and computation. The results of the regression cross validation can be seen in Table 4.5.

The tuned parameters from the cross validation were used to construct three classification models and nine regression models. These models consist of trees separated by conditional statements in relation to feature values. Not all feature values

are equivalent. Features that have a higher F Score are more important to a model and these F Scores are reported in an importance table. The results of the feature importance for each model can be found in Table 4.6 and Table 4.7.

Table 4.5: Tuned parameters from the regression cross validation

	Warehouse			Random			Gallows		
Parameter	**LRA***	**WHCA***	**CBS**	**LRA***	**WHCA***	**CBS**	**LRA***	**WHCA***	**CBS**
Eta	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Max depth	5	7	5	5	5	5	5	6	5
Minimum child weight	1	5	5	3	5	5	1	1	1
Gamma	9	1	5	0	0	9	3	6	7
Subsample	0.9	0.5	0.9	0.5	0.3	0.9	0.9	0.9	0.9
Score	68%	80%	24%	78%	80%	24%	72%	88%	26%

Table 4.6: F Scores for classification importance

Features	Warehouse	Random	Gallows
RTD_α	330	184	355
AAD_α	91	111	170
RCA_α	81	114	133
NA	219	108	244
TD_α	130	77	137
CA_α	15	28	131
FA_α	46	55	66

Table 4.7: F Scores for regression importance

	Warehouse			Random			Gallows		
Features	**LRA***	**WHCA***	**CBS**	**LRA***	**WHCA***	**CBS**	**LRA***	**WHCA***	**CBS**
RTD_α	6	9	40	11	13	62	13	42	51
AAD_α	3	18	56	7	6	54	13	53	37
RCA_α	9	6	44	15	6	32	31	33	45
NA	4	7	32	5	14	35	34	86	41
TD_α	59	79	57	58	45	41	47	111	50
CA_α	2	4	11	3	2	19	5	36	13
FA_α	16	27	25	11	7	15	10	55	35

4.6.2 Distance and Computation Results

To analyze the performance of the metareasoning the *regret* was measured in relation to the distance and computation. The regret was calculated by using the minimum distance or computation by any algorithm as the optimal value. The difference between the distance or computation for each algorithm and the optimal value is calculated otherwise known as the regret. This metric will identify algorithms that are consistently poor or effective in a certain map and system combination.

The first 25 runs for each system size in a map were used to compare the algorithms. The average regret for the 25 runs for each system size can be seen in Figures 4.1 – 4.6. The computational regret for the CBS algorithm was not plotted in Figures 4.2 and 4.4 because its value was 100 times the next largest computational time.

Tables 4.8 – 4.13 contain the average sum of the distances and computational times for each of the system sizes. In the following tables and figures MetaClass represents the metareasoning method that utilizes the classification learning model and MetaRegress represents the metareasoning method that utilizes the regression learning models.

Table 4.8: Average distance travelled by all agents in the system for a simulation on the warehouse map

Number of Agents	A* Average Distance	WHCA* Average Distance	CBS Average Distance	MetaClass Average Distance	MetaRegress Average Distance
10	926	926	925	926	925
20	2,029	2,030	2,026	2,029	2,022
30	3,184	3,184	3,182	3,184	3,178
40	4,423	4,414	4,415	4,423	4,413
50	5,715	5,716	5,712	5,716	5,714
60	6,815	6,834	6,832	6,815	6,814
70	8,348	8,345	8,332	8,348	8,348

Table 4.9: Average computational time for all agents in the system for a simulation on the warehouse map

Number of Agents	A* Average Computation Time	WHCA* Average Computation Time	CBS Average Computation Time	MetaClass Average Computation Time	MetaRegress Average Computation Time
10	1.837	1.829	2.782	1.845	1.866
20	4.982	3.98	37.53	5.005	5.127
30	6.25	6.334	36.758	6.25	6.464
40	8.392	8.993	120.775	8.383	8.681
50	10.834	11.185	200.447	10.864	11.143
60	13.393	12.626	281.541	13.393	12.72
70	14.765	16.031	302.259	14.854	15.212

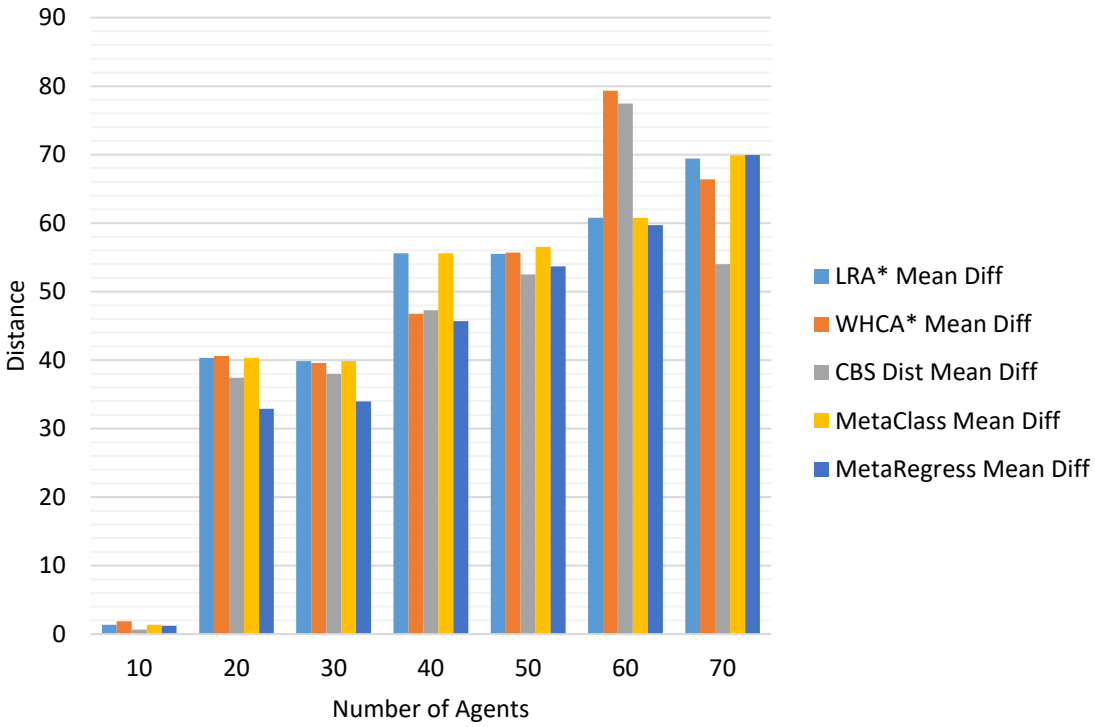


Figure 4.2: Average distance regret for algorithms on the warehouse map

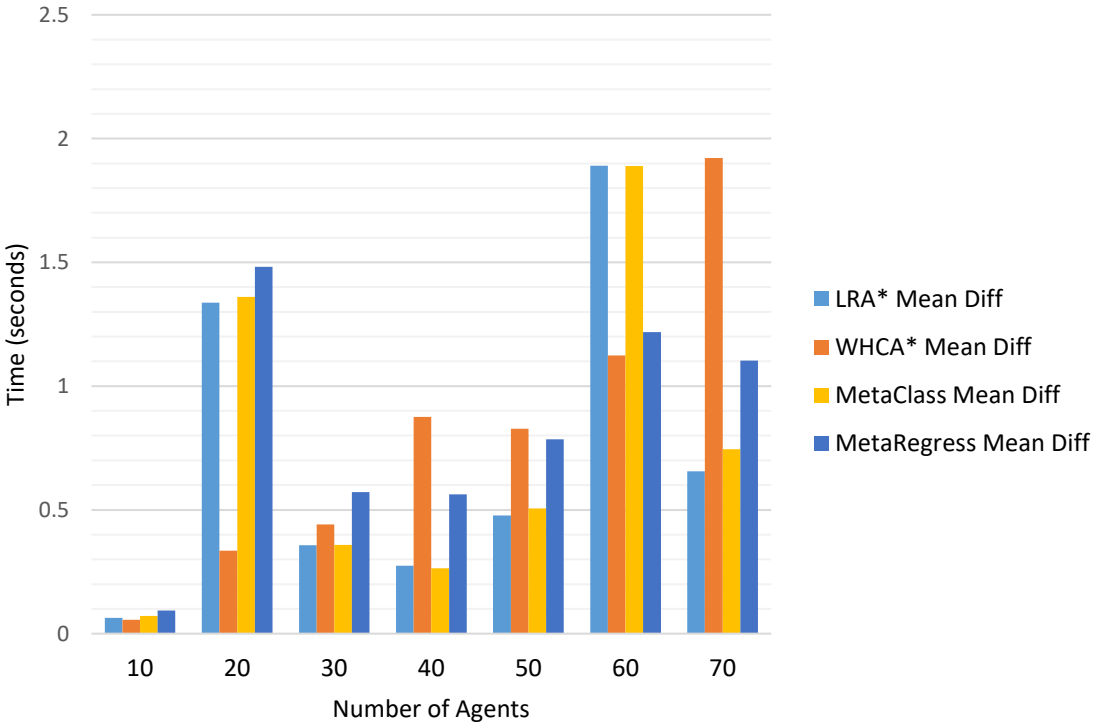


Figure 4.3: Average computational regret for algorithms on the warehouse map

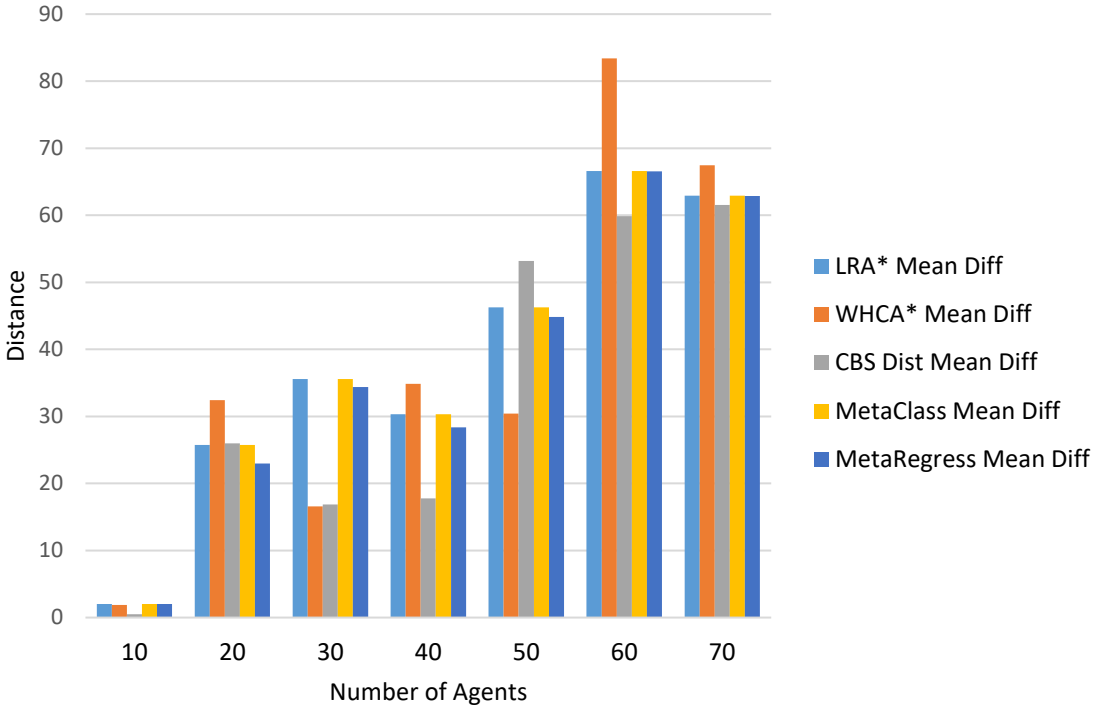


Figure 4.4: Average distance regret for algorithms on the random map

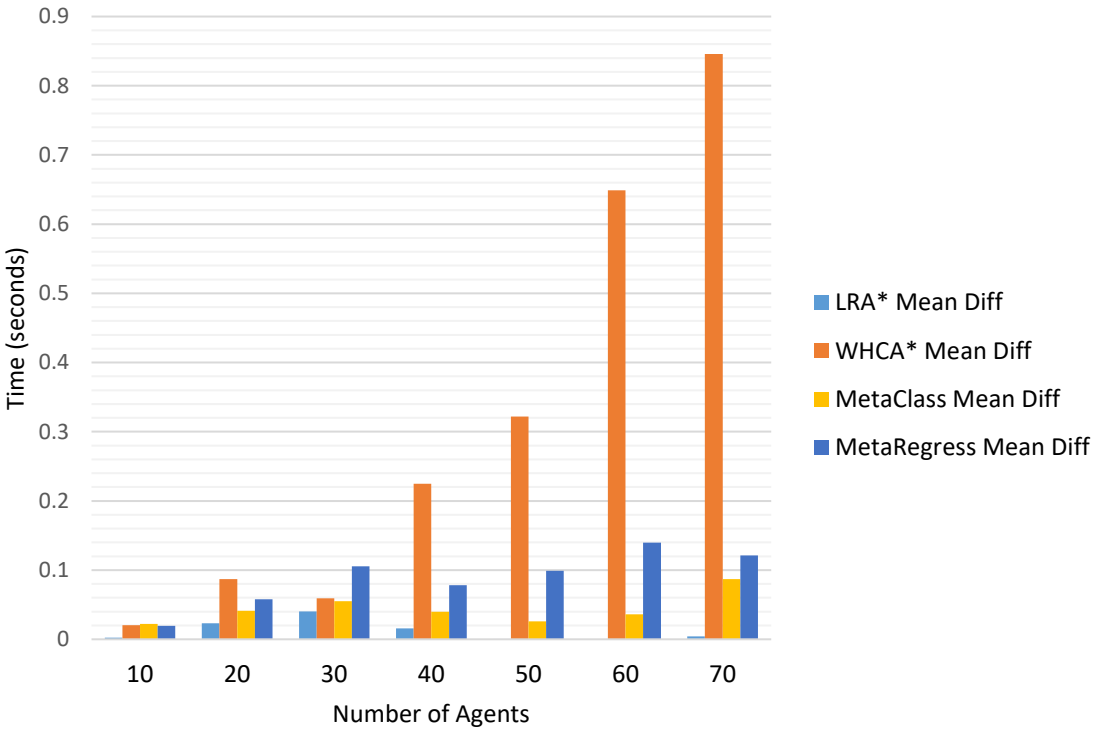


Figure 4.5: Average computational regret for algorithms on the random map

Table 4.10: Average distance travelled by all agents in the system for a simulation on the random map

Number of Agents	A* Average Distance	WHCA* Average Distance	CBS Average Distance	MetaClass Average Distance	MetaRegress Average Distance
10	510	510	508	510	510
20	1,102	1,109	1,102	1,102	1,099
30	1,781	1,762	1,762	1,781	1,780
40	2,533	2,538	2,521	2,533	2,531
50	3,250	3,234	3,257	3,250	3,248
60	4,092	4,110	4,086	4,092	4,092
70	4,870	4,874	4,868	4,870	4,870

Table 4.11: Average computational time for all agents in the system for a simulation on the random map

Number of Agents	A* Average Computation Time	WHCA* Average Computation Time	CBS Average Computation Time	MetaClass Average Computation Time	MetaRegress Average Computation Time
10	0.236	0.254	0.373	0.256	0.253
20	0.538	0.602	3.629	0.556	0.572
30	0.925	0.944	47.376	0.94	0.99
40	1.25	1.459	152.117	1.274	1.312
50	1.49	1.812	300.279	1.516	1.589
60	2.036	2.685	300.492	2.072	2.176
70	2.346	3.187	299.564	2.429	2.463

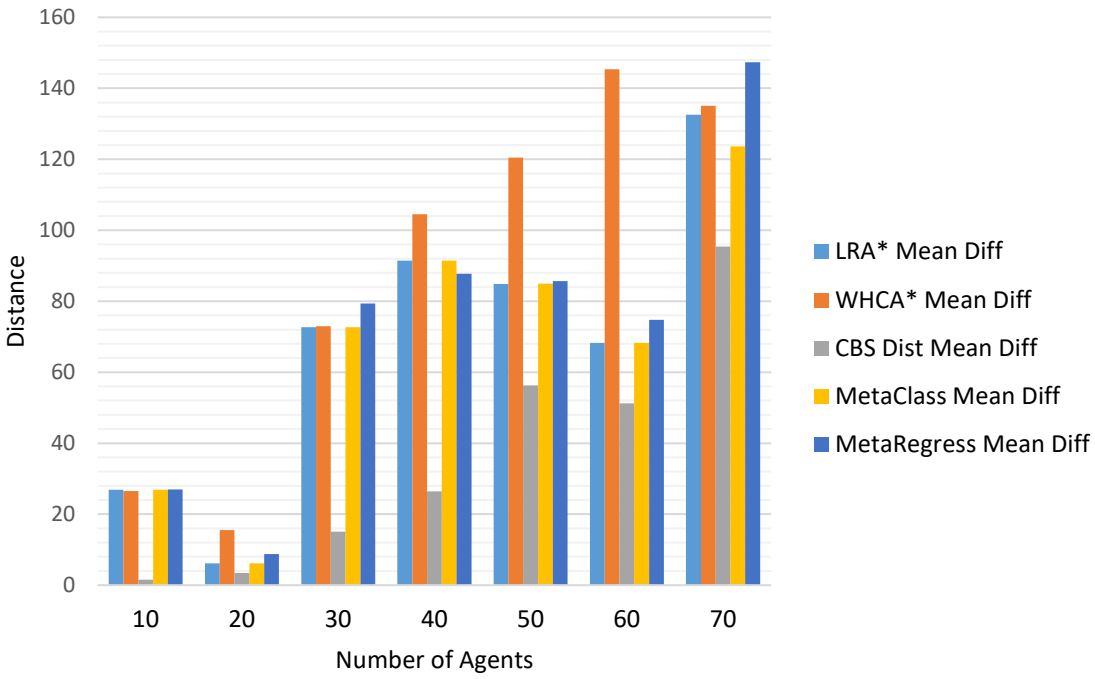


Figure 4.6: Average distance regret for algorithms on the gallows map

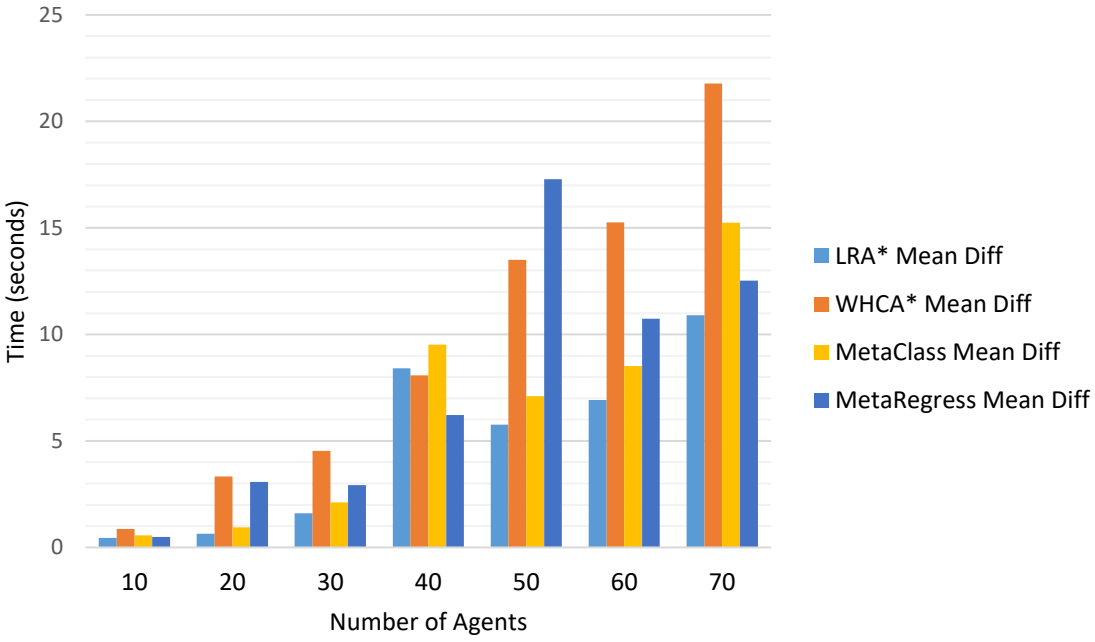


Figure 4.7: Average computational regret for algorithms on the gallows map

Table 4.12: Average distance travelled by all agents in the system for a simulation on the gallows map

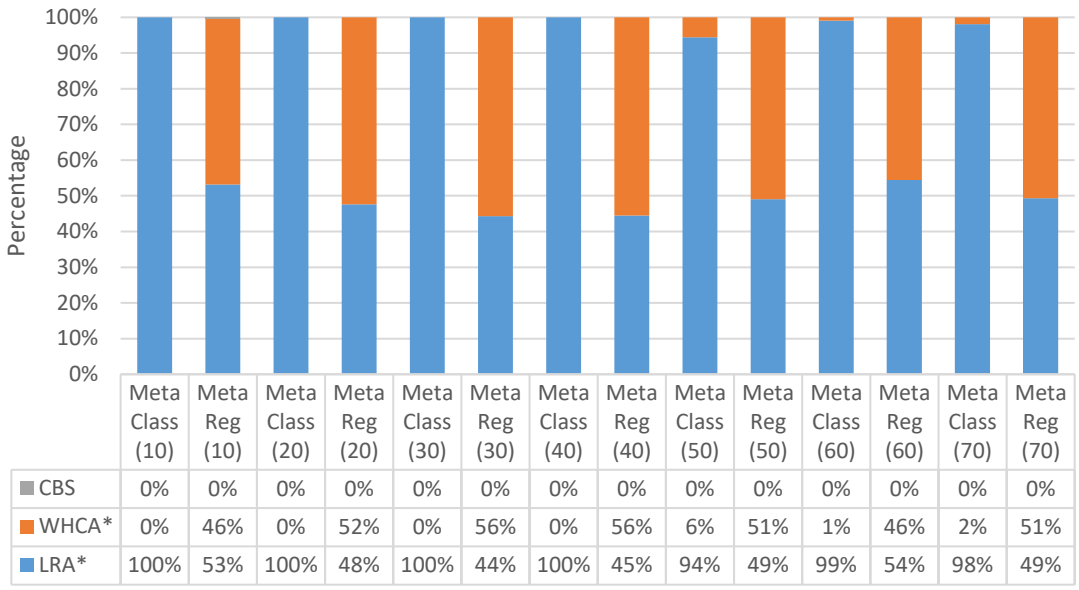
Number of Agents	A* Average Distance	WHCA* Average Distance	CBS Average Distance	MetaClass Average Distance	MetaRegress Average Distance
10	1,180	1,180	1,155	1,180	1,180
20	2,467	2,476	2,464	2,467	2,469
30	3,945	3,945	3,887	3,945	3,951
40	5,426	5,439	5,361	5,426	5,422
50	6,894	6,929	6,865	6,894	6,895
60	8,444	8,521	8,427	8,444	8,450
70	10,209	10,211	10,172	10,200	10,224

Table 4.13: Average computational time for all agents in the system for a simulation on the gallows map

Number of Agents	A* Average Computation Time	WHCA* Average Computation Time	CBS Average Computation Time	MetaClass Average Computation Time	MetaRegress Average Computation Time
10	6.73	7.14	39.93	6.84	6.77
20	15.99	18.69	209.78	16.29	18.43
30	28.81	31.73	309.26	29.32	30.12
40	51.57	51.24	324.26	52.68	49.37
50	63.66	71.39	336.29	64.98	75.18
60	79.4	87.75	347.84	81.01	83.23
70	114.24	125.12	375.16	118.57	115.86

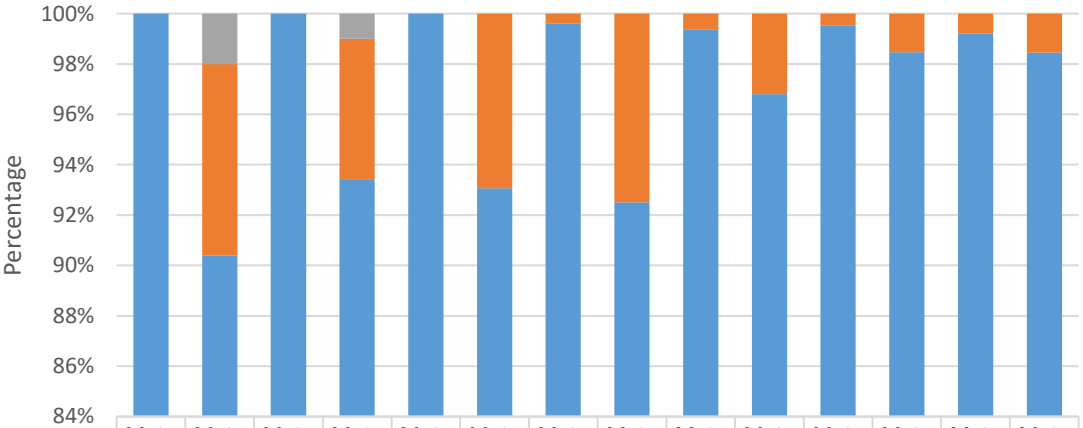
4.6.3 Algorithm Usage

The metareasoning policies allow for different algorithms in the same run. Figures 4.7 – 4.9 shows the percentage of the algorithms used for the different system sizes on each map. The classification and regression are compared to show how the different methods resulted in varying system compositions. In each figure the number of agents in the system was specified in parenthesis under the algorithm name.



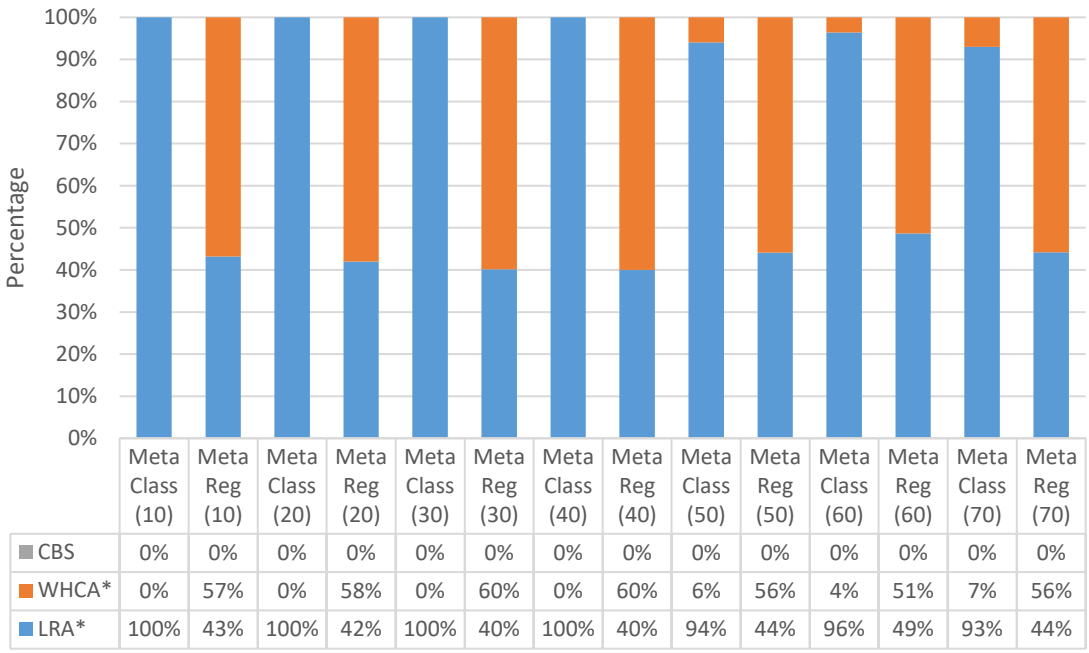
	Meta Class (10)	Meta Reg (10)	Meta Class (20)	Meta Reg (20)	Meta Class (30)	Meta Reg (30)	Meta Class (40)	Meta Reg (40)	Meta Class (50)	Meta Reg (50)	Meta Class (60)	Meta Reg (60)	Meta Class (70)	Meta Reg (70)
CBS	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
WHCA*	0%	46%	0%	52%	0%	56%	0%	56%	6%	51%	1%	46%	2%	51%
LRA*	100%	53%	100%	48%	100%	44%	100%	45%	94%	49%	99%	54%	98%	49%

Figure 4.8: Percentage distributions of algorithms per system size on the warehouse map



	Meta Class (10)	Meta Reg (10)	Meta Class (20)	Meta Reg (20)	Meta Class (30)	Meta Reg (30)	Meta Class (40)	Meta Reg (40)	Meta Class (50)	Meta Reg (50)	Meta Class (60)	Meta Reg (60)	Meta Class (70)	Meta Reg (70)
CBS	0%	2%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
WHCA*	0%	8%	0%	6%	0%	7%	0%	8%	1%	3%	0%	2%	1%	2%
LRA*	100%	90%	100%	93%	100%	93%	100%	93%	99%	97%	100%	98%	99%	98%

Figure 4.9: Percentage distributions of algorithms per system size on the random map



	Meta Class (10)	Meta Reg (10)	Meta Class (20)	Meta Reg (20)	Meta Class (30)	Meta Reg (30)	Meta Class (40)	Meta Reg (40)	Meta Class (50)	Meta Reg (50)	Meta Class (60)	Meta Reg (60)	Meta Class (70)	Meta Reg (70)
CBS	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
WHCA*	0%	57%	0%	58%	0%	60%	0%	60%	6%	56%	4%	51%	7%	56%
LRA*	100%	43%	100%	42%	100%	40%	100%	40%	94%	44%	96%	49%	93%	44%

Figure 4.10: Percentage distributions of algorithms per system size on the gallows map

4.7 Discussion

The accuracy of the classification method yielded about 60% for all the maps. When this information is used in combination with the algorithm breakdowns, the classification method resulted in the LRA* algorithm most of the time. This could be because the classification identifies LRA* if it is tied for the minimum value. It could also mean that the state features used are not effectively separating the data.

This tie breaker method could be the cause of the lack of CBS in the algorithm breakdowns. For example, the CBS algorithm consistently has the least amount of distance regret out of all algorithms, yet it is not represented in any of the classification metareasoning. If the agent never encounters a collision the CBS and LRA* algorithms are equivalent in terms of distance. If the CBS outperforms the LRA* algorithm it must still outperform the WHCA* because a tie with this algorithm results in the WHCA*

selection. These tie breakers could be over correcting for the LRA* and WHCA* methods causing the CBS to be removed from the classification model.

The lack of the CBS in the regression model is due to its large timeout, computational time given for agents to find a set of paths without conflicts. To make the CBS complete the most optimal path, the timeout for the algorithm was increased to 300 seconds. This value produced the least distance regret, but it also increased the regression to values much larger than LRA* and WHCA*.

LRA* and WHCA* were evenly split in the breakdown for the warehouse and gallows maps. This is due to the competitive computational times for the agents for each of these algorithms. The two algorithms require about the same level of effort but when there is no conflicts LRA* dominates and when there are many conflicts WHCA* dominates because of the reduced amount of algorithmic computing. Thus, when a multiagent system is half composed of agents that will not incur a conflict and half that will the agents the algorithm breakdown is split relatively in two. The breakdown results in multiple occasions where the regression model outperforms the two algorithms included in its breakdown. This can be seen with 40 agents in the gallows map and most of the system sizes in warehouse. However, the only instance where the regression outperforms the algorithms included in its breakdown is the 40 agents' gallows scenario.

For both the regression and classification the two least important features were CA_α and FA_α. This means that there may be no relationship between the agent distances directly. Based on the greater importance of the RCA_α feature, it may be more beneficial to relate these features to others.

The random map is unlike the other two in that the LRA* algorithm dominated the WHCA* and CBS algorithms. The environment had fewer restrictions and it is easier to solve the path problem when collisions exist, therefore the overhead WHCA* and CBS has does not provide the same benefit LRA* can come to the same solutions without having to solve all the conflicts a priori or having to populate a reservation table constantly. As the number of agents increases, the LRA* gains usage but one would expect that as the number of agents grows, conflicts would also grow, and the random space would be more constrained than it currently is, and this may not hold.

Chapter 5: Summary

This paper proposed an application of a metareasoning approach that used state features. The state features were chosen so that their values would be different for each agent and they would change as the agent moved through the simulation. This approach was used to allow for independent metareasoning in a decentralized multiagent system. The agents had the ability to control their own reasoning with only their own perceptions and communication. The meta-level was constructed using an extreme gradient boosting machine learning method for both classification and regression.

This approach was used on two different problems in the multiagent field: task allocation and path finding. Two single fixed algorithms (CBAAMM and DHBAMM) were tested in two different scenarios (search and search and visit) for the task allocation problem. A system size of five was tested in this problem along with different numbers of targets and their spacing. The metareasoning model for this experiment was conducted using the XGBoost classification approach to identify when the agent should not be running the task allocation algorithm. The results showed that the number of algorithm runs, and the number of messages were decreased to a fourth of the baseline value. In the CBAAMM instances the performance (distance) metric was nearly equivalent to the baseline while in the DHBAMM application there was a decrease in performance.

For the multiagent path finding problem, three single fixed algorithms were tested (LRA*, WHCA*, and CBS) on three different maps (warehouse, gallows, and random). The system sizes varied from 10 to 70 agents and there was one task for each of the agents. Two different metareasoning models were constructed, one using classification

combined with the distance metric and the other using computational time for regression. The results showed that using this method systems can be constructed where the performance or computation of the metaresoning system can be greater than the single fixed algorithms. It also showed that the combination of algorithms in a system changes depending on the map.

Future work into this metareasoning application may include different sets of algorithms, maps, state features, or machine learning models. While this work uses a few algorithms, others exist that may have different reactions to this application or other machine learning methods that could improve the accuracy. Another direction of study could be the application of the approach on the lifelong path finding problem introduced by Ma *et al.* [34, 35]. Using this type of approach, the agents should be able to sense their environment whenever a new task appears so it may be beneficial to this problem as the system number of tasks to be solved increases.

Bibliography

[1] Ahmadi, Kamilia, and Vicki H. Allan. "Efficient Self Adapting Agent Organizations." In *ICAART (1)*, pp. 294-303. 2013.

[2] Artikis, Alexander. "Dynamic protocols for open agent systems." In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 97-104. 2009.

[3] Bapat, Akshay Vinay. "Development of Decentralized Task Allocation Algorithms for Multi-Agent Systems with Very Low Communication." PhD diss., 2020.

[4] Becker, Raphen, Alan Carlin, Victor Lesser, and Shlomo Zilberstein. "Analyzing myopic approaches for multi-agent communication." *Computational Intelligence* 25, no. 1 (2009): 31-50.

[5] Beul, Marius, David Droeschel, Matthias Nieuwenhuisen, Jan Quenzel, Sebastian Houben, and Sven Behnke. "Fast autonomous flight in warehouses for inventory applications." *IEEE Robotics and Automation Letters* 3, no. 4 (2018): 3121-3128.

[6] Bolu, Ali, and Ömer Korçak. "Adaptive Task Planning for Multi-Robot Smart Warehouse." *IEEE Access* 9 (2021): 27346-27358.

[7] Borghetti, Brett J., and Maria L. Gini. "Weighted Prediction Divergence for Metareasoning." (2011): 249-264.

[8] Brueckner, Sven A. "Swarming Geographic Event Profiling, Link Analysis, and Prediction." In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 71-81. IEEE, 2009.

[9] Brunet, Luc, Han-Lim Choi, and Jonathan How. "Consensus-based auction approaches for decentralized task assignment." In *AIAA guidance, navigation and control conference and exhibit*, p. 6839. 2008.

[10] Busoniu, Lucian, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, no. 2 (2008): 156-172.

[11] Carlin, Alan, and Shlomo Zilberstein. "Bounded rationality in multiagent systems using decentralized metareasoning." In *Decision Making with Imperfect Decision Makers*, pp. 1-28. Springer, Berlin, Heidelberg, 2012.

[12] Carrillo, Estefany, Suyash Yeotikar Sharan Nayak, Mohammad Khalid M. Jaffar, Shapour Azarm, Jeffrey W. Herrmann, Michael Otte, and Huan Xu. “Communication-Aware Multi-agent Metareasoning for Decentralized Task Allocation.” Unpublished manuscript, 2020, typescript.

[13] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785-794. 2016.

[14] Choi, Han-Lim, Luc Brunet, and Jonathan P. How. "Consensus-based decentralized auctions for robust task allocation." *IEEE transactions on robotics* 25, no. 4 (2009): 912-926.

[15] Cooper, John R. "Optimal Multi-Agent Search and Rescue Using Potential Field Theory." In *AIAA Scitech 2020 Forum*, p. 0879. 2020.

[16] Cox, Michael, and Anita Raja. "Metareasoning: A manifesto." *BBN Technical* (2007).

[17] Davoodi, Mohammadreza, Saba Faryadi, and Javad Mohammadpour Velni. "A Graph Theoretic-Based Approach for Deploying Heterogeneous Multi-agent Systems with Application in Precision Agriculture." *Journal of Intelligent & Robotic Systems* 101, no. 1 (2021): 1-15.

[18] de Koster, René. "Automated and robotic warehouses: developments and research opportunities." *Logistics and Transport* 38 (2018): 33-40.

[19] Felner, Ariel, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges." In *Tenth Annual Symposium on Combinatorial Search*. 2017.

[20] Frasheri, Mirgita, Baran Cürüklü, Mikael Esktröm, and Alessandro Vittorio Papadopoulos. "Adaptive autonomy in a search and rescue scenario." In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 150-155. IEEE, 2018.

[21] Fuentealba, Diego, Ismael Soto, Kecheng Liu, and Alejandro J. Martinez. "Tracking system with VLC for underground mine using multi-agent systems." In *2017 First South American Colloquium on Visible Light Communications (SACVLC)*, pp. 1-5. IEEE, 2017.

[22] Håkansson, Anne, and Ronald Hartung. "Using Meta-Agents for Multi-Agents in Networks." (2007): 6.

[23] Herrmann, Jeffrey. "Data-driven metareasoning for collaborative autonomous systems." (2020).

[24] Ismail, Sarah, and Liang Sun. "Decentralized hungarian-based approach for fast and scalable task allocation." In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 23-28. IEEE, 2017.

[25] Kaduri, Omri, Eli Boyarski, and Roni Stern. "Algorithm Selection for Optimal Multi-Agent Pathfinding." In *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, pp. 161-165. 2020.

[26] Kota, Ramachandra, Nicholas Gibbins, and Nicholas R. Jennings. "Decentralized approaches for self-adaptation in agent organizations." *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7, no. 1 (2012): 1-28.

[27] Kuchar, Karel, Eva Holasova, Lukas Hrboticky, Martin Rajnoha, and Radim Burget. "Supervised Learning in Multi-Agent Environments Using Inverse Point of View." In *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 625-628. IEEE, 2019.

[28] Kwon, Woong, Jun Ho Park, Minsu Lee, Jongbeom Her, Sang-Hyeon Kim, and Ja-Won Seo. "Robust autonomous navigation of unmanned aerial vehicles (UAVs) for warehouses' inventory application." *IEEE Robotics and Automation Letters* 5, no. 1 (2019): 243-249.

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

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

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

[32] Langlois, Samuel T., Oghenetekevwe Akoroda, Estefany Carrillo, Jeffrey W. Herrmann, Shapour Azarm, Huan Xu, and Michael Otte. "Metareasoning Structures, Problems, and Modes for Multiagent Systems: A Survey." *IEEE Access* 8 (2020): 183080-183089.

[33] Li, Jiaoyang, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Kumar, and Sven Koenig. "Lifelong multi-agent path finding in large-scale warehouses." *arXiv preprint arXiv:2005.07371* (2020).

[34] Ma, Hang, Jiaoyang Li, T. K. Kumar, and Sven Koenig. "Lifelong multi-agent path finding for online pickup and delivery tasks." *arXiv preprint arXiv:1705.10868* (2017).

[35] Ma, Hang, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7651-7658. 2019.

[36] Mitchell, Rory, Andrey Adinets, Thejaswi Rao, and Eibe Frank. "Xgboost: Scalable GPU accelerated learning." *arXiv preprint arXiv:1806.11248* (2018).

[37] Nielsen, Didrik. "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?." Master's thesis, NTNU, 2016.

[38] Nissim, Raz, and Ronen I. Brafman. "Multi-agent A* for parallel and distributed systems." In *ICAPS Workshop on Heuristics and Search for Domain-Independent Planning*, pp. 43-51. 2012.

[39] Noda, Itsuki, and Masayuki Ohta. "Meta-level control of multiagent learning in dynamic repeated resource sharing problems." In *Pacific Rim International Conference on Artificial Intelligence*, pp. 296-308. Springer, Berlin, Heidelberg, 2008.

[40] Pěchouček, Michal, Jan Tožička, and Vladimír Mařík. "Meta-reasoning methods for agent's intention modelling." In *International Workshop on Autonomous Intelligent Systems: Agents and Data Mining*, pp. 134-148. Springer, Berlin, Heidelberg, 2005.

[41] Pentjuss, Agris, Aleksejs Zacepins, and Aleksandrs Gailums. "Improving precision agriculture methods with multiagent systems in Latvian agricultural field." *Engineering for rural development, Latvia, Jelgava* (2011): 109-114.

[42] Piardi, Luis, Vivian Cremer Kalempa, Marcelo Limeira, André Schneider de Oliveira, and Paulo Leitão. "ARENA—augmented reality to enhanced experimentation in smart warehouses." *Sensors* 19, no. 19 (2019): 4308.

[43] Pinyol, Isaac, and Jordi Sabater-Mir. "Metareasoning and social evaluations in cognitive agents." In *International Conference on Autonomic Computing and Communications Systems*, pp. 220-235. Springer, Berlin, Heidelberg, 2009.

[44] Puljiz, David, Gleb Gorbachev, and Björn Hein. "Implementation of augmented reality in autonomous warehouses: challenges and opportunities." *arXiv preprint arXiv:1806.00324* (2018).

[45] Raja, Anita, and Victor Lesser. "A framework for meta-level control in multi-agent systems." *Autonomous Agents and Multi-Agent Systems* 15, no. 2 (2007): 147-196.

[46] Robots, Fully Autonomous. "The Warehouse Workers of the Near Future." *Robbie Whelan-http://www. wsj. com/articles/fully-autonomous-robots-the-warehouse-workers-of-the-near-future-1474383024*.

[47] Rubinstein, Zachary B., Stephen F. Smith, and Terry L. Zimmerman. "14 The Role of Metareasoning in Achieving Effective Multiagent Coordination." *Metareasoning: Thinking about Thinking* (2011): 217.

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

[49] Salzman, Oren, and Roni Stern. "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems." In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1711-1715. 2020.

[50] Sharon, Guni, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Conflict-based search for optimal multi-agent pathfinding." *Artificial Intelligence* 219 (2015): 40-66.

[51] Sharon, Guni, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding." *SoCS* 1 (2012): 39-40.

[52] Sharon, Guni, Roni Stern, Meir Goldenberg, and Ariel Felner. "The increasing cost tree search for optimal multi-agent pathfinding." *Artificial Intelligence* 195 (2013): 470-495.

[53] Sigurdson, Devon, Vadim Bulitko, Sven Koenig, Carlos Hernandez, and William Yeoh. "Automatic algorithm selection in multi-agent pathfinding." *arXiv preprint arXiv:1906.03992* (2019).

[54] Silver, David. "Cooperative Pathfinding." *Aiide* 1 (2005): 117-122.

[55] Stentz, Anthony. *Optimal and efficient path planning for unknown and dynamic environments*. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1993.

[56] Stern, Roni, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks." *arXiv preprint arXiv:1906.08291* (2019).

[57] Sturtevant, Nathan R. "Benchmarks for grid-based pathfinding." *IEEE Transactions on Computational Intelligence and AI in Games* 4, no. 2 (2012): 144-148.

[58] Wagner, Glenn, and Howie Choset. "Subdimensional expansion for multirobot path planning." *Artificial Intelligence* 219 (2015): 1-24.

[59] W. Zhao, Q. Meng, and P. W. Chung, “A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario,” IEEE transactions on cybernetics, vol. 46, no. 4, pp. 902–915, 2015. 55

[60] Wurman, Peter R., Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses." *AI magazine* 29, no. 1 (2008): 9-9.

[61] Xie, Jing, and Chen-Ching Liu. "Multi-agent systems and their applications." *Journal of International Council on Electrical Engineering* 7, no. 1 (2017): 188-197.

[62] Xuan, Ping, Victor Lesser, and Shlomo Zilberstein. "Communication decisions in multi-agent cooperation: Model and experiments." In *Proceedings of the fifth international conference on Autonomous agents*, pp. 616-623. 2001.

[63] Zafar, Kashif, Shahzad Badar Qazi, and A. Rauf Baig. "Mine detection and route planning in military warfare using multi agent system." In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 2, pp. 327-332. IEEE, 2006.

[64] Zelinsky, Alexander. "A mobile robot navigation exploration algorithm." *IEEE Transactions of Robotics and Automation* 8, no. 6 (1992): 707-717.

[65] Zhang, Chongjie, and Victor Lesser. "Coordinating multi-agent reinforcement learning with limited communication." In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 1101-1108. 2013.

[66] Zilberstein, Shlomo, and Alan Carlin. "Bounded rationality in multiagent systems using decentralized metareasoning." In *NIPS 2011 workshop Decision Making with Multiple Imperfect Decision Makers*. 2011.