LEARNING THE AGENT SPECIFIC SUB-OPTIMAL BOUND FOR MULTI-AGENT PATH FINDING

by

Ramisa Fariha Prova
20301001

S. M. Fuad Hassan
18301070

Bijoya Sarker
20301352

Shariful Islam
24341077

Md. Shamiul Islam Khan Ishrak
20301235

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
October 2024

Declaration

It is hereby declared that

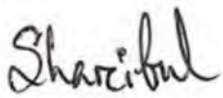
1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

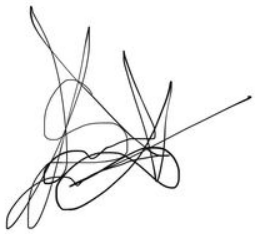
Ramisa Fariha Prova
20301001

S. M. Fuad Hassan
18301070

Bijoya Sarkar
20301352



Shariful Islam
24341077



Md. Shamiul Islam Khan Ishrak
20301235

Approval

The thesis titled **Learning The Agent Specific Sub-Optimal Bound For Multi-Agent Path Finding** submitted by

1. Ramisa Fariha Prova (ID: 20301001)
2. S. M. Fuad Hassan (ID: 18301070)
3. Bijoya Sarker (ID: 20301352)
4. Shariful Islam (ID: 24341077)
5. Md. Shamiul Islam Khan Ishrak (ID: 20301235)

of Summer, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 22, 2024.

Examining Committee:

Supervisor:
(Member)

Md. Ahasanul Alam
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

In the research conducted for this thesis titled "Learning the Agent-Specific Sub-Optimal Bound for Multi-Agent Path Finding," ethical considerations have been of utmost importance. This work adheres to the principles of academic integrity, ensuring that all research findings are reported truthfully and transparently.

1. **Research Integrity:** All data used in this research has been collected and analyzed in accordance with ethical research standards. Any algorithms or models developed have been rigorously tested to ensure reliability and validity.

2. **Contribution to Knowledge:** The goal of this research is to contribute to the body of knowledge in the field of multi-agent systems and pathfinding algorithms. The findings aim to enhance understanding and improve applications in various domains, including robotics and artificial intelligence, while considering the potential societal impacts.

3. **Respect for Collaboration:** This research acknowledges the contributions of other scholars and researchers. All sources of information, including literature, datasets, and prior research, have been properly cited to respect intellectual property rights and to give credit to original authors.

By adhering to these ethical principles, this research aims to advance the understanding of multi-agent pathfinding algorithm while maintaining the highest standards of ethical conduct.

Abstract

Multi-Agent Path Finding (MAPF) is a prominent challenge in robotics and artificial intelligence, encompassing the task of finding collision-free paths for multiple agents sharing a common environment. CBS, a fundamental component of this research, plays a pivotal role in resolving conflicts encountered during path planning. There can be conflict between vertices (V) or there can be conflict in edges (E). The more agents there are in the environment, the higher the potential for collisions. With a greater number of agents, the likelihood of agents intersecting paths or occupying the same space at the same time increases. However, CBS typically aims for optimality, which can be computationally expensive and not always practical in real-time scenarios. To enhance the scalability and adaptability of MAPF, this study advocates for sub-optimal path planning, which takes into account agent-specific objectives and constraints. By relaxing the pursuit of optimal solutions, the approach reduces computational complexity and accommodates diverse agent requirements. Sub-optimal paths offer a deal between solution quality and computational effectiveness.

Furthermore, the integration of ML models into MAPF augments its capabilities. Machine learning models can capture complex environmental dynamics and agent behaviors, enabling the prediction of future states and proactive path adjustments. This introduces an adaptive element to MAPF, where agents can dynamically adapt their paths based on real-time data and predictions.

In conclusion, this research advocates a novel approach to MAPF by combining CBS, sub-optimal path planning tailored to individual agents, and the utilization of machine learning models. The synergy of these components offers a promising avenue for addressing complex multi-agent path-finding problems in a scalable, adaptive, and efficient manner, opening new possibilities for real-world applications in robotics and AI.

Key words: *MAPF, Multi Agent System, Agent Coordination, Complexity, CBS, Sub-Optimal, Machine learning.*

Dedication

This thesis is dedicated to our beloved parents, whose unwavering support and encouragement have been the foundation of our academic journey. Their sacrifices and love have inspired us to pursue our dreams with determination.

We also dedicate this work to our supervisor, Md. Ahasanul Alam, for his invaluable guidance, expertise, and encouragement throughout this research. His insights and support have been instrumental in shaping this thesis.

Finally, we extend our heartfelt gratitude to all those who have contributed to this endeavor, whether through assistance, encouragement, or inspiration. Your support has made this achievement possible.

Table of Contents

List of Figures

Chapter 1

Introduction

In recent years, the fields of artificial intelligence (AI) and robotics have experienced incredible advancements, enabling the creation of systems that can now perform complex tasks autonomously in real-time. A significant obstacle in these systems, particularly those involving multiple agents such as self-governing robots, cars, or other entities moving through the same area simultaneously, is guaranteeing that every agent can navigate effectively while minimizing collisions [1]. This is the primary rationale behind the Multi-Agent Path-finding (MAPF) problem, an essential area of AI research focusing on scheduling collision-free or optimal paths for multiple agents operating in a shared environment [2]. By minimizing the total cost of movement, including time and distance traveled, the objective is to ensure that every agent gets from its starting position to its respective goal without colliding with others [3].

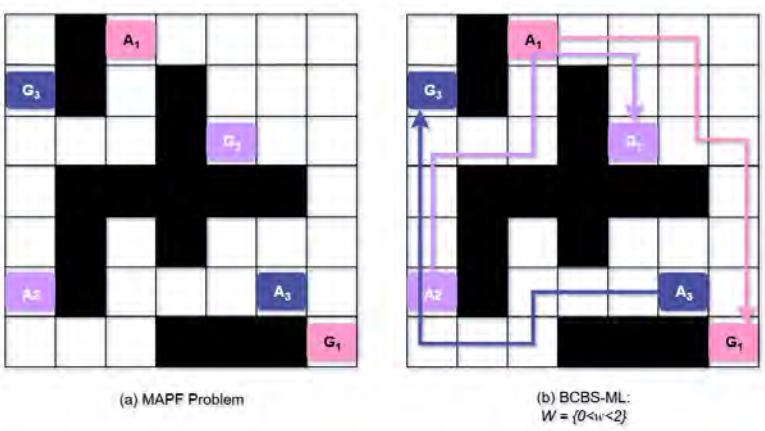


Figure 1.1: *Multi-Agent Path Finding*

MAPF offers a wide range of real-world uses, for instance, in autonomous warehouses where fleets of robots are required to operate through confined spaces to move goods quickly and safely while adhering to schedules [4]. To sustain high throughput in logistics and fulfillment centers, systems such as Amazon's Kiva robots rely significantly on MAPF algorithms [4]. In video game development, MAPF creates smoother gameplay experiences by enabling non-player characters (NPCs) to navigate virtual landscapes and engage in conflict-free interactions with other characters or objects [5]. In air traffic management systems, MAPF assists in coordinating the safe landing, take-off, and taxiing of numerous aircraft in congested airports, enhancing both operational efficiency and safety [6]. Furthermore, MAPF guarantees effective and collision-free navigation in disaster response scenarios involving numerous autonomous agents—such as drones or rescue robots—which improves the reliability of search and rescue operations [7]. These applications demonstrate the value of MAPF in situations that require efficient coordination between several entities.

Although the MAPF problem has a wide range of applications, optimally solving it poses significant computational difficulties, particularly when the environment's complexity or the number of agents increases [8]. With additional agents, it becomes exponentially harder to coordinate them to avoid vertex and edge collisions, making path planning computationally expensive [3]. Since every possible conflict between agents must be taken into account, traditional algorithms such as A* are inefficient in multi-agent scenarios [9]. Different algorithms have been proposed to address this inefficiency, with the **Conflict-Based Search (CBS)**—which implements a two-level search strategy to resolve conflicts and create paths for agents—emerging as a leading method for optimally solving MAPF [9]. However, because CBS depends on generating an exponentially increasing constraint tree, it is computationally costly in large and complex environments, which hinders its scalability [10]. As a countermeasure, several sub-optimal CBS variants have been proposed to improve computational efficiency, including **Enhanced CBS (ECBS)** and **Agent-Specific Sub-Optimal Bounding CBS (ASB-ECBS)**. In particular, ECBS enables faster outcome computation, while ASB-ECBS gives each agent a set of sub-optimal boundaries based on discrete conflict scenarios [8]. Despite these methods improving runtime, they are unable to cope with large-scale or dynamic environments, as they impose fixed parameters universally. However, recent developments in **Machine Learning (ML)** have created new opportunities for addressing MAPF problems. These approaches can now reduce the system's computational overhead by predicting potential conflicts and adjusting pathways in real-time, thus avoiding conflicts before they arise [11].

Building on the recent advancements in ML, we propose a novel strategy that incorporates machine learning into the CBS framework to improve efficiency and scalability. Specifically, we present the application of **Linear Regression** models to predict sub-optimal boundaries tailored to each agent's specific conditions, including environmental complexity, conflict density, and the distance between start and target positions. The fundamental idea underlying our approach is that different agents require different levels of optimization; for instance, less optimization is needed for agents with fewer conflicts, and stronger constraints are applied to those in conflict-heavy areas. Thus, our approach improves system performance while reducing computational time by dynamically changing sub-optimal bounds during pathfinding. In large-scale environments, this ensures improved resolution of conflicts and increased computational efficiency.

Overall, this study offers an improved MAPF framework that incorporates machine learning, which enhances performance and scalability over conventional CBS-based approaches. Our method reduces computing costs without compromising the quality of the solution. Thus, it can be applied in various industries, like robotics, autonomous vehicles, and logistics, for predicting agent-specific sub-optimal bounds.

The following summarizes the main contributions of this paper:

- A machine learning-enhanced CBS framework: We propose a dynamic strategy that integrates linear regression models into CBS to estimate sub-optimal boundaries for agents based on environmental characteristics, including conflict density, start-goal distances, and agent density. This method improves computing efficiency, especially in dynamic, large-scale MAPF scenarios.
- Increased computational efficiency and scalability: By customizing sub-optimal bounds to each agent's unique requirements, our approach reduces runtime and the low-level search space, enabling quicker and more effective searches in complicated situations.
- Comparative study of ML-enhanced CBS variants: We empirically show that our approach performs substantially better than ASB-ECBS and regular CBS, even in situations with different degrees of complexity.

- Extensive research: We offer thorough experimental findings to compare our method's performance with that of current methods. Findings show that incorporating machine learning improves performance in both sparse and dense situations by reducing runtime and node formation.

The layout for the remaining sections of the paper is as follows:

We examine the current status of research on MAPF, CBS, and its sub-optimal variations in Chapter 2. Furthermore, we offer the contextual information required to comprehend the proposed methodology. In Chapter 3, we describe our proposed approach to incorporating machine learning into CBS, with an emphasis on utilizing linear regression to dynamically predict sub-optimal boundaries for agents. The experimental results are presented in Chapter 4, where the effectiveness of our ML-enhanced CBS is compared with that of conventional approaches. Finally, the study concludes in Chapter 5, where we analyze the ramifications of our findings and suggest intriguing possibilities for further research.

Chapter 2

Background and Related Works

The key concepts and relevant research on Multi-Agent Path Finding (MAPF) are covered in this section, along with the challenges encountered in finding optimal solutions for cases involving multiple agents. To get started, we are going to define the MAPF problem.

2.1 Problem Formulation

Let's assume, a Multi Agent Path Finding or MAPF instance represents a group of moving agents $A = \{a_1, \ldots, a_n\}$ navigating on a map G, where, $G = (V, E)$, and V is the set of vertices, and E represents the edges. Two vertices v_i and v_j are connected by an edge e if $e \in E$ and $v_i \neq v_j$, meaning no self-loops are allowed. Typically, the graph G is connected, ensuring that every node is reachable from any other node. Each agent a_i starts from an initial position $\mathcal{S} = (s_1, \ldots, s_n)$ and has a goal location $\mathcal{G} = (g_1, \ldots, g_n)$, where $s_i, g_i \in V$. The objective is to plan paths for each agent such that they move from their respective start locations to their goal locations without any collisions.

At any given discrete time step $t \in \mathbb{N}$, let $v_i(t)$ represents the vertex currently occupied by agent a_i. At the next time step $t + 1$, the agent can move to a neighboring node or stay at its current position. The set of candidate nodes is $C_i = \{v_i(t)\} \cup \{v \mid (v, v_i(t)) \in E\}$, meaning that agent a_i can either stay on its current node or move to one of its adjacent nodes. Every movement incurs a cost of 1.

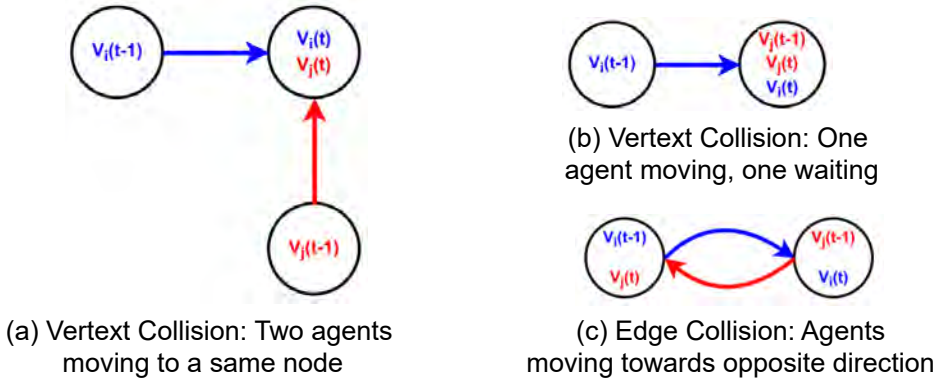


Figure 2.1: *Different types of collisions in a MAPF problem.*

In this framework, agents must avoid two types of collision while moving, (i) vertex collisions and (ii) edge collisions. A vertex collision usually takes place when a multiple numbers of agents try to occupy a node simultaneously, meaning $v_i(t) \neq v_j(t)$. In Figure 2.1 (a), two agents are trying

to move into the same vertex, resulting in a vertex collision. In Figure 2.1 (b), one agent remains stationary while another attempts to move into its position, which also counts as a vertex collision.

The second type of collision is an edge collision, where more than one agent tries to cross paths in opposite directions. Specifically, $v_i(t+1) \neq v_j(t) \vee v_j(t+1) \neq v_i(t)$. In Figure 2.1 (c), two agents are trying to swap places, leading to a head-on collision, which is classified as an edge collision. However, cyclic rotations are allowed, where a sequence such as $v_i(t+1) \rightarrow v_j(t) \wedge v_j(t+1) \rightarrow v_k(t) \wedge \cdots \wedge v_l(t+1) \rightarrow v_i(t)$ is possible.

A **solution** π for a MAPF problem consists of a set of collision-free routes, $\pi = \{\pi_1, \ldots, \pi_n\}$, for every individual agent $a_1, \ldots, a_n$ enabling each agent to reach its goal by a certain time step $T \in \mathbb{N}, T \geq 0$. To be specific, the solution generates a path $\pi_i = (v_i(0), v_i(1), \ldots, v_i(T))$ to every agent, where $v_i(0) \rightarrow s_i$, $v_i(T) \rightarrow g_i$, and the paths are free from conflicts. The quality of the solution can be measured using either the (i) Makespan or (ii) Sum of Individual Costs (SIC). SIC represents the total number of time steps required for all agents to reach their destinations (Equation 2.1), while Makespan is the longest time taken by any individual agent to reach its goal (Equation 2.2). The symbol $|\pi_i|$ refers to the length of the path for agent a_i.

$$SIC = \sum_{i=1}^{n} \mid (\pi_i) \mid, \; where \, \pi_i \in \pi \tag{2.1}$$

$$Makespan = \max(\mid (\pi_i) \mid) \; ; \; \pi_i \in \pi \tag{2.2}$$

2.2 Literature review

Based on the problem formulation described above, the following section discusses the recent MAPF algorithms, with the assistance of a working example. Then it outlines some issues of the existing approaches. Subsequently, some recent works on ML related to Multi-Agent Path Finding is briefly highlighted in the last section of this chapter. (Section 2.3).

2.2.1 CBS (Conflict-Based Search)

Conflict-Based Search (CBS) is a powerful algorithm for tackling the challenging Multi-Agent Path Finding (MAPF) problem, where multiple agents must reach their respective goals while avoiding collisions. CBS operates on a two-level strategy, incorporating elements of both coupled and decoupled methods [6]. At its core is the Constraint Tree (CT), where each node represents a set of constraints and an agent solution, ensuring collision avoidance [6]. CBS employs a best-first search on the CT, prioritizing nodes by cost. When all paths within a node are valid and conflict-free, it becomes a goal node, marking the solution's success. The lower-level search in CBS is dedicated to finding consistent paths for individual agents while respecting their constraints [12]. These paths are cross-validated for conflicts, and if any arise, CBS splits the node into children nodes, each addressing a conflict by introducing a constraint. This iterative process continues until a conflict-free solution is achieved. CBS offers various suboptimal solvers tailored to specific needs:

GCBS (Greedy Conflict-Based Search): GCBS employs a best-first search within the CBS framework to find solutions for MAPF efficiently. It prioritizes runtime over solution quality [12]. The basic principle behind GCBS is to prioritize CT nodes that appear to be closer to a goal node (in terms of depth in the CT, also known as distance-to-go).

BCBS (Bounded Conflict-Based Search): BCBS, or Bounded CBS, is an algorithm used in multi-agent pathfinding and task allocation problems. It seeks to find solutions that are suboptimal but bounded by a given factor, where the factor is determined by the parameters wH and wL. The main idea behind BCBS is to use focal search techniques at both the high level and low level of the CBS algorithm to efficiently explore the search space [12].

ECBS (Enhanced Conflict-Based Search): Enhanced CBS (ECBS) is an enhancement of the Conflict-Based Search (CBS) algorithm for multi-agent pathfinding. It simplifies the distribution of search parameters, making it more domain-independent. ECBS employs a lower bound (LB) estimation on best solution costs, improving search efficiency. Nodes in the FOCAL set are selected based on LB and cost, ensuring that solutions found are within a bounded suboptimality factor w from the optimal cost. ECBS offers flexibility by adapting to low-cost solutions, enhancing practical performance. Both BCBS and ECBS maintain completeness, guaranteeing a solution if one exists while efficiently navigating complex multi-agent scenarios [12].

In addition to CBS-based solvers, several other MAPF algorithms are available, such as PPS (Parallel Push and Swap) which uses a rule-based approach for generating unbounded suboptimal solutions [13], and Priority Inheritance with Backtracking (PIBT) algorithm which is a decentralized approach used in Multi-Agent Path Finding (MAPF) for coordinating agents in environments like grids or graphs [14]. According to [6], a node within the Conflict Tree (CT) is considered a goal node when its solution satisfies all constraints, ensuring no conflicts among the paths of all agents. The high-level CBS utilizes a best-first search on the CT, prioritizing nodes based on their costs [6]. In cases of ties, preference is given to CT nodes with solutions involving fewer conflicts, and any remaining ties are resolved in a first-in, first-out (FIFO) manner.

The high-level algorithm employs a best-first search approach, illustrated through a mice-to-cheese scenario in Figure 2.2. In this scenario, Figure 2.2(a) represents the destination node from the starting node, and Figure 2.2(b) shows the Constraint Tree. The algorithm aims to find the best solutions for each agent, such as agent m_1 following the path $< S_1, A_1, C, G_1 >$ and agent m_2 following $< S_2, B_1, C, G_2 >$, resulting in a total cost of 6. During expansion, a conflict arises when both agents reach vertex C at time step 2 (conflict: $m_1, m_2, C, 2$), marking the root as non-target and leading to two child nodes to resolve the conflict. One child includes constraint $(m_1, C, 2)$, while the other adds $(m_2, C, 2)$. The search algorithm is then applied to find the best path adhering to the new constraints. For the left child, m_1 waits at $S_1 (or A_1)$, resulting in a path $< S_1, A_1, A_1, C, G_1 >$, with a total cost of 7. The right child is generated similarly with a cost of 7. Ultimately, the left child is chosen for expansion, a validated path, and becomes the target node, providing the optimal solution without conflicts.

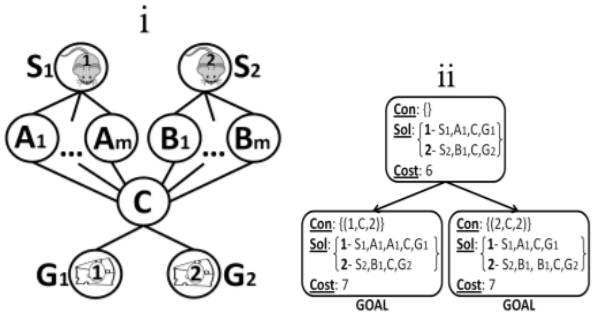


Figure 2.2: *(a) Example instance of MAPF (b) Constraint tree related to (a)*

Imagine another problem where we have two agents, Green and Yellow, trying to get from one place to another in Figure 2.3. At first, they have their paths planned, but they collide at a point called C at the same time in Figure 2.3(a). To fix this, a special method called Conflict-Based

Search (CBS) splits the problem into two smaller parts (Figure 2.3(b)). In one part, Green gets a new path that goes $[A, B, B, C, F]$, and Yellow keeps the same path. In the other part, Yellow gets a new path $[D, H, H, C, E]$, while Green keeps its original path. These new paths cost a bit more (7 instead of 6) because of the changes. However, the CBS chooses the first part to work on because it has no collisions. So, it gives us a solution where Green goes $[A, B, B, C, F]$ and Yellow goes $[D, H, H, C, E]$, with a total cost of 7. This is the best solution without any collisions and provides the optimal solution.

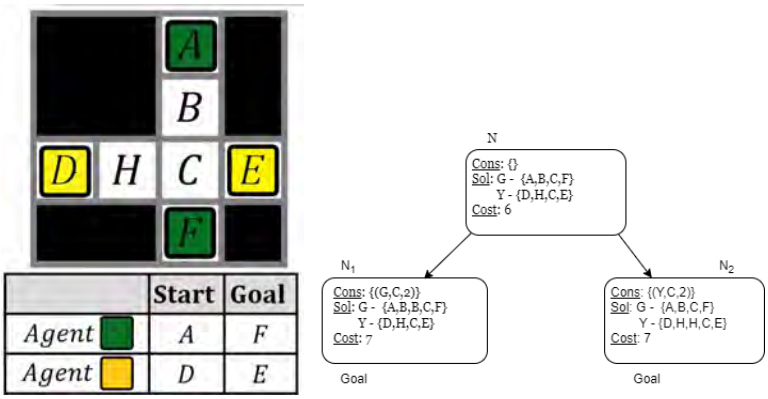


Figure 2.3: *(a) MAPF instance (b) Constraint tree (CT)*

2.2.2 Limitations of CBS Algorithm

Despite the efficacy, CBS still has a few shortcomings. The drawback of CBS and its variants is their limited scalability when dealing with a huge number of agents according to their NP-hard nature. While these algorithms aim for optimal solutions and conflict resolution, they become inefficient and computationally intensive when confronted with a substantial number of agents. This limitation can result in impractical computation times in real-world scenarios where many agents need to be coordinated.

To address the aforementioned problems, we present ASB-ECBS, a method that tailors sub-optimal bounds to each requirement of agents for locating conflict-free routes to their objectives. This specialized method reduces the search space of low-level, so that will be only for greater W values, which will develop the run-time of MAPF solver.

2.2.3 Agent-Specific Sub-optimal Bounding

ASB-ECBS is presented as a new strategy for tackling the MAPF problem, which configures sub-optimal boundaries for each agent. The purpose of this adaption is to improve the runtime and reduce the search space of low-level in the bounded sub-optimal MAPF solutions. It has two variants, SASB-ECBS and DASB-ECBS, which provide different agent conflict resolution needs. [8]. The key points covered in this section include:

Motivation for ASB-ECBS: ASB-ECBS understands that not all agents require the same global sub-optimal bound (”W”) [8]. Such agents may not benefit by higher bounds and hence to provide a more effective resolution of conflict, ASB-ECBS calculates sub-optimal bounds as per the conflict constraints of every agent individually.

Adaptive Sub-Optimal Bound Assignment: Initially in the search procedure, SASB-ECBS provides sub-optimal bounds based on initial agent requirements, while DASB-ECBS dynamically adjusts bounds during run-time to adapt to changing conflict conditions.

Completeness: ASB-ECBS is a constructive search algorithm, meaning that it is a complete algorithm and would always arrive at a solution as long as one exists in the solution space. It is an improved adaptive version of CBS which is famous for its completeness and optimism. [8].

Algorithm Overview: This section provides an overview of each fundamental step involved in the ASB-ECBS algorithm, initializing the agent routes, identifying the sub-optimal bounds of each agent, performing the high-level and low-level search [8] and finally, resolving the contradiction between the agents to acquire the solution.

As pointed out in [8], ASB-ECBS enhances multi-agent path planning by providing sub-optimal bounds specific to every agent and utilization of various conflict scenarios. It comes in two versions: Static-ASB-ECBS (SASB-ECBS) and Dynamic-ASB-ECBS (DASB-ECBS), depending on agent conflict resolution requirements [8].

Static ASB-ECBS (SASB-ECBS): By extending ECBS, SASB-ECBS gives agents distinct sub-optimal bounds according to their conflict circumstances [8]. The bounds remain constant during pathfinding, and the algorithm starts by selecting the best paths for each agent independently. Agents receive sub-optimal bounds proportional to their conflicts, and high-level search prioritizes nodes with fewer conflicts. The algorithm returns a solution upon reaching a goal node or continues to resolve conflicts until a viable solution is found.

Dynamic ASB-ECBS (DASB-ECBS): During the search process, DASB-ECBS dynamically modifies agent-specific sub-optimal boundaries as conflict needs change [8]. It considers varying conflict levels and gradually increases agent-specific weights as conflicts decrease, expanding the low-level search space for more efficient calculations [8]. DASB-ECBS uses a "weight_assignment" function similar to SASB-ECBS but periodically updates sub-optimal bounds based on real-time which the agent needs to enhance the search process's effectiveness.

As shown in [8], ASB-ECBS maintains bounded sub-optimality. The authors' extensive experiments on various maps have shown that SASB-ECBS and DASB-ECBS outperform current MAPF algorithms by decreasing run-time and low-level search space in different conditions of environments [8].

2.2.4 Algorithm Description

The ASB-ECBS algorithm is a sub-optimal variation of the Conflict-Based Search (CBS) algorithm [15]. It guarantees a W-sub-optimal solution for the Multi-Agent Path Finding (MAPF) problem by applying a focal-search technique, where W represents a manually set global upper limit on the solution cost. The ASB-ECBS algorithm consists of two stages, as described in Algo. 1 [8]. To find a solution, ASB-ECBS uses a two-level focused search. At a high level, a constraint tree is constructed to determine the set of constraints that lead to a feasible solution for the MAPF problem. A high-level node N contains three elements: i) a set of constraints to resolve conflicts, ii) potential paths for k agents that satisfy the constraints, and iii) the solution cost [Algo. 1: lines 1-3]. ASB-ECBS then applies focused search at the low level to compute sub-optimal paths for the agents while adhering to the high-level constraints [8].

Both the high-level and low-level searches maintain two types of priority lists: $OPEN$ and $FOCUL$. The high-level $OPEN$ contains all high-level nodes sorted by the cost function $f(N)$. The function $f(N)$, defined as $f(N) = \sum_{i=1}^{k} f^i_{min}(n)$, represents a lower bound on the solution cost for the constraint tree node N. In the low-level search, $f^i(n)$, which is defined as $f^i(n) = g^i(n) + h^i(n)$, is used to order the low-level $OPEN$ for each agent a_i. Here, $g^i(n)$ is the distance from node n to the starting point $s_i^{(0)}$, and $h^i(n)$ is the estimated cost from node n to agent

Algorithm 1: ASB-ECBS(W, A)

Input: W: sub-optimal bound, A: set of agents
Output: high-level node, or no solution

1 $R.Constraints \leftarrow \phi$
2 $R.Solution \leftarrow low\_level(W)$ $R.Cost \leftarrow solution\_cost()$
3 **if** $method == SASB - ECBS$ **then**
4 $\quad A \leftarrow weight\_assignment(W, R, A)$
5 **end**
6 $OPEN \leftarrow append(OPEN, R)$ **while** $OPEN \neq \emptyset$ **do**
7 $\quad FOCAL \leftarrow append(FOCAL, N)$; $N \in OPEN$ & $N.cost \leq W.LB$
8 $\quad N \leftarrow$ node with minimum conflict in FOCAL
9 $\quad$ **if** N *has no* $conflict$ **then**
10 $\quad\quad$ **return** N
11 $\quad$ **end**
12 $\quad$ **if** $method == DSOLN$ **then**
13 $\quad\quad A \leftarrow weight\_assignment(W, N, A)$
14 $\quad$ **end**
15 $\quad C \leftarrow get\_conflict(N)$
16 $\quad OPEN \leftarrow make\_child\_node(N, C, OPEN)$
17 **end**
18 **return** *No solution*

a_i's goal location g_i. The high-level $FOCAL$ contains all nodes N where $N.Cost \leq W \times LB$, where LB is the lowest $f(N)$ among all nodes in the high-level $OPEN$ [Algo. 1: line 9] [8]. In contrast to the high-level, the low-level $FOCAL$ is guided by the low-level LB, which is the minimum $f^i_{min}(n)$ for agent a_i. Like the high-level search, the low-level $FOCAL$ operates using the low-level $OPEN$ list [8].

In each iteration [Algo. 1: lines 8-19], ASB-ECBS selects the node N from the $FOCAL$ list that has the fewest conflicts. If there is a tie, the node with the lowest solution cost is chosen. If N is a goal node (i.e., conflict-free), it is returned as a sub-optimal solution [Algo. 1: lines 11-13]. Otherwise, if a conflict occurs at the tentative solution of the high-level node N [Algo. 1: line 17], the conflict is resolved by creating two new child nodes, N_1 and N_2, using the $make\_child\_node$ method. These child nodes inherit all the constraints of the parent node N, with an additional constraint added to avoid the current conflict.

A low-level focal search is then applied for each agent involved in the conflict to generate sub-optimal paths. This search not only provides the solution path but also returns the minimum $f^i(n)$ from the low-level $OPEN$ list, $f^i_{min}(n)$, representing a lower bound on the agent a_i's shortest path. The updated $OPEN$ list is then returned from the $make\_child\_node$ method [Algo. 1: line 18]. The process repeats until a valid solution is found or the $OPEN$ list becomes empty, indicating that no solution exists for the problem [Algo. 1: line 20].

Algorithm 2: weight_assignment(W, N, A)

Input: W: sub-optimal bound, N: high-level node, A: set of agents
Output: A with updated sub-optimal bound

1 $count \leftarrow \max_{a \in A} conflict\_count(a, N)$
2 $offset \leftarrow \frac{(W-1)}{count}$
3 **for** $a_i \in A$ **do**
4 $\quad n = (a_i).conf$
5 $\quad (a_i).w \leftarrow 1 + (n * offset)$
6 **end**
7 **return** A

The $weight\_assignment$ function [Algo. 2] is invoked to assign appropriate weights to agents based on the number of conflicts in their current paths [Algo. 1: lines 4-6] [8]. This function takes as inputs the sub-optimal bound W, a high-level node N (denoted as R for the root node), and the set of agents A. It then calculates the number of conflicts each agent has with other agents' paths. The highest number of conflicts [Algo. 2: line 1] is used to determine the weight required for each conflict, termed the $offset$ [Algo. 2: line 2]. Each agent a_i's conflict count $(a_i).conf$ is multiplied by this $offset$ [Algo. 2: lines 3-6], resulting in each agent receiving a unique sub-optimal bound proportional to the number of conflicts along its path.

The set of agents A with their updated weights is then returned to line 5 of Algo. 1. After this, the root node R is pushed into the $OPEN$ list. ASB-ECBS proceeds with the high-level search following the ECBS process [Algo. 1: lines 8-20]. The node at the top of the $FOCAL$ list, with the fewest conflicts, is selected for expansion [Algo. 1: line 10]. If this node is a goal node (i.e., its paths are conflict-free), it is returned as the solution [Algo. 1: lines 11-13]. Otherwise, to resolve the conflict, two child nodes are created [Algo. 1: lines 17-18] with added constraints for the conflicting agents. The high-level iteration continues until a valid solution is found or the $OPEN$ list becomes empty, signifying that no solution exists for the problem [Algo. 1: line 20].

2.2.5 Limitations of ASB-ECBS

ASB-ECBS, like other heuristic-based algorithms, sacrifices optimality for computing speed, particularly under time restrictions. While the *anytime* aspect of the algorithm allows for ongoing improvement of the solution over time, the initial solutions may be far from ideal, especially in severely limited contexts. Previous approaches, such as Enhanced Conflict-Based Search (ECBS), have a similar drawback in that poor answers are presented early on, and optimality is only ensured with more processing time. However, in real-time applications requiring quick choices, this can constitute a bottleneck. ASB-ECBS functions in a centralized way, which means that a single authority is necessary to compute paths for all agents. This can be a constraint in large-scale, decentralized systems when agents have limited information and communication capabilities. While centralized methods like CBS and ECBS have the advantage of producing globally optimal or near-optimal solutions, they lack the flexibility of decentralized approaches like MAPF-DC (Multi-Agent Path Finding with Dynamic Conflict Resolution), which allows agents to operate independently in certain scenarios. ASB-ECBS also inherits this problem.

Although ASB-ECBS enhances the scalability and efficiency of MAPF solvers, it does not come without drawbacks. The added processing complexity caused by dynamically modifying sub-optimal boundaries is one of the main obstacles. Recalculating these constraints repeatedly introduces overhead and reduces the efficiency improvements provided by the approach in scenarios where changes are frequent or unpredictable. Furthermore, precise predictions of needs specific to each agent are critical to ASB-ECBS. Incorrect predictions could cause the system to assign agents the wrong bounds, which would result in less-than-ideal performance or a higher likelihood of conflict [8].

Furthermore, the system becomes more complex when handling different boundaries for several agents, especially when scaling up to a large number of agents. It is still very difficult to strike a balance between computing speed and solution quality, particularly in real-time applications where both are crucial [8]. However, to address the shortcomings and make the workflow more efficient, we intend to use a Machine Learning Model with the assigned bounds.

2.3 ML Based Approached in MAPF

Recent works have explored the integration of Machine Learning (ML) into Multi-Agent Path Finding (MAPF), but none have specifically addressed heuristics or LaCAM. For instance, [16] proposed a priority-based MAPF algorithm using Prioritized Planning (PP), where an ML framework was developed to learn a ranking function that assigns priorities to agents. However, this approach is highly agent-dependent. The model computes an h value for each agent, and priorities are set based on this value. Unfortunately, when dealing with a large number of agents, the model needs to run for each agent to generate the h values, introducing significant inference overhead. This added computation slows down the overall search process, conflicting with the goal of sub-optimal algorithms, which is to minimize runtime.

Other approaches have incorporated ML into Conflict-Based Search (CBS) algorithms for MAPF [17]-[18]. These works focused on enhancing node selection [17] and conflict resolution strategies [18], though their goals and methods differ from the approach discussed here. Additionally, another ML approach has been used in the context of the anytime MAPF solver known as Large Neighborhood Search (LNS) [19]. This solver initially generates a solution and then iteratively improves it by selecting a subset of agents, termed a neighborhood, to refine the solution quality if time allows. In this case, ML is used to learn a function that helps select neighborhood agents more effectively to improve solution quality.

2.3.1 Linear Regression in MAPF

With the increasing popularity of machine learning, data-driven methods like linear regression have been incorporated into several study areas, including MAPF, to improve conventional techniques. A basic statistical technique for modeling the connection between one or more independent variables and a dependent variable is called **Linear Regression**. To model the connection, a linear equation is fitted to the observed data, enabling predictions to be made depending on the input features. For comprehending and forecasting behavior in complex systems, linear regression is a helpful technique due to its ease of use and interpretability [20]. In its most basic version, a single independent variable is used in linear regression that aims to predict a single output from this input. The formula for this relationship is:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.3}$$

Where:

- y indicates predetermined output (It is a dependent variable),
- x indicates the feature of input (It is an independent variable),
- β_0 is the interruption of the regression line,
- β_1 is the coefficients (or slope) for the independent variable x, representing its effect on y,
- ϵ is the error term accounting for unexplained variability.

When dealing with challenges such as MAPF, Linear Regression is a good fit since it can approximate a linear relationship between the desired result (sub-optimal boundaries) and agent attributes (such as conflict metrics and distance measures). Real-time systems like MAPF, where predictions must be made quickly and practically, are ideally suited for linear regression due to their ease of interpretation and simplicity [4]. The advantages of linear regression for MAPF include:

- **Simplicity**: The assumption of linearity simplifies the interpretation of the attributes and outcome correlations.

- **Efficiency**: Since linear regression models are cheap to train and use computationally, they are a good choice for large-scale systems that require real-time prediction.
- **Predictive Power**: Despite being rudimentary, linear regression works well to identify overarching patterns in the data and offers a preliminary foundation for enhancing the efficiency of CBS-based MAPF solutions.

Thus, the system may dynamically forecast agent-specific sub-optimal boundaries by integrating linear regression into the CBS algorithm, freeing the algorithm to concentrate on more effective path-finding techniques. By utilizing previous data from MAPF simulations, the Regression model learns the correlations between characteristics and the desired boundaries which reduces computing overhead dramatically [21].

Chapter 3

Methodology

As we already know, Linear Regression is the statistical method of modeling the relationship between one or more independent variables (e.g. features) and one dependent variable (e.g. goal). This statistical method is used to model the extent of the relationship between one or more predictor variables and a criterion variable. The aim is to find the linear relationship that best matches the data. In the context of MAPF, linear regression can be used to model the relationship between particular characteristics or qualities and the decisions or behaviors of agents.

This involves coordinating the activities of multiple agents to minimize an objective function, like overall journey duration, and prevent collisions. In a multi-agent system, while aiming to predict or model the action of agents a connection between linear regression and MAPF may occur. For instance, one may use linear regression to analyze previous data on agent actions and environmental variables. Using the correlations that were found through regression analysis, the model can be used to assist agents in choosing their paths of action in a MAPF scenario.

The primary goal of this project is to improve Conflict-Based Search (CBS) by integrating machine learning, hence increasing the computing efficiency and scalability of MAPF. To be more precise, we use linear regression models to forecast agent-specific sub-optimal boundaries, which frees the CBS algorithm from the need to reach full optimality and enables it to opt for more promising alternatives. The system's scalability and performance in dynamic environments—where things might change quickly—are greatly enhanced by this adaptation [11].

In this study, we present a system that combines the predictive power of linear regression with the adaptability of CBS. Predicted suboptimal bounds are employed to optimize the search space for every agent, guaranteeing effective resolution of conflicts while preserving the overall quality of the solution. This reduces processing overhead by allowing the algorithm to adapt more successfully in complex MAPF environments.

3.1 Algorithm Description

Algorithm 3: ML_weight_assignment(W, N, A)

Input: W: Global sub-optimal weight, N: Node from high-level, A: Agents set
Output: agents set A' with updated sub-optimal bound

1 $\Phi_I \leftarrow$ extract_feature(N)
2 **for** $a_i \in A$ **do**
3 $\quad (a_i).w \leftarrow f(\Phi_I, a_i)$
4 $\quad (a_i).w \leftarrow Normalise((a_i).w, W)$
5 **end**
6 **return** A'

The Algorithm 3 inputs include the global sub-optimal bound W, the current high-level node N, and a set of agents A. It gives an updated set of agents A, each of which is assigned certain weights relevant to the present conditions of the environment for which the service is being constructed. The algorithm takes as input the global sub-optimal bound W, the current high-level node N, and the set of agents A. It returns an updated set of agents A', each assigned with specific weights that reflect their importance in the current context of the underlying environment.

Initially, the algorithm begins by examining the information contained within the node N. This node encapsulates vital environmental data, including agent characteristics, contextual factors, and any relevant features that might influence the agents' decision-making processes. In Line 1, the algorithm focuses on extracting the necessary features from N that will be utilized by the machine learning model. This feature extraction is critical, as it sets the foundation for the subsequent weight assignment process. The features identified here may encompass various metrics or indicators that provide insight into the agents' performance or relevance within the given environment.

Following the feature extraction, the algorithm enters a loop that iterates over each agent in the set A [Lines 2-5]. This iterative approach ensures that each agent is considered independently, allowing for tailored weight assignments based on specific attributes and environmental features. Within this loop, for each agent a_i, the algorithm assigns a weight $(a_i).w$ based on the agent's unique information and the previously extracted features from N [Line 3]. This weight assignment process is designed to reflect the agent's current capability, role, or relevance, effectively prioritizing agents that exhibit characteristics deemed valuable for achieving the desired outcomes.

To ensure that the assigned weights are meaningful and manageable, the algorithm includes a normalization step in Line 4. Here, the weights are adjusted to remain within the defined range of the global sub-optimal bound [1-W]. Normalization is a critical aspect of the weight assignment process, as it guarantees that the weights do not exceed the global sub-optimal, thereby maintaining consistency and stability within the multi-agent system. This step helps prevent any single agent from disproportionately influencing the overall decision-making process, which could lead to suboptimal outcomes.

After the algorithm has iteratively assigned weights to all agents, it compiles the updated set of agents, now denoted as A' [Line 6]. This output represents the final state of the agents with their respective weights, which can then be utilized in further stages of the decision-making framework. The updated set $A'$is crucial for subsequent processes, such as task allocation, resource distribution, or any other operation where agent performance is a key factor.

Algorithm 3 shows the pseudocode of weight assignment of our proposed approach. The process takes global sub-optimal bound W, current high-level node N, and the set of agents A as input and returns the agents set A' with specified weights for each agent as output. Node N contains the necessary information about the environment. Based on the information, the process tries to

extract the necessary features to infer using the machine learning model [Line 1]. Then for each individual agent, the sub-optimal bound is assigned iteratively [Lines 2-5]. In each iteration, for a particular agent a_i, based on the agent information and extracted features from N, weight $(a_i).w$ are assigned [Line 3]. Then the weight is normalized so that the weight remains within the range of the global sub-optimal bound [1-W] [Line 4]. After assigning weights for each agent, the updated agents set A' as output [Line 6].

3.2 ML Approach

In this section, we present the overall framework of the Machine learning (ML) approach that helps the algorithm to learn a function that would effectively predict the sub-optimal bound for the agents. The main steps of the entire process include:

- **Data Collection:** To collect the necessary data for model training, two types of MAPF instances are generated by running the ASB-ECBS algorithm on several MAPF instances consisting of different start and goal locations. They are train instances $\mathcal{I}_{train}$ and test instances $\mathcal{I}_{test}$. Each instance is associated with a target label $[y$ providing the value of sub-optimal bound w. For both the instances, we obtain the instance dataset $D_{\mathcal{I}}^{train}$ and $D_{\mathcal{I}}^{test}$.
- **Model Learning:** The training dataset $D_{\mathcal{I}}^{train}$ was used to teach the model a function that predicts the appropriate value for w such that the predictions are as "similar" to the labels in train dataset as possible.
- **Inferring during the Search:** We use the learned function to infer the value of sub-optimal bound w during the search process.

3.2.1 Data Collection Process

To effectively train models for detecting sub-optimal boundaries, obtaining appropriate data is a crucial first step in machine learning-based Multi-Agent Path Finding (MAPF). Creating an eclectic array of MAPF instances with different agent counts, start-goal locations, and environmental complexity is part of the data-collecting process. These instances are further split into training and testing datasets, which are used to assess how effectively the suggested model performs. To generate a rich feature vector that effectively depicts the dynamics of the problem, agent-specific features are retrieved for each instance, including start-goal distances, conflict metrics, and environmental elements. Furthermore, conflict resolution constraints are used to assign target labels, which indicate ideal sub-boundaries and are crucial for optimizing the machine learning model's predictive power. These features are essential for encapsulating the complexities of pathfinding problems and agent interactions. Every occurrence is also given a target label, which stands for the ideal sub-boundaries that are employed in resolving conflicts Through the collection of this data, we seek to train a machine learning model capable of dynamically predicting sub-optimal bounds, thereby increasing the MAPF algorithm's scalability and efficiency by lowering the computational load while preserving solution quality.

Creating the training and testing datasets, $D_{\mathcal{I}}^{train}$ and $D_{\mathcal{I}}^{test}$, is the first stage in the machine learning pipeline. We gather two sets of MAPF instances on the same map, with varying start and goal locations and varying numbers of agents: train instances $\mathcal{I}_{train}$ and test instances $\mathcal{I}_{test}$. We create an example $X_{\mathcal{I}}$ for each MAPF instance $I \in \mathcal{I}_{train}$. This example consists of (i) a $p - dimensional$ feature vector that uses p features to characterize the present environment, and (ii) a target label y giving the value of w.

Feature Vector

For every case $\mathcal{I} \in D_{\mathcal{I}}$, a $p - dimensional$ feature vector Φ_I is defined. Table 3.1 summarizes the $p = 21$ features that we employed in our implementation. A study on feature importance is offered in the results section. We have carefully picked the features to guarantee that their extraction throughout the search process does not entail unnecessary runtime overhead. They can, however, also accurately represent the present states, which helps the model correctly comprehend the underlying environment. To effectively gather data about the underlying MAPF environment, several of these traits are frequently employed in the literature [16]–[19]. 4 main categories can be used to classify the features:

- Start-Goal Distances and Nearest Neighbor: This category includes features related to the distance between an agent's start and goal locations, as well as the distance to its nearest neighbor. These distances are crucial in determining the potential for conflicts and the complexity of the path an agent needs to take. For example, the distance from an agent's start location ($\mathbf{S_i}$) to its goal location ($\mathbf{G_i}$) is a primary feature, along with the distance to the nearest neighboring agent.

- Conflict Metrics: Features in this category capture the various types and counts of conflicts that an agent might encounter during pathfinding. This includes the total conflict count for an agent, the number of vertex, edge, and cardinal conflicts, and the number of agents that have at least one conflict with each other. These features are critical for understanding how conflicts influence the sub-optimal bounds of each agent.

- Cost-Related Features: These features focus on the costs associated with an agent's path. They include ratios and differences such as $\mathbf{A_i}$ Cost/LB (the cost of agent $\mathbf{A_i}$ relative to the lower bound), $\mathbf{A_i}$ Cost - LB (the difference between an agent's cost and the lower bound), and $\mathbf{A_i}$ Cost - S (the difference between an agent's cost and the sum of the cost of the individually cost-minimal paths of all agents). These metrics help quantify the efficiency of an agent's path relative to theoretical minimal paths.

- Environmental and Systemic Metrics: This category includes broader environmental features, such as the obstacle percentage relative to total nodes and the agent percentage relative to total nodes. These features help contextualize the agent's pathfinding challenge within the overall environment, providing insights into how dense or sparse the environment is.

In summary, this study explored a range of unbounded and bounded sub-optimal solvers for the MAPF problem based on the CBS algorithm. Among bounded solvers, ECBS emerged as the top performer, while PPS proved to be the fastest unbounded solver but often delivered higher-cost solutions. Additionally, GCBS occasionally provided nearly optimal solutions. From our study ASB-ECBS, an ideal approach to tackle MAPF with two versions, SASB-ECBS and DASB-ECBS, modified to agent-specific conflict resolution needs and aimed at improving runtime performance [8]. Our plan is to implement a Machine Learning model for predicting agent-specific bounds. We anticipate that this will lead to a substantial improvement in runtime while simultaneously enhancing the quality of the solutions. In this section, we describe our supervised learning method for determining weights to solve the Multi-Agent Path Finding (MAPF) problem. Our approach involves implementing a machine-learning model designed to address MAPF. The main idea is to train the machine-learning model with hand-drawn features and labels that generate unique weights for each agent. The weights are then utilized for predicting sub-optimal bounds, with the overall objective being to increase the quality of solutions while simultaneously enhancing runtime efficiency. The training process involves employing handcrafted features detailed in Table 3.1.

Feature Description	Count
% of obstacles compared to total no. of nodes	f_1
% of agents compared to total no. of nodes	f_2
% of agent at goal and non-goal position	$f_3 - f_4$
Normalised conflict count of agent a_i	f_5
Normalised distance from start S_i and Goal G_i locations of agent a_i	$f_6 - f_7$
Distance of nearest and furthest neighbor's of agent a_i	$f_8 - f_9$
Start Location s_i of agent a_i, their Min, Max & Median	$f_{10} - f_{12}$
Goal Location s_i of agent a_i, their Min, Max & Median	$f_{13} - f_{15}$
% of agents that have at least one conflict with each other	f_{16}
The SIC of the individually cost-minimal path of agent a_i	f_{17}
Features Related to a_i Cost: Cost/LB, Cost - LB, Cost - S, Cost/S	$f_{18} - f_{21}$

Table 3.1: *Features for ML model. Column "Count" reports the number of features contributed by the corresponding entries.*

Normalization
To ensure that the data fed into our model is consistent and comparable, we normalized all feature values using a Min-Max normalization method. This process scaled the values of each feature to fall within a range of 0 to 1. This step is essential for preventing any single feature from disproportionately influencing the model's predictions, which allows for a more balanced and accurate training process. The normalized data is then split into training and test sets, with the training set used to learn the relationships between features and sub-optimal bounds, and the test set used to evaluate the model's performance.

Labels
A ground truth label y_I provides the values of appropriate weight, w. The model should generate suitable weights for each agent. Based on that, the sub-optimal bound of each agent is calculated.

Model Learning
We learn a linear function with parameters $\omega \in \mathcal{R}^p$

$$f : \mathcal{R}^p \rightarrow \mathcal{R} : f\left(\Phi_I\right) = w^T \Phi \tag{3.1}$$

that minimizes the loss function

$$L(\boldsymbol{w}) = \sum_{I \in \mathcal{I}_{\text{Train}}} mse\left(y_I, \hat{y}_I\right) + \frac{C}{2}\|\boldsymbol{w}\|_2^2 \tag{3.2}$$

Here, y_I represents the ground truth and $\hat{y}_I$ represents the predicted scores. MSE is the loss function that measures the Mean Square Error between the ground truth and the predicted score. Finally, $C > 0$ is the regularization parameter.

3.2.2 ML Model architecture and hyper-parameters

The use of deep neural networks in the majority of existing models for the machine learning-based approach to linear regression is well-documented in the literature [22]. Large models, however, may place a significant computational load on the search process, which goes against our primary goal of runtime reduction. Prohibitive delays could result from a deeper neural network model, which would add significant overhead at every high-level layer. We tried various designs while keeping the model size smaller, keeping this in mind. In the end, we decided on a network setup with two hidden layers:

- **Input Layer:** With 21 neurons representing the feature vector Φ_I.
- **Hidden Layers:** with 64 neurons and 32 neurons respectively each having an activation function as sigmoid, $\sigma(x) = \frac{1}{1+e^{-x}}$.
- **Output Layer:** representing the predicted values of α, β and γ as $y_I = ([\alpha, \beta, \gamma])$.

To improve the Model, we employ Adaptive Moment Estimation (ADAM), a well-liked optimization method. During training, ADAM dynamically modifies the learning rate for every parameter; the starting learning rate was 0.01. The Mean Squared Error (MSE) is used as the loss function to determine the difference between the projected score and the actual score. The datasets were divided into train, validation, and test sets, with 80%, 10%, and 10% each.

Guided Search Apporach

We replace the conventional weight assignment approach learned function after gathering data offline and studying the function $f(\cdot)$. This function uses the attributes of the current instance that it has obtained to forecast the values of sub-optimal bound w. Agents are then assigned with these sub-optimal bounds so that the process can guide the agents for quick convergence to the goal locations.

Chapter 4

Empirical Analysis

We conducted an empirical assessment of the performance of ECBS-ML by contrasting it with the most advanced MAPF algorithms. We exclusively took into account the most advanced bounded sub-optimal solvers with ECBS-ML, namely ASB-ECBS, ECBS, and BCBS (W_H, W_L). This was done to assure fairness. The four standard maps are shown below and these are the ones that have been tried in this work and are commonly used to evaluate MAPF algorithms.

- **4 grids connection** consist of two maps: one that is 32×32 and has obstacles of 20%, and the other is 8×8 with obstacles of 12% [23], [24].
- **Map similar to Kiva** (22×53) another type of map among Kiva systems is what can be considered as a warehouse map. Other light and mobile self-operative robots choose myriad shelves (the starting point) and then throw them to some certain coordinates (the final point) in this map [10].

  Roundabout environment is an object (24×48) is a map that has four rooms connected by a circle.

For each map and the set of given agents, In our experiments, we have designed 100 separate scenarios for each map and set of agents, with randomly generated starting and target locations. This implies that we have used every solver to perform these experiment cases. We have limited the run-time of the algorithm to 180 s as the sub-optimal algorithms may take more time based on the maps or the number of agents or the weights. Consequently, if a solver returns failure or does not return any solution within the time limit, then the execution is incomplete. In compliance with earlier research [9], [25], [26], we have taken into account of two metrics of the experiment cases that were successfully resolved for assessment purposes are as follows:

- The average amount of time of execution per agent.
- Each agent's average number of nodes (low-level search space) generated on the low-level search produced on the low-level search

ASB-ECBS, ECBS, and BCBS are implemented using the code provided by their respective authors; whereas. BCBS-ML was implemented in C++ even though the model was initially trained in Python. We employed libtorch to trace the model in C++ for inference purposes. The experiments were conducted on a Ryzen 5950x 3.4 GHz CPU with 64 GB RAM.

To analyze the performance of our approach, we have divided the experiment process into two parts. In the first part, we observe the impact of weights on different maps while keeping the number of agents fixed. The experiments have been done in both sparse and simple graphs (8X8 and 32x32) as well as dense and complex graphs (warehouse and roundabout). Then we experimented to observe the impact of a number of agents while keeping the sub-optimal bound fixed for the agents. Same as before, we have experimented both on sparse and simple graphs (8X8 and

32x32) as well as dense and complex graphs (warehouse and roundabout). For simplicity, we have named our approach as BCBS-ML (Bounded Conflict Based Search with Machine Learning) to effectively make comparisons with other algorithms in the experimental graphs and figures. The results and analysis of the experiments are discussed in the subsequent sections.

4.1 Impact of Weight

4.1.1 Sparse and Simple Graph

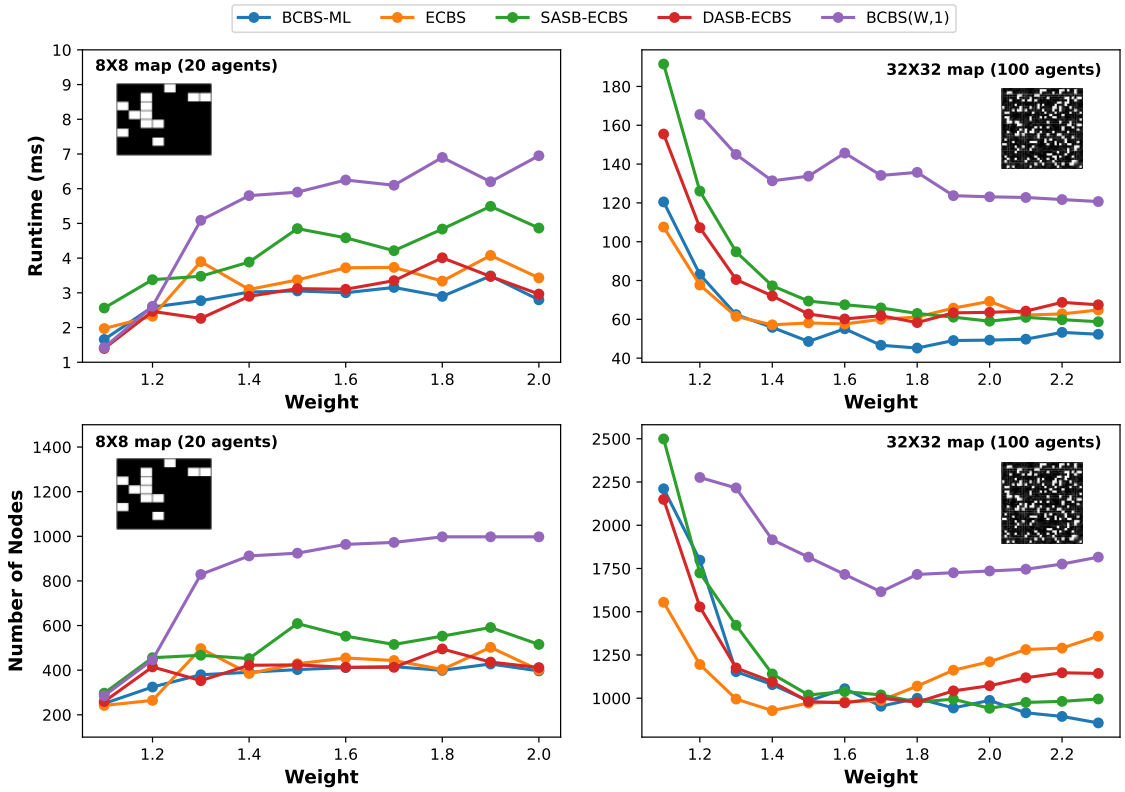


Figure 4.1: *Impact of distinct weights on MAPF agent performance in sparse and simple graphs. Average time and nodes generated in low-level search shown. Legend indicates algorithm by color and marker. Black cells represent obstacles.*

Figure 4.1 evaluates the effects of various sub-optimal bound (weight) values on the number of nodes generated and runtime during the low-level search for the following five MAPF algorithms: BCBS-ML, ECBS, SASB-ECBS, DASB-ECBS, and BCBS(W,1). Two grid sizes were used in the experiments: an 8x8 map with 20 agents and a 32x32 map with 100 agents. The x-axis represents the weight, while the y-axis represents both runtime (in milliseconds) and the number of nodes generated.

Figure 4.1 demonstrates that, for the majority of weights, BCBS-ML outperforms competing algorithms in terms of runtime and search space on all benchmark maps. Across all weight levels, BCBS-ML consistently displays the shortest run-time on the 8x8 map, demonstrating its effectiveness in simpler settings. ECBS and DASB-ECBS perform competitively, but as weight values rise, they are unable to surpass BCBS-ML. The highest runtimes and node generation are shown in BCBS(W,1), demonstrating its inefficiency. Although SASB-ECBS and DASB-ECBS perform similarly, BCBS-ML maintains its lead in runtime at lower weights on the 32x32 map. However, the advantage of BCBS-ML continues at larger weights, especially above 1.5, making it the most

effective algorithm overall, even for more complex searches.

The 8x8 map demonstrates BCBS-ML's superior performance while indicating its capacity to explore simpler paths effectively. Although DASB-ECBS and SASB-ECBS's adaptive mechanisms function better as weight increases, they are still less effective than BCBS-ML. The consistently poor performance of BCBS(W,1) highlights how static-weight algorithms are unable to handle growing complexity.

4.1.2 Dense and Complex Graph

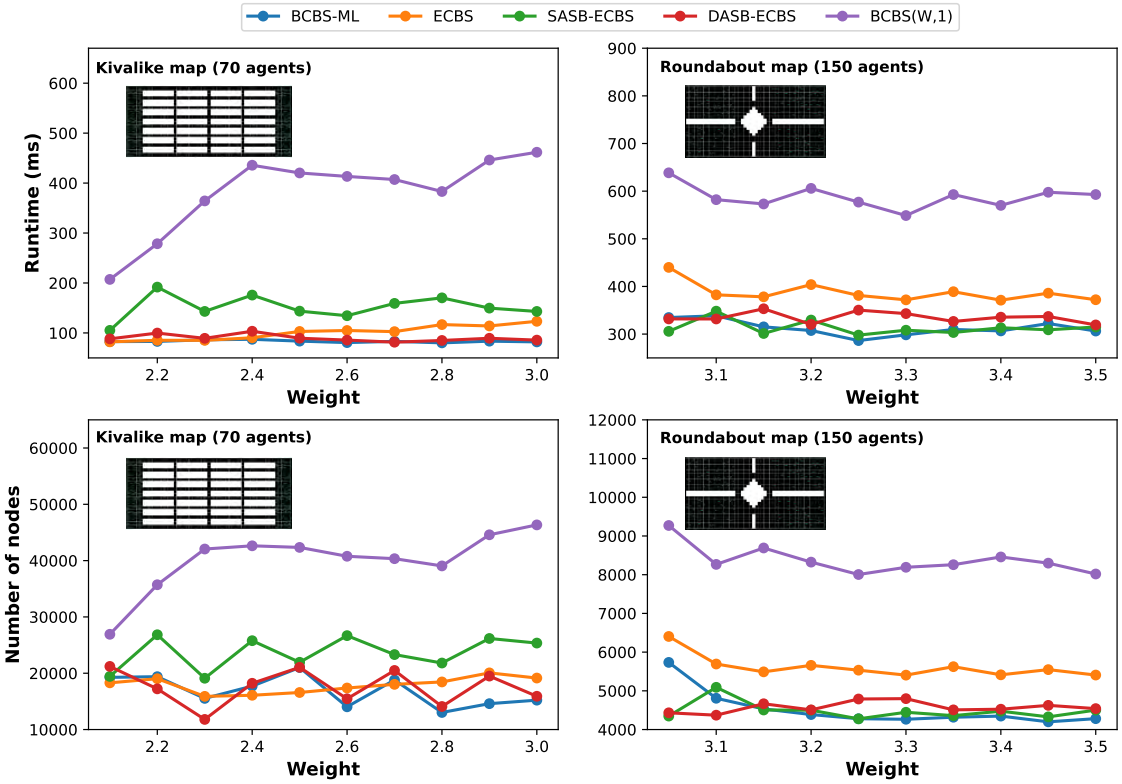


Figure 4.2: *Impact of distinct weights on MAPF agent performance in dense and complex graphs. Average time and nodes generated in low-level search shown. Legend indicates algorithm by color and marker. Black cells represent obstacles.*

In Figure 4.2, a Kiva-like map with 70 agents and a Roundabout map with 150 agents are used to assess how weight affects runtime and node formation in a denser environment. Due to the increasing complexity of the environment, the weight range of the Roundabout map has been extended to 3.5. Similar to sparse and basic graphs, the y-axis shows the runtime (in milliseconds) and the number of nodes generated during the low-level search, while the x-axis shows the weight values.

As the weight increases from 2.2 to 3.0, the runtime of all algorithms in the Kiva-like map increases, with BCBS-ML and DASB-ECBS consistently outperforming the others. DASB-ECBS performs better at greater weights in the Kiva-like map, yet it fails to outperform BCBS-ML, which continuously maintains competitive runtimes. At higher weights, ECBS and SASB-ECBS work satisfactorily, but their effectiveness diminishes compared to that of BCBS-ML. Although BCBS-ML still leads in runtime in the roundabout map, DASB-ECBS becomes a close competitor,

especially as weight values rise above 3.0, demonstrating its capacity to manage greater complexity effectively. Once more, BCBS(W,1) exhibits the highest node generation and runtimes.

In dense environments, DASB-ECBS's adaptability is essential for efficiently handling conflicts, leading to shorter runtimes and node formation. At lower weights, however, BCBS-ML's performance indicates that it can well handle complexity and continuously maintain its position as a top performer. In these situations, the ability of BCBS-ML and DASB-ECBS to dynamically adjust sub-optimal weights to accommodate the particular needs of each agent is a definite advantage. As the weight increases, BCBS-ML and DASB-ECBS limit unnecessary exploration in the low-level search by making sure that agents needing fewer conflicts are assigned lower weights. The necessity for more adaptable approaches is shown by the constantly increased runtimes and node generation for BCBS(W,1).

4.2 Impact of No. of Agents

4.2.1 Sparse and Simple Graph

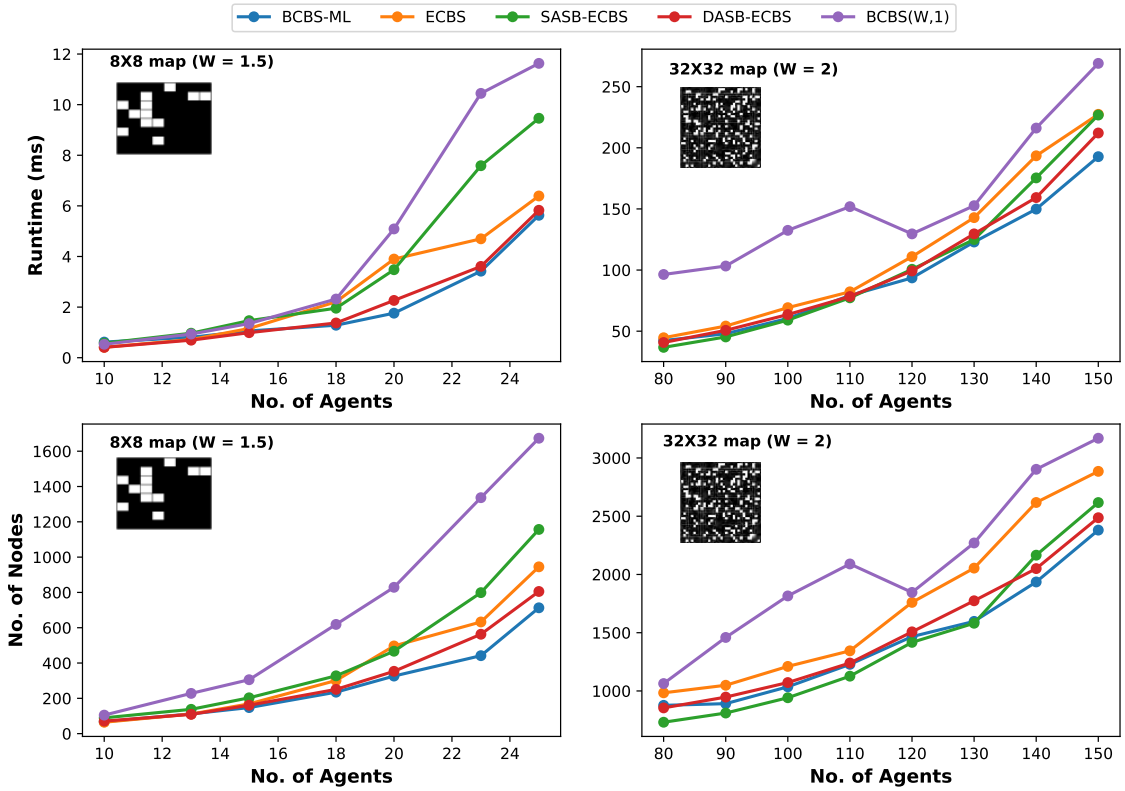


Figure 4.3: *Impact of number of agents on MAPF agent performance in sparse and simple graphs. Average time and nodes generated in low-level search shown. Legend indicates algorithm by color and marker. Black cells represent obstacles.*

In Figure 4.3, we investigate the effects of multiplying the number of agents on time and node creation regarding the five MAPF algorithms. The experiments were performed on two maps: the 8x8 map with a weight of 1.5 and the 32/32 map with a weight of 2. In the 8x8 map, there were 10-24 agents; in the 32x32 map, there were 80-150 agents. Here, the y-axis represents the count of the formed nodes and also the time (in milliseconds), and the x-axis illustrates the number of agents.

The runtime and node generation of all algorithms increases with the number of agents in the 8x8

map. Despite increasing the number of agents (e.g., 24 agents), BCBS-ML maintains the lowest runtime, proving its efficiency. Both ASB-ECBS and ASB-ECBS are competitive up to about 20 agents, but after that, BCBS-ML starts to perform noticeably better than both, especially when it comes to node generation. Across the range of agent counts, BCBS(W,1) has the highest runtimes and node counts. When it comes to managing higher agent counts, BCBS-ML outperforms the other algorithms on the 32x32 map.

DASB-ECBS can effectively handle greater numbers of agents due to its dynamic weight adjustment, BCBS-ML's robustness is proven by its ability to outperform the others as the number of agents increases. In situations with several agents, the static-weight techniques of BCBS(W,1) result in longer runtimes and larger search areas, underscoring the need for adaptive algorithms.

4.2.2 Dense and Complex Graph

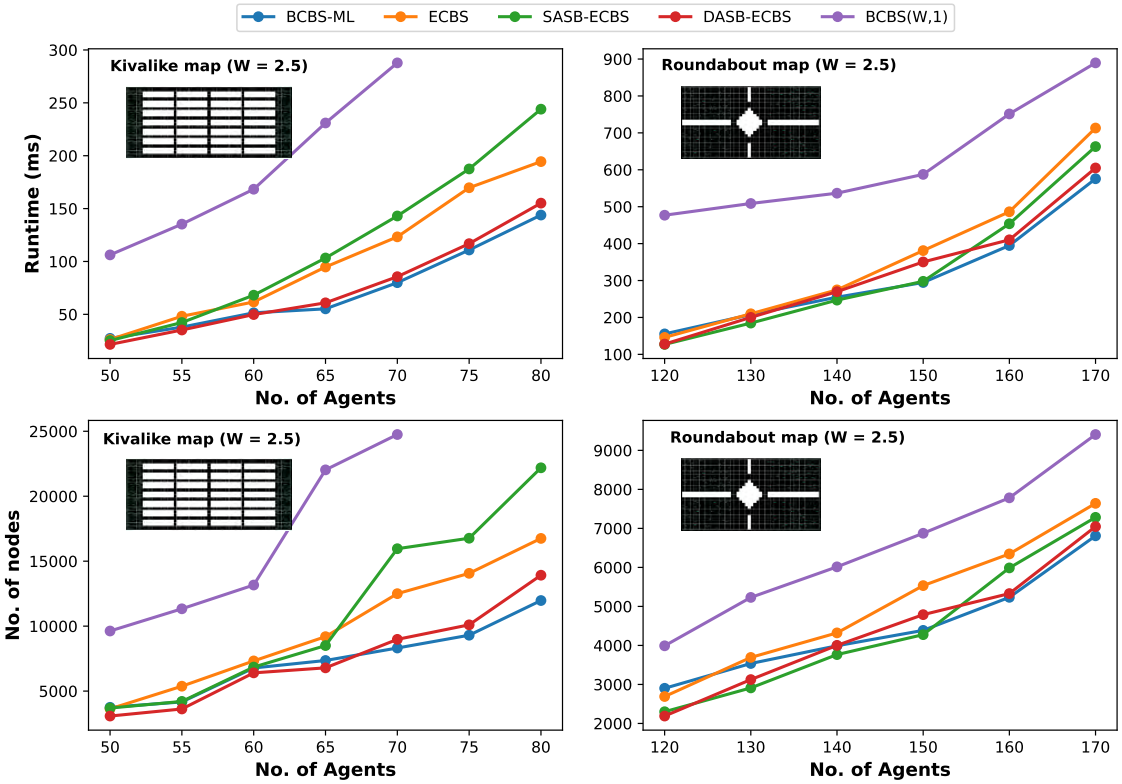


Figure 4.4: *Impact of number of agents on MAPF agent performance in dense and complex graphs. Average time and nodes generated in low-level search shown. Legend indicates algorithm by color and marker. Black cells represent obstacles.*

In Figure 4.4, a Kiva-like map (weight 2.5) and a Roundabout map (weight 2.5) are studied to examine the effects of different numbers of agents in dense environments. The x-axis displays the number of agents, while the y-axis displays both runtime (in milliseconds) and node generation.

In the Kiva-like map, DASB-ECBS continues to be among the most efficient, with the shortest runtimes and node generation at lower agent counts. However, after about 60 agents, BCBS-ML begins to perform better than DASB-ECBS and SASB-ECBS, demonstrating its ability to handle conflicts well, as the number of agents rises. While DASB-ECBS continues to outperform in the Roundabout map, BCBS-ML significantly improves as agent counts increase and emerges as the most effective choice at higher counts. On both maps, BCBS(W,1) has the highest runtimes and

node counts across the agent count range.

While the dynamic weight assignment of DASB-ECBS is essential for managing conflicts in congested environments with growing agent counts, BCBS-ML's performance demonstrates its ability to control agent pathways well without imposing a large overhead. BCBS(W,1)'s persistently poor performance highlights its shortcomings in adjusting to more complex high-density circumstances, highlighting the need for adaptive solutions.

To conclude, considering the findings from the research conducted on both the Sparse/Simple and Dense/Complex graphs, BCBS-ML emerges as the most effective algorithm overall in terms of runtime and node generation. Since BCBS-ML adaptively extracts features from the environment and assigns weights based on them, it is more suitable, which is why the results are better. On the other hand, this feature extraction and weight assignment requires some additional calculations, which add to the runtime. When the number of agents is small, the solution generation time is already less, so this extra calculation does not significantly increase the overall time compared to other algorithms. However, when the number of agents is large, the overall time increases, but this additional overhead does not have a significant effect on the overall runtime.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This study introduces an innovative approach to combine machine learning (ML) that scales the efficiency of Multi-Agent Path Finding (MAPF) solutions with the CBS framework. It successfully operates Linear Regression models to predict sub-optimal bounds of the agent-specifying type without relying on CBS algorithm trying for full optimality. This change enhances the performance of this system, which is necessary for large and dynamic environments that need to be able to adapt quickly based on changing situations.

The main message from this study, which we wish to deliver and have successfully delivered in the context of automated machine learning, is that an augmented MAPF framework could evade limitations observed among typical CBS-based methods with further benefits at a level both distinct (in performance) and generalizable. We scale down the costs using our newly built approach so that it does not influence solution quality to a great extent. This essentially means that bounds on agent-specific sub-optimality, uniformly applicable across different applications (e.g. robotics/autonomous vehicles/logistics), are guaranteed to be small as long we remain close to holding the strong equality at all times.

In conclusion, the combination of machine learning in general, more specifically Linear Regression models within a CBS framework provides an interesting perspective to tackle intricate MAPF problems. This approach can significantly improve computational efficiency while preserving the quality of final solutions, rendering it highly applicable to various real applications.

5.2 Future work

This work has successfully shown the promise of using machine learning along with the CBS framework to improve MAPF solutions. But there are several exciting avenues for research here that we have no doubt will make this approach faster and more generally useful in the future.

Using Advanced Machine Learning Models: Delve into the application of sophisticated models that utilize deeper layers to learn complex relationships between environmental features and sub-optimal thresholds (e.g., deep neural networks, ensemble methods) [9]. While this study was limited to linear regression due its computational convenience, more advanced models may provide better prediction performance.

Dynamic Environments: Extend the current framework to handle dynamic environments where obstacles or agent goals change in real-time. This would involve incorporating online learning

techniques to adapt the machine learning model to evolving conditions. Decentralization: Investigate the feasibility of decentralizing the proposed approach. This would allow agents to learn and adapt their sub-optimal bounds independently, potentially improving scalability and robustness in large multi-agent systems.

Real-World Applications: Evaluate the performance of the proposed approach in real-world scenarios, such as robotics or autonomous vehicles. This would involve addressing practical challenges such as sensor noise, communication limitations, and safety constraints.

Bibliography

[1] M. Veloso, J. Biswas, B. Coltin, *et al.*, "Cobots: Collaborative robots servicing multi-floor buildings," Oct. 2012, pp. 5446–5447, ISBN: 978-1-4673-1737-5. DOI: 10.1109/IROS.2012.6386300.

[2] J. Li, Z. Chen, D. Harabor, P. Stuckey, and S. Koenig, "Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 10 256–10 265, Jun. 2022. DOI: 10.1609/aaai.v36i9.21266.

[3] R. Stern, N. Sturtevant, A. Felner, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Twelfth Annual Symposium on Combinatorial Search*, vol. 10, no. 1, Jun. 2021. DOI: 10.48550/arXiv.1906.08291. [Online]. Available: https://doi.org/10.48550/arXiv.1906.08291.

[4] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, p. 9, 2008. DOI: 10.1609/aimag.v29i1.2082. [Online]. Available: https://doi.org/10.1609/aimag.v29i1.2082.

[5] H. Ma, J. Yang, L. Cohen, T. Kumar, and S. Koenig, "Feasibility study: Moving non-homogeneous teams in congested video game environments," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, Oct. 2017. DOI: 10.1609/aiide.v13i1.12919.

[6] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, Feb. 2015. DOI: 10.1016/j.artint.2014.11.006.

[7] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Springer Tracts in Advanced Robotics*, 2011, pp. 3–19. DOI: 10.1007/978-3-642-19457-3_1. [Online]. Available: https://doi.org/10.1007/978-3-642-19457-3_1.

[8] M. Rahman, M. A. Alam, M. M. Islam, I. Rahman, M. M. Khan, and T. Iqbal, "An adaptive agent-specific sub-optimal bounding approach for multi-agent path finding," *IEEE Access*, vol. 10, pp. 22 226–22 237, 2022. DOI: 10.1109/access.2022.3151092.

[9] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[10] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Seventh Annual Symposium on Combinatorial Search*, Citeseer, 2014.

[11] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Adaptive Agents and Multi-Agent Systems*, 2018.

[12] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," 1, vol. 5, Jan. 2014, pp. 961–962. DOI: 10.3233/978-1-61499-419-0-961.

[13] Q. Sajid, R. Luna, and K. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," *Proceedings of the 5th Annual Symposium on Combinatorial Search, SoCS 2012*, vol. 3, pp. 88–96, Jan. 2012. DOI: 10.1609/socs.v3i1.18243.

[14] K. Okumura, M. Machida, M. Kamezaki, and T. Amano, "Priority inheritance with backtracking for iterative multi-agent path finding," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6797–6803. DOI: 10.1109/ICRA.2019.8794359.

[15] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Transaction PAMI*, no. 4, pp. 392–399, 1982.

[16] S. Zhang, J. Li, T. Huang, S. Koenig, and B. Dilkina, "Learning a priority ordering for prioritized planning in multi-agent path finding," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, 2022, pp. 208–216.

[17] T. Huang, B. Dilkina, and S. Koenig, "Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding," in *International joint conference on autonomous agents and multiagent systems (AAMAS)*, 2021.

[18] T. Huang, S. Koenig, and B. Dilkina, "Learning to resolve conflicts for multi-agent path finding with conflict-based search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 11 246–11 253.

[19] T. Huang, J. Li, S. Koenig, and B. Dilkina, "Anytime multi-agent path finding via machine learning-guided large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 9368–9376.

[20] T. Gabel and M. Riedmiller, "Learning a multi-agent system in dynamic environments: Empirical investigation on the pursuit game," *Autonomous Agents and Multi-Agent Systems*, vol. 10, no. 2, pp. 159–192, 2005.

[21] T. Huang, B. Dilkina, and S. Koenig, "Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2021.

[22] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 140–147, 2020.

[23] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, 2022.

[24] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[25] L. Cohen, T. Uras, T. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Bounded suboptimal multi-agent path finding using highways," in *IJCAI*, 2016, pp. 3067–3074.

[26] L. Cohen and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding.," in *IJCAI-6*, 2016, pp. 3978–3979.