Multi-Agent Algorithms With Assignment Strategy Pursuing Multiple Moving Targets in Dynamic Environments

Azizkhon Afzalov

A thesis submitted in partial fulfilment of the requirements of Nottingham Trent University for the degree of

Doctor of Philosophy

November 2022

This thesis is dedicated to
my late mother, beloved wife, and my family with great gratitude.
I know you will be proud of this milestone accomplished.

Acknowledgements

"Seek knowledge from the cradle to the grave."
Prophet Muhammad (peace be upon him)

During my part-time PhD journey of intensive research for years, I am indebted to many people for their help throughout the process of completing this work.

First and foremost, I would like to praise Allah Almighty; without whose grace and mercy it would never have been possible to achieve this success.

I owe my utmost sincere gratitude to my supervisor Professor Ahmad Lotfi who significantly contributed to making my PhD research a success story. He has been an inspiration and his unlimited encouragement and support motivated me to this point and hopefully beyond.

I am also in great dept to Dr Jun He, my director of studies, who took over me as his PhD student at the later stages of the project and helped a lot with his regular discussions, careful reading and providing useful comments for my publications and thesis.

I would like to express my sincere gratitude to my previous supervisors, Dr Benjamin Inden for his support and insightful criticism as well as Dr Mehmet Emin Aydin for inviting me to work in AI, mainly on pathfinding search algorithms, and providing plenty of guidance and direction. I am thankful to both of whom continued to support me even though they moved to different universities.

My supervisors guided me not only regarding completing this project but also in my entire academic life. It has been an honour for me to be their PhD student.

I am also grateful to wonderful and friendly fellow members of the Computational Intelligence and Applications research group team who shared their knowledge and experience whenever needed and especially to newly graduated Dr Abdallah Naser. Thanks to all of the administrative staff at the Doctoral School and to the library staff, especially Victoria Boskett, who helped make graduate life easier.

My immense gratitude to all my friends who helped me and supported me. My special thanks to my friends Dr Cevat Ozarpa, Furkan Tektas and many others who have been around and prayed for me. I greatly appreciate all their contributions to my success so far.

Love and respect for their unwavering support and patience throughout my studies go to my loving, wonderful wife who always believed in me even when I hesitated about whether I could succeed. She was always stoic and dignified and I am most grateful for all her patience and devotion during these years. I am grateful to my three beautiful daughters, Mesude, Nebahat and Humeyra, with whom I could not spend much time together, especially towards the end of my studies. I would like to thank my sister who morally supported me especially when we lost our beloved mother to cancer.

Finally, and most importantly, I would like to give my special note to my father-in-law, imam Mikdat Kutlu, for his unconditional love, prayers and motivation to pursue my PhD studies.

Aziz Afzalov
November 2022

Abstract

Devising intelligent agents to successfully plan a path to a target is a common problem in artificial intelligence and in recent years, attention has increased to multi-agent pathfinding problems, especially due to the expansion in computer video games and robotics. Pathfinding for agents in real-world applications is a defined problem of multi-agent systems, where pursuing agents collaborate among themselves and autonomously plan their path to the targets.

There are multi-agent algorithms that provide solutions with the shortest path without considering other pursuers and several of those use coordination. However, less attention has been paid to computing an assignment strategy for the pursuers and finding paths that collectively surround the targets. Comparatively fewer studies have been on target algorithms either. Besides, the multi-agent pathfinding problem becomes even more challenging if the goal destinations change over time. Existing solutions consider either a single target with moving capability or multiple targets that are stationary. The work presented in this thesis considers multiple moving targets in multi-agent systems. Therefore, the path planning problem for multiple pursuing agents requires more efficient pathfinding algorithms. In addition, when the target algorithms are improved for advanced behaviour with moving capabilities that smartly evade the pursuers makes the problem even harder.

The research reported in this thesis aims to investigate multi-agent search algorithms to address the challenge associated with pursuing agents towards moving targets within a dynamically changing environment. In multi-agent scenarios, agents compute a path towards the target, while these target destinations in some cases are predefined in advance. Thus, this research proposes to investigate a solution to the path planning problem by utilising heuristic

algorithms as well as assignment strategies for multiple pursuing agents. Furthermore, a state-of-the-art moving target algorithm, TrailMax, has been enhanced and implemented for multiple agent pathfinding problems, which aims to maximise the capture time if possible until timeout.

The focus of this thesis is the investigation of the assignment strategy algorithms to coordinate multiple pursuing agents and explore pathfinding search algorithms to find a route towards moving targets. This will be achieved by dividing it into two stages. The first one is the coupled approach where the assignment strategy with a given criterion finds the optimal combination based on the current position of players. The second stage is the decoupled approach, where each agent independently finds its path towards the moving target. On the other hand, targets flee from pursuing agents using the specified escaping strategy.

The novel contributions of the research presented in this thesis are summarised as follows:

- A new algorithm is developed that uses existing assignment strategies, sum-of-costs and makespan, to assign targets, and then runs repetitive A* search until reaches the target.
- An enhancement is provided for a state-of-the-art target algorithm that takes smart moves by avoiding capture from all pursuers.
- To improve efficiency, six new approaches are investigated to find an optimal agent-to-target combination for target assignment.
- A novel multi-agent algorithm is developed which uses cover heuristics to maximise its coverage to outmanoeuvre, trap and catch moving targets.

The proposed pathfinding solutions and the results presented in this thesis demonstrate a significant contribution towards search algorithms in multi-agent systems.

Publications

As a result of the research presented in this thesis, the following publications have been published:

Refereed Journal Papers:

Azizkhon Afzalov, Ahmad Lotfi, Benjamin Inden, and Mehmet Emin Aydin. "A strategy-based algorithm for moving targets in an environment with multiple agents." **SN Computer Science**. **3**, 435 (2022).

Azizkhon Afzalov, Ahmad Lotfi, Benjamin Inden, Jun He, and Mehmet Emin Aydin. "Coordinating Multiple Agents with Assignment Strategy to Pursue Multiple Moving Targets." **Spinger Nature Progress in Artificial Intelligence** (Under Review).

Journal Papers to be Submitted:

Azizkhon Afzalov, Jun He, Ahmad Lotfi, Benjamin Inden and Mehmet Emin Aydin. "Increasing Covered Area to Capture Moving Targets in a Dynamic Environment." **Elsevier Expert Systems With Applications**.

Azizkhon Afzalov, Jun He, Ahmad Lotfi, Benjamin Inden, and Mehmet Emin Aydin. "Adaptive Weighted-Cost Assignment Strategy for Efficient Multi-Agent Path Planning." **SN Computer Science**.

Refereed Conference Papers:

Azizkhon Afzalov, Jun He, Ahmad Lotfi, and Mehmet Emin Aydin "Multi-agent path planning approach using assignment strategy variations in pursuit of moving targets." Agents and Multi-Agent Systems: Technologies and Applications 2021, the 15th KES International Conference, Springer, (2021).

Azizkhon Afzalov, Ahmad Lotfi, Benjamin Inden, and Mehmet Emin Aydin, "Multiple pursuers TrailMax algorithm for dynamic environments." The 13th International Conference on Agents and Artificial Intelligence, ICAART, (2021).

Azizkhon Afzalov, Ahmad Lotfi, and Mehmet Emin Aydin, "A strategic search algorithm in multi-agent and multiple target environment." The 8th International Conference on Robot Intelligence Technology and Applications 2020 (RiTA 2020), Springer, (2021).

Azizkhon Afzalov, Ahmad Lotfi, and Jun He, "Coupled Assignment Strategy of Agents in Many Targets Environment." The 15th International Conference on Agents and Artificial Intelligence, ICAART, (2023). (Under Review).

Workshop Papers:

Azizkhon Afzalov, "Novel Techniques for Moving Target Search in Dynamic Environments." New Horizons in Digital Media (NHDM12) workshop organised by the University of Bedfordshire, (2012).

Posters:

Azizkhon Afzalov, Edmond Prakash, and Mehmet Emin Aydin, "Novel Techniques for Moving Target Search in Dynamic Environments." The 26th International Conference on Computer Animation and Social Agents, (2013).

Nomenclature

The following Acronyms are used throughout the thesis.

ADG	Action Dependency Graph
ADP	Agent Decomposition Planner
AI	Artificial Intelligence
APSP	All-Pair Shortest Path
BCP	Biased Cost Pathfinding
CC	CombinationCoverage
CPD	Compressed Path Databases
CPF	Cooperative Path-Finding
CT	Constraint Tree
DAI	Distributed Artificial Intelligence
D-MAPF	Dynamic Multi-Agent Path Finding
d-mode	dynamic assignment mode
DS	DistancesSum
h-value	heuristic value
IHA	Incremental Heuristic Algorithm
MAM	Multi-Agent Meeting
MAP	Multi-Agent Planning
MAPD	Multi-Agent Pickup and Delivery
MAPF	Multi-Agent PathFinding
MAPR	Multi-Agent Planning by plan Reuse
MAS	Multi-Agent Systems
MAT	Multiple Agents and Targets
MD	MaxDistance

MG-MAPF	Multi-Goal Multi-Agent Path-Finding
MPP	Multirobot Path Planning on graphs
MRPP	Multi-Robot Path Planning
PAMT	Pursuing Agents and Moving Targets
RHCR	Rolling-Horizon Collision-Resolution
SIC	Sum of Individual Costs
s-mode	static assignment mode
SOC	Sum-Of-Costs
TAPF	Target-Assignment and PathFinding
UGV	Unmanned Ground Vehicle

The following Acronyms for Algorithms are used throughout the thesis.

A-MTS	Abstraction MTS
CBM	Conflict-Based Min-Cost-Flow
CBS	Conflict Based Search
CDMTA*	Cover Dynamic Moving Target A*
CRA	Cover with Risk and Abstraction
D*	Focused Dynamic A*
DAM	Dynamic Abstract Minimax
eMIP	efficient path planning
FAR	Flow Annotation Replanning
F-MTS	Fuzzy MTS
GAA*	Generalized Adaptive A*
HCA*	Hierarchical Cooperative A*
ID	Independence Detection
LPA*	Lifelong Planning A*
LRA*	Local Repair A*
LRTA*	Learning Real-Time A*
MA-CBS	Meta-Agent Conflict Based Search
MLA*	Multi-Label A*
MM*	Multi-Directional Meet in the Middle

MMTS	Multiple Agent–Based Moving Target Search
MPGAA*	Multipath Generalized Adaptive A*
MPTM	Multiple Pursuers TrailMax
MTES	Real-Time Moving Target Evaluation Search
MTS	Moving Target Search
Path-AA*	Path Adaptive A*
PRA*	Partial-Refinement A*
RTA*	Real-Time A*
RTAA*	Real-Time Adaptive A*
SF	Simple Flee
STMTA*	Strategy Multiple Target A*
TBAA*	Time-Bounded Adaptive A*
TP	Token Passing
TPTS	Token Passing with Task Swap
Tree-AA*	Tree Adaptive A*
WHCA*	Windowed Hierarchical Cooperative A*

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Pathfinding consists of a particular order of movements with associated cost values that lead from the starting position to the intended goal position. It involves going around either static or moving obstacles. The total sum of movements can be optimal if the total cost is the lowest among all possible computed paths [2]. Pathfinding search algorithms have been one of the interesting and challenging problems in the Artificial Intelligence (AI) research field [3], and there has been extended work for many years [4, 5, 6]. The study and development of such algorithms were based on the basic scenario of a single agent tasked with finding a target or goal state on a grid-based map with minimum cost or within minimal time. In environments with multiple agents, which are complex, the problem becomes more challenging than in a simple, static, single-agent environment [7]. With various assumptions of this single agent with a single target, the scenario can be relaxed, while the following aspects can lead to further complexities:

- There can be multiple pursuing agents that need to coordinate their actions.
- Assigning a strategy to the agents before chasing targets.
- Existence of multiple targets and their ability to move on the map over time rather than being in a fixed position.
- Pursuing agents and targets have complete (known) or limited (partially-known) information about the environment.

In recent years, attention has increased to pathfinding problems in multi-agent systems, mainly due to the expansion in video games [8, 9, 10], robotics [11, 12, 13], and warehouse management [14, 15]. An example of a robotics application is the Amazon warehouse, where autonomous robots can lift and carry storage pots and transport them between staging areas [16]. Similar demands can be observed in space exploration, for instance, Curiosity [17] or Perseverance [18] Mars rover prototypes, in the surveillance of moving targets for security reasons in authorised areas [19, 20], in the military applications [21, 22] or with Endeavor Robotics' The 510 PackBot robot [23], in autonomous aircraft or underwater vehicles [24], or in search and rescue operations [25, 26], where robots are tasked to aid rescue teams in life-threatening conditions such as in urban disaster environments.

Pathfinding can relate to single-agent and multi-agent problems. Various suitable pathfinding solutions have been proposed for the mentioned application domains. Some of these pathfinding algorithms are for a single agent, such as Moving Target Search (MTS) [27], D* Lite [28], Real-Time Target Evaluation Search (RTTES) [29] or Adaptive A* [30]. Similarly, there are some multi-agent pathfinding search algorithms, for instance, Flow Annotation Replanning (FAR) [31], Windowed Hierarchical Cooperative A* (WHCA*) [32], Conflict Based Search (CBS) [33], Partial-Refinement A* (PRA*) [34] or Multiple Agents Moving Target (MAMT) [35]. However, these algorithms aim to find the shortest path to the target position. While the shortest path is important, the run time is essential, too, as considered by real-time heuristic algorithms [36].

The pathfinding problem for multi-agent systems in real-time is more challenging [31]. It requires agents' navigation towards the moving goal (target) while avoiding any static or dynamic obstacles. The agents need to manage the shared information data, collaborate, and work collectively to achieve the goal [37]. Therefore, this thesis investigates an efficient pathfinding algorithm for multi-agent environments. Moreover, the solution needs to develop a competent algorithm where the pursuing agents cooperate within an environment where targets are dynamic. It is expected that multiple targets are considered with moving away the ability to escape from pursuers in the problem scenario. These approaches are addressed in multiple Pursuing Agents and multiple Moving Targets (PAMT) environments. Further, this research contributes to solving

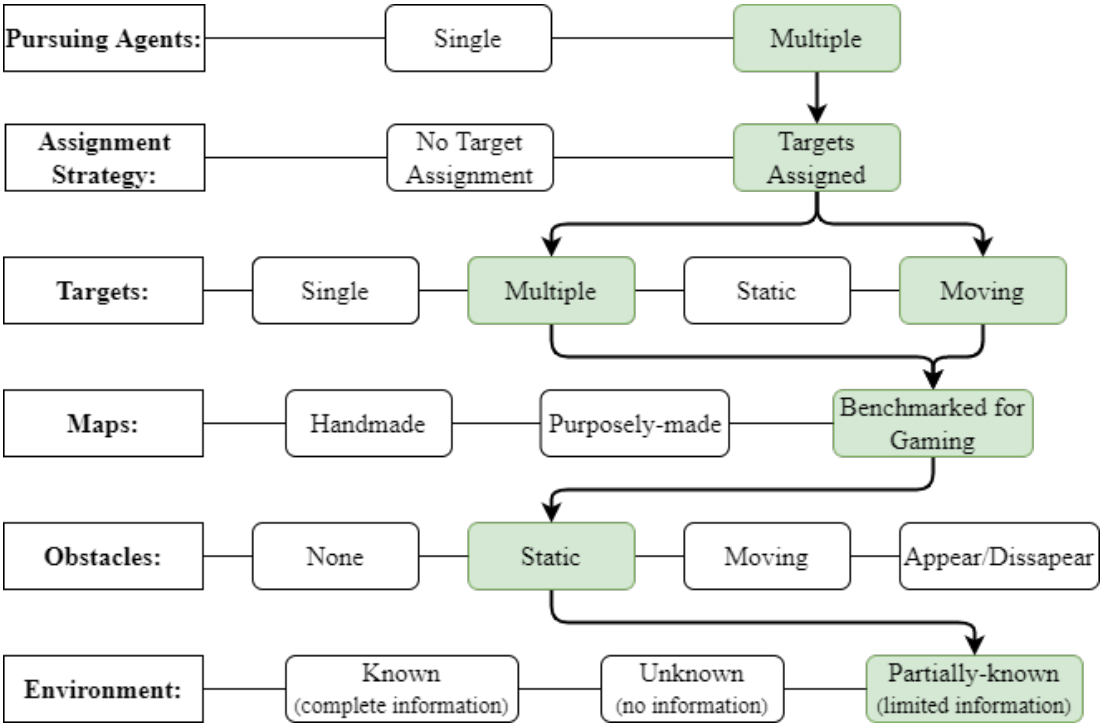


Figure 1.1: Identifying the problem setting for multi-agent pathfinding. The arrows navigate the highlighted parts which are the focus of this research.

major limitations of assignment strategy to find an optimal pursuer-to-target combination, i.e., assigning targets to pursuers in various settings. In addition, the research contributes to solving the limitations of an intelligent target algorithm to escape from pursuers by moving to the furthest away position to prevent capture. Derived results from the proposed approach, such as the pathfinding cost, success rate or runtime display significant improvements by proposing novel approaches to the multi-agent problems.

For illustration purposes, Figure 1.1 depicts the main aspects considered in identifying the problem for pathfinding. Each main component in rectangles on the left has different options that could be applied in the search scenarios. The highlighted parts with navigated arrows are the focus of this research.

The remainder of this chapter is organised as follows: an overview of this research is discussed and described in Section 1.1 followed by presenting the research aim and highlighting the proposed objectives in Section 1.2. The scope

of the research is included in Section 1.3. The original contributions achieved throughout the undertaken work are introduced in Section 1.4. Finally, the structure of the thesis is provided along with summaries of each chapter's contents in Section 1.5.

1.1 Overview of the Research

The prime motivation of this research is to solve a pathfinding problem for multiple pursuing agents that can coordinate in an environment where targets can move. The agents' common objective is to catch all targets successfully. This is the problem of multi-agent systems, where the circumstances of the environment, capabilities of the pursuing agents and their relations occupy key roles in it [38]. Nevertheless, the problem of pathfinding for cooperative multiple agents is subject to obstacles towards static or moving multiple targets. Heuristic search algorithms are promising AI approaches to address these types of dynamic problems. On the other hand, the pursuing agents need to coordinate and join their efforts to achieve their objectives. Thus, it requires an approach to assigning targets using the criteria from the assignment strategy. The research overview depicted in Figure 1.1 navigates the problem by outlining the components that are considered in this study.

In the context of this research, the problem of multi-agent systems is approached in two stages, coupled (task planning as a single composite entity) and decoupled (autonomous navigation of each pursuing agent). The intelligent agents can be either pursuers or targets. In the coupled stage, the pursuers should be able to identify targets and assign each target to the pursuer. Then, in the decoupled stage, each pursuer should be able to chase the moving targets independently from each other. The pursuit employs two different approaches that use different distance metrics: distance heuristic and cover heuristic to catch all targets successfully. The distance heuristic estimates the shortest path of an optimal cost between a pair of states while the cover heuristic takes an action that maximises the area that a pursuer can cover before reaching a target, which at the same time minimises the area of coverage for the target to escape. Throughout the thesis, the words *agent* or *pursuer* or *pursuing agent*

are used interchangeably. Similarly, the word *dynamic* describes an environment where the size or the shape of the grid maps does not change and obstacles do not move only the position of pursuers, as well as targets, can change within the time steps unless it is specified differently while reviewing literature in Chapter 2.

1.1.1 Multi-Agent Systems

Multi-Agent Systems (MAS) are a subcategory of Distributed Artificial Intelligence (DAI), which is a subcategory of AI [39]. MAS research considers independent agents that are teamed to share knowledge and communicate in between to find a solution to a problem that is beyond the capabilities of a single agent [40]. As a result of the communication, the agents in MAS can interact and have different linked relations in the environment [38]. Moreover, when there are many autonomous agents present, it is possible to specify rules in which each agent employs the method to maximise its value as well as maximise the overall value for all agents [41]. In MAS, the relation among the intelligent agents primarily focuses on solving the problems that are difficult to find solutions with the knowledge that a single agent has [42]. Additionally, the interaction and coordination of agents and their solutions to complex problems in MAS have demonstrated their functionality in complex applications and dynamic environments [43]. For instance, MAS develops new methods and techniques for learning the behaviour which has been experimented with in the Pac-Man computer game environment. Another solution is Double Action markets for online bidding in trades for multiple buyers and sellers or smart city designs which contain various sensors that collect data or share bicycles [43].

The agent in MAS relates to an independent entity that has a number of actions before reaching its goal. The agent receives instructions and makes independent decisions about its plan of action based on the instructions. More complete and detailed information increases the probability of making well-informed decisions. The agent observes its surroundings and can respond to changes that are detected [38]. Agent's actions are performed in an environment that is accessible with complete information about the state. The position of

agents and targets can change over time and the agent has no control over this. Moreover, there is an interaction between the agents upon which success requires cooperation, coordination, and negotiation. It is generally accepted that interaction is perhaps the most significant aspect of advanced multi-agent systems [44].

1.1.2 Pathfinding

Agent's path planning consists of a particular order of movements with cost values that lead from the starting position to a goal position [45]. The movement of the agent is considered optimal if the sum of the costs of the individual steps is the lowest among all possible paths [2]. Finding the lowest cost path from the agent to the goal in real-time is often difficult or impossible due to the large search spaces [46]. In some scenarios, the agent may not know the environment in advance and need to predict the route to the goal using heuristic search methods [47].

Existing algorithms, for instance, MTS [27], D* Lite [28], PRA* [34] or CBS [33], aim to find the shortest path towards the static or moving target state and it is mostly achievable when there is one admissible target. However, the task of navigating an agent towards a target becomes more difficult when constraints are tightened, and more complex variations of problems are introduced. For instance, the problem may be subject to a single constraint or a combination of constraints, such as map types (with no obstacles, cycle-shaped, corridor-shaped or roadmaps), the presence of more players (pursuing agents and targets), static or moving obstacles, and the target being able to flee the capture or wait until the rescue arrives [26]. Moreover, these issues have been extended to multiple agent scenarios, such problems as Cooperative Path-Finding (CPF) [48, 49], Multi-Agent PathFinding (MAPF) [50, 51], Dynamic Multi Agent PathFinding (D-MAPF) [52], Multiple Agents and Targets (MAT) [53], Multi-Robot Path Planning (MRPP) [54, 55], Multirobot Path Planning on graphs (MPP) [4], Multi Agent Planning (MAP) [56, 57] Multi Agent Planning by plan Reuse (MAPR) [58] or Multi-Agent Meeting (MAM) [59].

1.1.3 Assignment Strategy

A simple strategy to find a low-cost distance towards the target and move to its position can promise a capture if the speed of agents is faster than the target. An example is the real-time strategy video game, Company of Heroes, where a team of soldiers move quicker [60]. In an environment with multiple players, the search for the path from agents toward the targets is more complex and requires well-thought, rigorous task planning. The most straightforward solution for the agents is to follow the nearest target. However, it might not be the best option in the presence of multiple targets. If all agents choose to follow the target that is closest in the distance, then targets that are not pursued can affect the total pathfinding cost, success, and runtime.

In pursuing games, such as cop and robber, prey and hunter, and military simulated applications, players can move and change their positions, and this makes it difficult to plan, search and navigate towards the targets while avoiding obstacles. The challenge increases when the targets are not stationary and their number increases. The moving targets can evade capture while time permits if the pursuers do not have a winning strategy [61]. Therefore, well-defined assignment strategies aim to help efficient planning, reduce computation time, increase the success of the task, and affect the total performance of catching all moving targets. Thus, an assignment strategy is important, and a good assignment strategy is essential for the desired outcome.

Multiple pursuers can benefit from two stages, which are coupled and decoupled pathfinding algorithms. The coupled approach focuses on planning and distributing tasks to all pursuers as a single task, whereas the decoupled approach concentrates on finding a path individually. The assignment strategy algorithm with the given criterion initially computes all possible combinations (pursuer-to-target route) for all pursuers in the coupled stage. This coupled approach produces optimal solutions [62], however, the computation increases exponentially with the number of pursuers. The combination of assigning pursuer-to-target with the lowest value, i.e., an optimal result, gets all targets assigned to the pursuers before the move, and none of the pursuing agents should be idle. Once the targets are assigned, the next stage starts, where all

pursuers search their path independently and navigate themselves towards the moving targets using the heuristic search algorithm. This decoupled approach can be fast but occasionally fails in finding a complete solution [51, 63] because of conflicts that can arise afterwards, where pursuers let others pass [64]. Though its usage is practical, especially with a large number of pursuers [32] and robust as if one pursuer fails, that does not affect the entire team's success [65].

1.2 Research Aim and Objectives

The work presented in this thesis aims to develop new multi-agent pathfinding algorithms by proposing approaches that surround and trap the targets instead of running straight towards them. It also includes new assignment strategy criteria to assign targets to the pursuers. Additionally, a new target algorithm that can escape intelligently from the approaching pursuing agents. The new approach for the pursuers is that find an optimal assignment strategy to assign targets before the search starts and then use search algorithms to find the path. The research tries to explore a solution that involves an improvement through the use of cover heuristics, where the pursuing agents can search a path by taking different routes to outmanoeuvre and trap the targets. To achieve the project aim, the following research objectives have been identified:

1. Investigate a method for pursuers by implementing an assignment strategy while using repetitive heuristic searches towards multiple moving targets.

2. Extend the state-of-the-art target algorithm, which has been proposed for single-agent single-target, towards multiple pursuers by implementing a smart escape mechanism for targets.

3. Study how effectively the proposed and enhanced algorithms perform in the dynamic environments with multiple moving targets.

4. Propose novel methods for coordination that use the assignment strategy which computes combinations for pursuing agents by utilising the sum of costs and the makespan to find an optimal combination in assigning targets.

5. Extend the multi-agent CRA algorithm, which has been developed for a single moving target, by proposing a solution algorithm for multiple pursuers that chase to capture multiple moving targets by implementing an assignment strategy and a cover heuristic.

6. Implement the novel pursuing search algorithms using the assignment strategies and the new target algorithm in the dynamic environment using commercial gaming benchmarks.

7. Compare and statistically analyse the performance of all proposed algorithms in different environments with different pursuer-to-target ratios on various grid-based maps.

1.3 Scope of The Research

Multi-agent pathfinding algorithm is an important [66] and a broad [67] research area in AI with many different variations, extensions and applications. In general, it addresses the movement of the pursuing agents, the ability to determine where to execute the next step and consider its move strategically [68]. It serves an essential role in AI studies by providing agents with the opportunity and ability to make decisions intelligently [69].

For instance, games are one of the applications where pathfinding problem is widely studied and their environment is an excellent testbed [8] as seen in World of Warcraft [70] or Command and Conquer [71] video games. However, the games industry is so diverse, that this cannot be considered indicative of all games [72]. Although Figure 1.1 indicates the problem of this study, it is relevant to outline the scope of this project.

The pathfinding problem can be studied in various settings and configurations [73]. It is possible to plan a path on known, unknown or partially-known environments and the pathfinding can be categorised as deterministic or probabilistic. The deterministic methods applied on grid environments allow for a similar outcome to be obtained in each execution with the same starting settings, whereas, the probabilistic methods are more suitable for real-time algorithms [74]. Although the environment, the shape of the

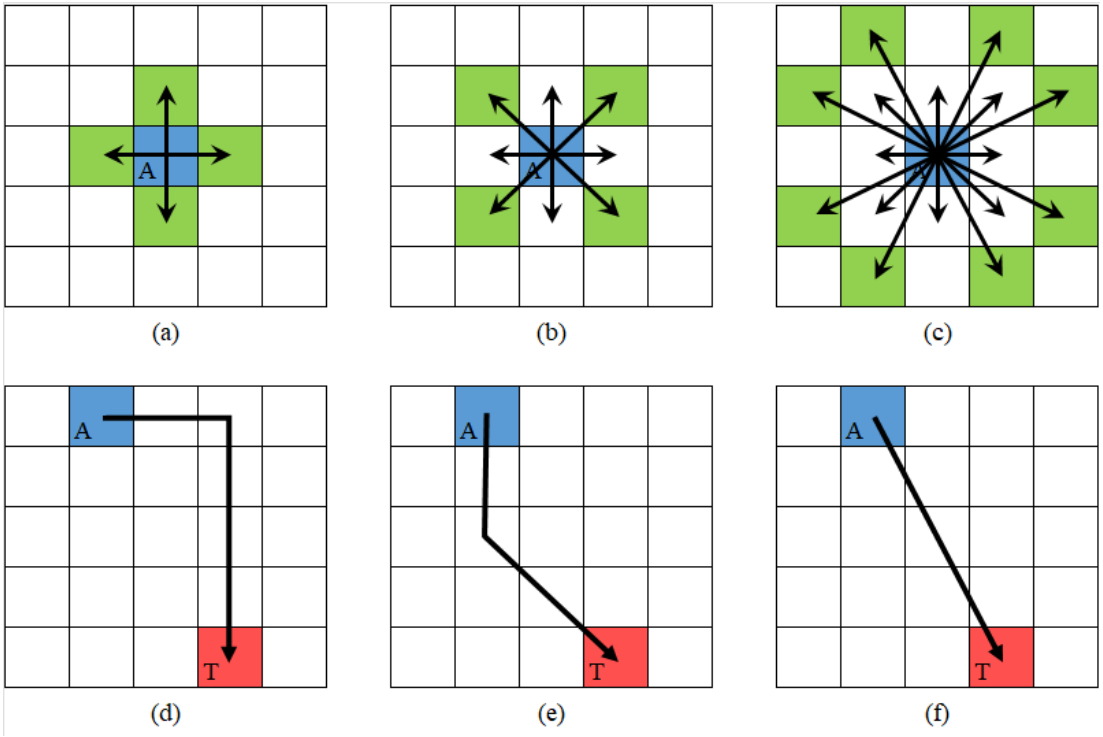


Figure 1.2: The representation of three different distance metrics with a cost of 1 for an orthogonal (a), $\sqrt{2}$ for an octile (b) and $\sqrt{5}$ for the Euclidean (c) movement directions. The possible route direction between two points, agent (A) and target (T), is for Manhattan (d), diagonal (e) and Euclidean (f) distance.

states, can be specified as square, triangular, or hexagonal, it is common to use square grids in video games or robotics as the implementation is easier and simpler [75]. This also shapes the movement direction of the players.

Every possible movement associates a cost within, which determines the final cost of the path taken. There are three metrics that define a way of a next successful state. An orthogonal distance is an adaptation of four discrete moves horizontal and vertical, and a diagonal distance includes four additional movement directions (northeast, northwest, southeast and southwest). The Euclidean distance [76], with a smoother path, can move in sixteen adjacent states as illustrated in Figure 1.2 for costs and routes. When the heuristic distance between a pursuer and a target is Manhattan then the cost is one for each move. If the heuristic distance is a diagonal move then the cost to the adjacent state is $\sqrt{2}$ and the cost for Euclidean is $\sqrt{5}$. Although Manhattan and

octile heuristics are common, the Euclidean can provide less angled, more realistic paths, nonetheless, it can underestimate the actual cost and provide computationally expensive solutions [77].

The presence of obstacles also shapes the maps. The experiments can be conducted on maps with no obstacles, with random obstacles, mazes, maps with street or road shapes, or even commercial gaming map environments. Baldur's Gate video game testbed environments, the two-dimensional grip maps, are the main sources to explore the behaviour of novel algorithms in this thesis.

The type of the map environment whether they are a maze, road shapes or random obstacles and the benefits of these maps, as well as the direction of movements and details of distance metrics, are beyond the scope of this research.

Targets are able to move with similar heuristics and at the same speed as the pursuers [78] and the pursuers do not have control over the movements of the target or its escaping directions [79]. There are some experiments that have been conducted in which the target skips a move [80] while pursuing agents continue the chase, eventually catching the target. Meanwhile, this study confines not on chasing directly and "rush-in" towards it but coordinating among pursuers to surround and trap one target or multiple targets for a successful outcome. This coordination can be achieved with the coupled approach which unfortunately has limits on how many pursuer-to-target combinations can be computed and due to its exponential growth with the number of pursuers and limited computing resources, it is a very difficult task to find combinations, for example, ten pursuers [81]. In this regard, it is not very scalable and is only limited to up to 5 pursuers where the assignment strategy algorithms within the coupled approach are able to handle the tasks. There are algorithms that scale and solve the problem on a one-to-one basis and some of them are described in the literature review Chapter 2, however, the scalable solutions for larger numbers are beyond the scope of this research, as the approach is to find solutions for multiple pursuers towards multiple targets, many-to-many scenarios, rather than individual independent search solutions.

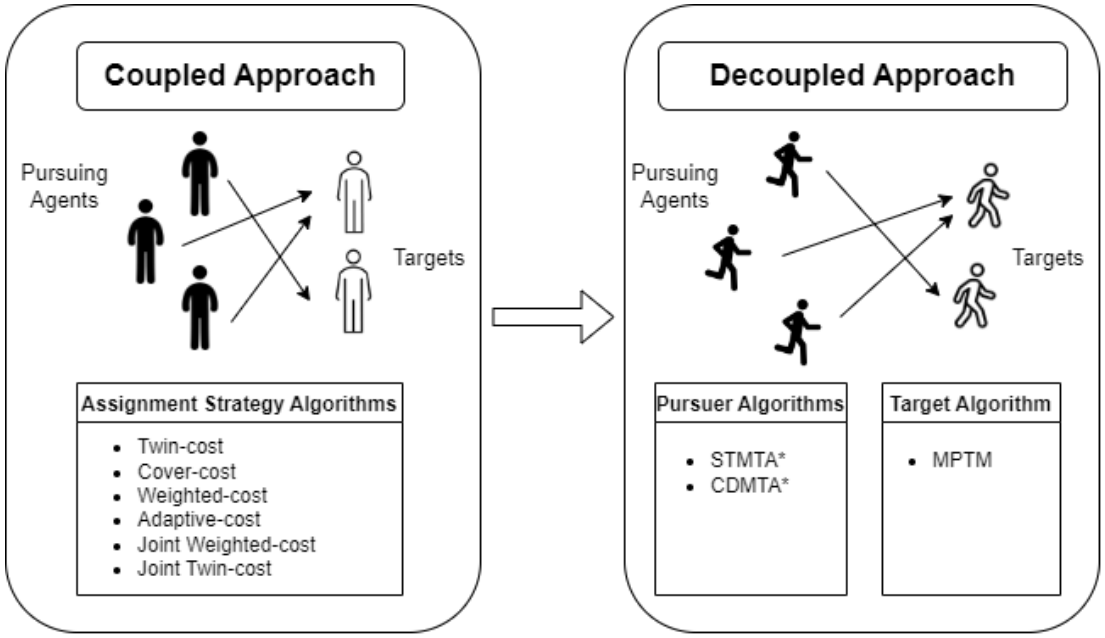


Figure 1.3: Coupled approach illustrates two targets getting assigned to three pursuers and lists new algorithms, while the decoupled approach is the chase with lists of new algorithm contributions to pursuing agents and targets.

1.4 Original Contributions of the Thesis

First, the targets get assigned to the pursuers in the coupled approach stage using six new assignment strategy algorithms bullet-pointed inside the box of Figure 1.3. Then, when the test run starts each player can make a move depending on the algorithm provided in the decoupled approach stage. The thesis presents two new algorithms for the pursuers and one new algorithm for the target.

The major contributions of the work presented in this thesis are listed in Figure 1.3 and the following points below are grouped in line with the coupled approach for assignment strategies and decoupled approach for pursuing agents and targets:

- An extensive literature review of the search algorithms for pathfinding including, single-agent search algorithms, as well as multi-agent algorithms. It includes various suggested methods and approaches from earlier studies that are pertinent to this thesis.

- An enhanced framework for search algorithms extending from a single target to adapt to multiple targets.

- Testing and evaluating the proposed algorithms using different pursuer-to-target combinations on standardised grid-based maps from the commercial game industry used as a benchmark.

Coupled Approach:

- Novel approaches for assignment strategies are proposed. The sum of costs and the makespan are both used to investigate various criteria to find the optimal combination for the target assignment.

- An exploration of the efficient criteria for the assignment strategies to be applied for pursuing agents. Besides, comprehensive experiments are conducted using Manhattan and diagonal movement directions to validate the use of the assignment strategies.

Decoupled Approach:

- Propose a novel technique for multiple pursuers that uses existing assignment strategies such as the Summation-cost and Mixed-cost criteria and finds its path from adjacent available states towards moving targets.

- A novel approach for multiple agents in pursuing multiple moving targets. The proposed approach uses the assignment strategy to identify and assign targets to the agents and finds the path with cover heuristics that maximise the coverage area to trap the targets. Also, the approach is adaptive to work in different dynamic environments.

- A new approach to identifying the escaping route and providing the solution for the targets. The developed approach considers multiple pursuers and makes a smart escape to avoid capture.

1.5 Thesis Outline

This thesis consists of eight chapters. Figure 1.4 illustrates the structure of the thesis, which provides an organisational overview for readers with an indication of how the chapters are linked. The ordering of the introduction of the algorithms develops initially providing a simple and effective solution to multi-agent problems with moving targets in Chapter 3 and moves to introduce new assignment strategy algorithms in Chapter 4, then proposes a target algorithm that intelligently escapes considering all pursuers in Chapter 5. Alongside, Chapter 6 introduces furthermore new assignment strategy algorithms and finally, the last Chapter 7 presents an optimal solution to a complex problem. The summary of the contents of this thesis is presented as follows:

Chapter 2: Literature Review - This chapter provides an overview of previous work in the field of search algorithms for pathfinding. In particular, the literature focuses on single-agent and multi-agent algorithms. The chapter gives an overview of the available methods and a discussion on relevant literature that is used for pathfinding. Also, the review of the previous research is summarised to identify the research gaps and highlight how this work differs from previous research works.

Chapter 3: Coordinating Multiple Agents with Assignment Strategy to Pursue Multiple Moving Targets - This chapter investigates a solution to the pathfinding problem of multiple pursuing agents towards moving targets within dynamically changing environments. A pairwise distance is computed between pursuing agents and targets at their initial positions, and the optimal combination assigns targets to the pursuers. Based on this initial assignment and with some random movements, the pursuing agents outmanoeuvre targets even though they only use independent heuristic searches, such as the A* algorithm. This is a simple and effective method that performs better when compared to the existing approach in multi-agent scenarios.

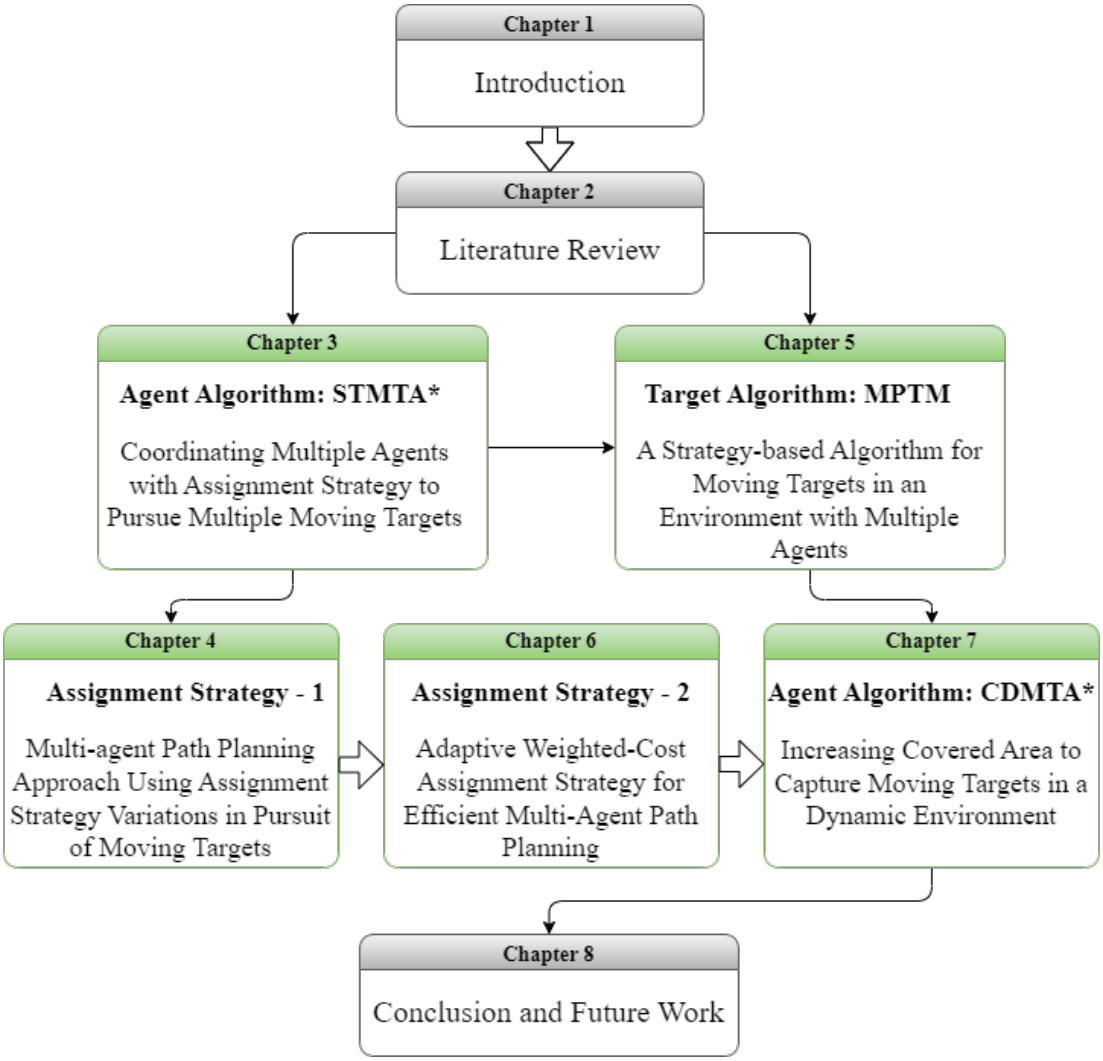


Figure 1.4: The thesis structure shows the organisation of the chapters and their respective dependencies.

Chapter 4: Multi-agent Path Planning Approach Using Assignment Strategy Variations in Pursuit of Moving Targets - This chapter investigates the problem of assignment strategies for multiple agents. Sometimes, the agents compute paths towards the static targets, while these target destinations are predefined in advance. On the other hand, the assignment strategies identify and assign a target at the initial position, before making any move. This problem is more challenging if the targets move on the map. This chapter presents new approaches to the assignment strategy to improve efficiencies by introducing

three novel approaches in multiple moving target environments. These new methods are tested against existing overall the best approach in the literature. The experimental results suggest that the new assignment strategy methods exhibit better results in comparison with the existing approaches.

Chapter 5: A Strategy-based Algorithm for Moving Targets in an Environment with Multiple Agents - This chapter introduces a state-of-the-art target algorithm, TrailMax, which works with one agent and one target combination. It has been enhanced to consider multiple approaching pursuers and implemented for multi-agent pathfinding problems. The presented algorithm aims to maximise the capture time if possible until timeout. The empirical analysis is performed on grid-based gaming benchmarks, measuring the capture cost and the success of escape. The new algorithm, Multiple Pursuers TrailMax, doubles the escaping time steps when compared with existing target algorithms and increases the success rate.

Chapter 6: Adaptive Weighted-Cost Assignment Strategy for Efficient Multi-Agent Path Planning - This chapter is an extension of the assignment strategies provided in Chapter 4, and it introduces three further new approaches to assigning targets to the pursuing agents. The chapter explores the new methods for the assignment strategy algorithm for multiple pursuing agents where agents must adapt their decisions according to the target moves and find the optimal combination based on the current position of all players. The performance measures the computation time for assigning targets, the pathfinding cost for pursuers, and the success of the task.

Chapter 7: Increasing Covered Area to Capture Moving Targets in a Dynamic Environment - This chapter proposes a novel algorithm for multiple agents in pursuing multiple moving targets. A novel algorithm is developed using the existing assignment strategy algorithms and the cover heuristics approach, where the pursuing agents take the surrounding approach instead of the shortest path in the dynamic environment. The presented algorithm aims to outmanoeuvre and trap the moving targets and the results obtained from the

experiments provide satisfactory outcomes. Similar to the previous work, the tests are conducted on benchmarked maps and the results are compared for pathfinding cost, success rate, minimum and maximum cost and runtime.

Chapter 8: Conclusion and Future Work - This chapter presents the conclusions arising from the research conducted in this thesis. The major findings obtained in this thesis are discussed with a reflection on the research questions identified in Chapter 1. Following the summary of the findings, the chapter further suggests possible directions for future work on pathfinding for multiple pursuing agents.

Chapter 2

Literature Review

2.1 Introduction

Finding a path and navigating the pursuing agent from its starting position to a target position while avoiding obstacles is a well-known problem in AI [63, 66, 82]. A single pursuing agent is the only agent on the map, whereas, multiple pursuing agents present two or more agents, however, multiple pursuers can proceed as a single agent or collectively as one entity. In the presence of a single pursuer in the environment, the A* algorithm [83, 84] is a classic example that is used for agents that can be an effective solution. Furthermore, the environment becomes challenging with multiple pursuing agents while each pursuer is given a target position to reach, assuming it is static. However, relaxing the assumption and repositioning the targets' positions make the multi-agent pathfinding problem more complicated [85]. While the problem is becoming increasingly important [86], issues with coordination, target assignment, communication, obstacle or collision avoidance, outsmarting targets while reaching with fewer time steps and moving quicker in a limited time need to be considered [87]. Therefore, it is essential to review existing search algorithms to justify the intent of the research work in this thesis.

This chapter surveys the related literature and provides a comprehensive review of the previous work by analysing their approach to solving pathfinding problems. Although there is a wide range of literature related to the topic, this

chapter concentrates on the literature relevant to the multi-agent problem. Initially, single-agent pathfinding algorithms are presented which are regarded as incremental heuristic and real-time algorithms. They provide various solutions that navigate a single agent towards a single target. Alongside these algorithms and while the problem increases with the number of pursuers, multi-agent pathfinding algorithms are introduced. It is possible to see some of the single-agent methods to be enhanced and developed to multi-agent algorithms. Multi-agent algorithms solve more complex problems and they do not necessarily chase one target, but there could be multiple targets. Depending on the problem scenarios, the target can be positioned on the map and stand still until an agent reaches it, or the target can move around the map and avoid being caught. Hence, some of the target algorithms are also discussed. The survey provides fundamental knowledge and an understanding of pathfinding search algorithms and their characteristics, variations, implementations, and issues.

The remainder of this chapter is structured as follows: Section 2.2 gives an overview of single-agent algorithms categorised as incremental heuristic algorithms and real-time algorithms, and includes the A* algorithm. Section 2.3 provides background on multi-agent algorithms and their approach towards single or multiple targets. Section 2.4 reviews the research on target algorithms. Section 2.5 defines the problem and presents existing assignment strategy criteria. Additionally, the section designs and structures the experiments that are used in the following chapters. To conclude this chapter, Section 2.6 summarises and analyses the limitations of existing algorithms and indicates a research gap.

2.2 Single-Agent Algorithms

Getting from the pursuing agent's starting position to the target is the pathfinding problem that has been addressed by many single-agent algorithms. The objective is to find an optimal path, if one exists, by employing a search algorithm [88]. The path should avoid obstacles and find a cost-minimal approach in real-time to the static or moving target [79]. Heuristic search algorithms should find a path with

minimal costs and do so using minimal computation time. They should be able to improve their performance over time when used in an environment that does not change much. This section briefly introduces the A* algorithm and discusses other algorithms that are classified as incremental heuristic algorithms [89] and real-time algorithms [90] that provide the solution for a single agent.

2.2.1 A* Algorithm

The A* algorithm has broadly been applied to many single-agent problems in AI [84, 91, 92] and is the basis of many search algorithms [68] as well as described in this project. It is an offline algorithm that must find the whole path even before committing to take the initial action. It estimates the optimum path by using heuristics. If the heuristics are admissible, then it is guaranteed that it finds the shortest path without overestimating the path cost from the position of the agent to the target [88]. A* requires exponential space which is the main disadvantage [93].

A* keeps the cost of discovered path values *g(s)* of the shortest found length from the initial state to the state *s*; a heuristic value *h(s)* is the heuristic distance from state *s* to the target state; $f(s) = g(s) + h(s)$, that is estimated distance to the target state via state *s*. Once the search is complete, it uses *tree (s)* to identify the shortest path in reverse [94].

A* has two sets of states, OPEN and CLOSED. The initial state is added to the OPEN set. A state with the lowest *f(s)* is removed from the OPEN set and inserted into the CLOSED set. If the *f(s)* of state *s* is no smaller than the *f(s)* of the target state, then the target is found and A* terminates. Otherwise, it continues the loop by expanding every neighbouring state *s* if it's traversable or not already in the CLOSED set. It assigns *f(s)* to the neighbouring state *s* if it is a shorter path or not in the OPEN set. Next, it assigns *tree (s)*, i.e. parent, to the neighbouring state to point to state *s*. Finally, the neighbouring state is added to the OPEN set, if it is not there already. Then, the procedure repeats [95].

2.2.2 Incremental Heuristic Algorithms

For efficient planning, an Incremental Heuristic Algorithm (IHA) uses a technique that finds a solution by replanning a path with information obtained from previous searches [96]. After computing the initial search path, the algorithm reuses the same information to make its next search faster instead of starting from the beginning again. Through this, it performs faster compared to repeated A* searches [97]. In the AI world, the dynamic is essential. If the current world has changed or the target has moved, then the initial search path may not be relevant, and it needs to be recalculated, learn a better path, or be refined by reusing the previous search. Reusing the search is useful if not much change is needed to the previous best search path [89].

One of the algorithms that were developed in a combination of incremental search and heuristic search is Lifelong Planning A* (LPA*) [98]. It continuously finds the shortest path from the start position to the goal position by reusing details from the previous search on partially-known finite graphs [99]. It expands positions similarly to A* in the first search. In following searches, it expands them maximum twice and not all of them whose values are equal (incremental search's efficiency) or whose heuristic values are larger than the goal (heuristic search's efficiency) [89].

To speed up the searches and increase the efficiency in navigation strategy for computer games or mobile robotics, Dynamic A* (D*) [100], and its extension Focused Dynamic A* [101] algorithms were proposed. Focused Dynamic A* has been extensively used and is the most known incremental heuristic search algorithm; it is a great achievement in robotics [89]. Computing the shortest path for Focused Dynamic A* is similar to the LPA*, hence the combination of both algorithms resulted in presenting the D* Lite algorithm for moving robots. D* Lite uses the same route strategy as Focused Dynamic A* and both are equally fast [102]. But the D* Lite algorithm is simpler, easy to understand, implement and extend [28]. Both search from the target state to the agent's current state. The agent notes obstacles around its current position and moves towards the stationary target. The root (target) of the search tree stays unchanged. Thus, D* Lite reuses previous search information. D* Lite is

usually faster than A* on stationary targets [95]. Moreover, it has been extended to a Multi-Objective Path-Based D* Lite (MOPBD*) [103] algorithm that finds multiple shortest-path solutions in a dynamic setting where edge cost can change.

It was believed that the incremental search algorithm Fringe-Retrieving A* (FRA*) [104] was the fastest solution for moving target search in the known environment. It solved the problems only on two-dimensional grids, which is impractical in robotics, such as Unmanned Ground Vehicles (UGVs). As a result, Generalised Fringe-Retrieving A* (G-FRA*) [105] was formed to solve UGV's navigation problems on a grid with motion constraints (UGV's location *(x,y)* and orientations (θ)). It finds the path with minimal cost from the UGV's current state to the target's state, and when the target changes its directions, it reuses the previous search tree to speed up the current search.

The Moving Target D* Lite algorithm [80] is the extension of the previously described D* Lite algorithm. The new algorithm is used in dynamic environments where obstacles appear and disappear and a target moves. Targets change their position to escape pursuers to pre-selected locations and the chase continues until caught or ends in a deadlock. Meanwhile, D* Lite does not perform as fast when the problem includes moving targets, therefore, the algorithm inherits the same principle of G-FRA*. However, the G-FRA* algorithm cannot perform in dynamic environments. Moving Target D* Lite constantly determines a low-cost path from the agent's current state to the target's state even though the environment changes obstacle positions and the target moves to a new position. It is considered to be the fastest incremental search algorithm in dynamic environments to solve moving target search problems.

To have more realistic and natural movements, the Field D* algorithm [106] was produced, which is a variation of D* Lite. It smoothens the path between the grid points, and the agents can move at any angle, not necessarily 45° or 90°. Field D* creates a path where each state can be entered and exited in any position, not only the corners. Low-cost paths are efficiently generated that omit unnecessary turning without reducing the performance [107].

Anytime algorithms such as Anytime A* [108], Anytime Repairing A*

(ARA*) [109, 110], Anytime Dynamic A* [111] and Anytime Weighted A* [112] are designed to give not the best but feasible solution. They are believed to give better results with more time for path calculations. Within the given computation time, the result converges, and its quality will not be refined anymore. Solution quality or execution time has been their main trade-off [106].

To enhance the search, the Incremental Anytime Repairing A* (I-ARA*) [113] algorithm is built on ARA* and also uses incremental search that is the same as G-FRA*. It is the algorithm that is applied to moving targets which are further improved using Compressed Path Databases (CPD) [114]. CPD stores pre-computed information of All-Pair Shortest Path (APSP) [115] in a compressed form to reduce the memory requirement during the runtime. This improved algorithm, Moving Target Search with Compressed Paths (MtsCopa) [116], with CPD's optimal and the shortest paths finds better results and outruns I-ARA* in known or partially-known environments [117]. Nevertheless, to propose a different solution, a Moving Target Search with Subgoal Graphs (MTSub) [118] algorithm is presented that is faster in finding paths than G-FRA* and quicker in the processing phase than MtsCopa by utilising the search with abstraction on small or large environments.

Series of Adaptive Algorithms. The incremental version of the A* algorithm is Adaptive A* [30] which solves the heuristic problems by updating the cost. Adaptive A* is called Lazy Adaptive Moving-Target Adaptive A* [94] as no effort is wasted. It updates the heuristic value (h-values) at the time of search when the search is needed. This makes it faster compared to A*. It can only do this because it remembers information during A* searches, such as distance from start to goal, and reuses this when future searches are needed to calculate the h-value state. Adaptive A* is simple to understand and implement.

Adaptive A* has been adapted to execute heuristic searches in real-time [94]. Real-Time Adaptive A* (RTAA*) [119] is another version of Learning Real-Time A*. The heuristics get updated after A* searches for these two algorithms but each one does it differently. The RTAA* can quickly update heuristics in a detailed way for those positions that are adjacent to the agent. It can select a low-cost path in a limited period of time in each search episode. RTAA* plans forward but cannot handle dynamic obstacles [1].

MT-Adaptive A* [95] is another modification of Adaptive A* where it also corrects the heuristic values to maintain consistency if the goal state changes. In many cases, it is faster than repeated A* searches and D* Lite. MT-Adaptive A* can search two ways, from the agent's current state to the target state and vice versa.

Generalised Adaptive A* (GAA*) [105] is another model of Adaptive A*. GAA* quickens the current search by updating h-values from previous searches and not by reusing them. Adaptive A* finds the shortest path for actions with increasing cost but no promise for decreasing action costs in the state spaces, which means it limits its relevance and h-values need to be corrected. The GAA* does not mind action cost increase or decrease but still finds the shortest path in state space [94].

The Adaptive A* algorithm is extended to the Multi-Target Adaptive A* [120] algorithm by including multiple targets in its new framework, which is a maze environment. An agent is required to reach one target or multiple targets (experiments conducted up to fifteen targets). A possible solution could be to compute each target for optimal results, which is inefficient for large numbers. It uses *OR settings*, where the agent must find the shortest path to the closest target and *AND settings* for the agent to find the shortest path for all targets. An optimal path is always achieved with all methods of *AND settings*.

Tree Adaptive A* (Tree-AA*) [121] is a generalisation of Path Adaptive A* (Path-AA*) [122] which generalises Adaptive A*. Path-AA* is the first search algorithm that is combined by the two types of incremental heuristic searches. The first type makes heuristic values of the current A* search more informed in order to quicken future A* searches, for example, Adaptive A*, GAA*. The second type reuses the previous search tree instead of calculating from scratch, for example, the D* or D* Lite algorithms. Path-AA* uses a termination strategy, it stops search early if all h-values states are equal to their goal costs. Tree-AA* finds a cost-minimal path from the current location to the goal destination and applies it to path planning with no obstacles. A novel feature of Tree-AA* is to reuse the search tree by forwarding the A* search.

A real-time Time-Bounded Adaptive A* (TBAA*) search algorithm is integrated with Adaptive A* to minimise goal-achieving time within given time

intervals [123]. It has been proved that it reaches the destination with a cost-minimal path or efficiently detects that there is no existing path. The TBAA* needs fewer or almost the same time intervals compared to the state-of-art algorithms in partially-known or unknown terrains.

Multipath Generalized Adaptive A* (MPGAA*) [124] is a simple but powerful algorithm that utilises more previous A* searches. It is theoretically the same as GAA* but uses more information from previous A* searches and is easier to understand than D* Lite. Reusing previous A* searches is also a feature in Multipath Adaptive A* (MPAA*) [125], but this algorithm cannot be used in dynamic environments. MPAA* was an inspiration for MPGAA*. The experimental evaluation demonstrates that MPGAA* is a superior algorithm compared to D* Lite in relation to memory usage and for a goal-directed route in the dynamic environment.

2.2.3 Real-Time Algorithms

Unlike offline search algorithms, such as the A* algorithm where the entire solution path is computed before the first action is executed [126], real-time algorithms restrict the search to a small part of the environment and perform sufficient computation to determine the first action from the current state, that is incorporating planning and execution independent of the problem size [127, 128]. Real-time algorithms do not plan the whole path, instead with a given lookahead (a maximum number of states to expand) provide smaller searches that result in quicker times but higher pathfinding costs [119]. Moreover, any size environment can be adapted and larger areas compute faster planning in comparison to IHA which is slower with the environment size increase [47]. Although finding the optimal solution is not guaranteed, suboptimal solutions for real-time algorithms are faster than offline algorithms [127].

Real-time heuristic search computes from the current position of the agent to the adjacent neighbouring unblocked position within the given lookahead and takes an action to the lowest value determined by the heuristics. This process is repeated until the agent reaches the target position or the target becomes

unreachable [119]. Real-time heuristic search holds information on visited or learned positions throughout the process in the hash table, where planning gets updated using asynchronous dynamic programming techniques and thus efficiently avoids cycling [93]. An unknown target's location makes it difficult for real-time algorithms, so they need to be fast, respond quickly to the changes in terrain and move smoothly [47]. It is also realised that most of the algorithms use two-dimensional grids for simplicity purposes [129].

One of the real-time variations of the A* is Real-Time A* (RTA*) [35]. RTA* guides itself towards the goal using heuristic values. It uses a hash table to keep previous search results. The agent's move depends on the value in the hash table, which determines the closest state to the goal. The second-best value is written to the hash table too, where it tracks back the path when it seems promising to avoid dead ends and infinitive loops (heuristic depression) [93]. In contrast to RTA*, Learning Real-Time A* (LRTA*) [35] writes only the best heuristic value into the hash table. After repeated searches, as the name suggests, the heuristic value in the table will guarantee convergence to its exact value. Both RTA* and LRTA* have been one of the first real-time algorithms and have been an inspiration to many approaches. There are four different approaches introduced [130] to avoid impassable paths, deadlocks, and escape a stationary position without forward movement, heuristic depression. Among these solutions, the daRTAA* algorithm displayed better performance and provided optimal results.

Moving Target Search (MTS) [27] is the first real-time search algorithm to solve a problem in a dynamic environment. It is the generalisation of LRTA* with a moving target. It gets the heuristic values for each target and keeps all location values in a matrix. In the progress of the search, the heuristic values are updated, and the accuracy improves [22]. To enhance its efficiency, commitment to goals and deliberation were included [131].

Real-Time Moving Target Evaluation Search (MTES) [132] was developed for dynamic and partially-known environments. It is capable of estimating the distance to the target with obstacle consideration. MTES eliminates closed directions in real-time to move the agent to the static or dynamic target avoiding obstacles that were found using virtual rays. Virtual rays form the border of obstacles, and the resolution mechanism chooses the single moving

direction [133].

Abstraction MTS (A-MTS) [36] uses a heuristic array table. It has a 2-level search and uses abstracted state space for calculating the path towards the target. Each searched cell is numbered and stored in a 2D array labelled Abstract. These numbers represent a *group* and contain a *group head*. The *group head* measures nodes' distance within the same *group*. A-MTS has an *abstraction move list* and a *real move list*, where both lists contain a sequence of movements. If the target gets detected, then it is checked if it is within the group. If so, the *abstraction move list* produces a list of movements and the *real move list* captures the target. To avoid delays and quicken the movements, a *real move list* is only produced when the previous path has been traversed. Even though the response time is low, it is not optimal as the generated path is abstract.

The Fuzzy MTS (F-MTS) [36] algorithm is based on forecasting the future position of the target by identifying the moving direction. If the target is not discovered, then its last known position is used. There are two parameters: *probability* – which uses the equal distribution of movements and *pattern* - uneven distribution of movements, for example, the maze map environments. These are used for locating the target's possible future position. Prediction performance in large environments is unsatisfactory.

2.3 Multi-Agent Algorithms

Pathfinding search algorithms have been generally employed as a single-agent task; however, recent research is extensively focused on multi-agent algorithms to develop solutions that coordinate multiple agents to work as a team to satisfy the intended goal [134], either in known [135], partially-known [136, 137] or unknown dynamic environments [122, 138]. Even though multi-agent pathfinding algorithms are difficult, complex, and challenging, with issues such as communication, coordination, path planning, exhaustive computation process and distributing the tasks, there are still several challenges that need to be addressed.

This section reviews several existing algorithms that are used in the multi-agent scenarios in the literature. Each algorithm is characterised by the given

problem, some of which were discussed in Chapter 1, and applied with multiple target or goal destinations in multi-agent pathfinding frameworks. Additionally, the algorithms differ in the target assignment and the scenarios they are applied. None of the algorithms can be employed in every scenario [139] or solve every possible problem [57]. Thus, this section classifies these algorithms according to the number of targets and whether pursuing agents approach stationary (static) or fleeing (moving) targets. Table 2.1 illustrates multi-agent pathfinding algorithms and two new algorithms that contributed to this thesis. The review of these algorithms is discussed in the following sections.

2.3.1 Pursuers and Single Target

The Multiple Agents Moving Target (MAMT) [35] algorithm, is a decentralised approach where each agent computes its path and moves towards the targets independently from each other, however, the agents can share information among themselves. They see what is in front of them, either agent or target, and there are obstacles, too. The target's or other agents' location information is unknown and depends on clearly seeing, assumptions, or communication among the agents but not the target. The agent uses a *believe set* which depends on the grid map, knowledge of the target and its last known place. The *believe set* uses filters within a certain time to increase effectiveness and speed up the process. If the target is invisible, the coordination and pursuit work well, otherwise, the collaboration is ignored, and the shortest path is searched independently.

Multiple agents cooperatively outsmart the target. They do not try to outrun but coordinate with each other to find the solution by splitting up and surrounding the target. An effective strategy requires time balance for agents in planning and coordination. To make it practical, the Cover with Risk and Abstraction (CRA) algorithm is introduced in [60]. It uses a *cover set* to consider the position and speed of other agents and to catch the target faster. The agents aim to collectively reduce the target's moving choices and not to find the shortest path like many existing algorithms do. CRA uses *abstraction* to increase the efficiency and risk to balance between keeping the target encircled and "rush-in" to capture. CRA is slow in computing time and

Table 2.1: The multi-agent algorithms are categorised in relation to the targets and their brief description. The new contributions are at the bottom of the table.

Multi-Agent Algorithm	Target Relation	A Brief Description
MAMT	Pursuing agents aim to find a path towards a single static target.	Approach the target without knowing its place but use *believe set* of possible positions of the target.
LRA*		Each agent computes its route independently without considering other agents which can revisit the same position and is time-consuming.
e*MIP*		Coordinating robots with minimum travel costs and sharing maximum information are experimented with in a lake.
ADP		The number of agents is limited to adapting a strategic behaviour for coordination, but each action is committed independently.
MLA*		It is for pickup and delivery problems in warehouses for multiple agents towards static goal positions with collision-free paths.
MM*		A meeting point for multiple agents can be a challenge and with assignment strategies an optimal distance is arranged.
PRA*	Although it is a single target, it can move away from pursuers.	The cost of the search is reduced by generating a path on an abstract level of the search space.
CRA		Instead of rush-in and attacking the target, the agents split up and surround the target using cover heuristics.
HCA*	These algorithms aim towards multiple targets that are stationary during the search.	A hierarchy of abstractions is employed, and previously computed paths are stored in a reservation table to avoid future collisions.
WHCA*		A window space is created for agents to cooperate and in a short period of time dynamically prioritise the search in the current *window* .
FAR		Another approach is to use road networks where the flow of a direction is annotated as a one-way road on a grid map.
OD		One action is considered for one agent at a time and continues to the next agent until all agents know their next move. It is optimal, complete but expensive to run.
ID		The problem is divided into sub-problems and merges agents into a group. The solutions are found independently without any conflicts or sacrificing optimality.
CBS		A centralised approach with high and low levels of searches, however, its searches are single-agent. The conflicts are resolved for pairs of agents.
MA-CBS		It is a generalisation of CBS where it merges the conflicting agents into one compound, which then is processed to find a path at the lower level.
CBM		Agents are split into teams with the same number of targets and plan their path with no collision by minimising the makespan in the known environments.
DiMPP		Targets are allocated in advance to the agents and each computed path uses a priority assignment. The longer the path, the higher the priority.
TP		One solution for warehouses where agents plan one after another. The tasks are assigned using the service time efficiency with the lowest value with no collision.
TPTS		All tasks are assigned, and agents can request to swap tasks if has a smaller cost before reaching the location.
TA-Hybrid		Another solution for agents in warehouse problems (pickup and delivery), is where task planning, path planning and deadlock avoidance are improved. These are offline algorithms which can scale to a larger number of agents and tasks.
TA-Prioritized		
STMTA*	A novel contribution to the thesis where pursuers chase multiple moving targets.	It uses assignment strategies and the A* algorithm to surround moving targets.
CDMTA*		Cover heuristics is the main feature as well as assignment strategies. The agents use risk analysis either to rush-in method or to trap moving targets.

indecisive in tie-breaking when each agent's move has equal cover values if the whole area is considered. This algorithm is thoroughly discussed in Section 7.2.

The video games industry uses the Local Repair A* (LRA*) [32] algorithm widely. Each agent calculates its own route independently from other agents [140]. LRA* does not consider other agents while planning the route and may conflict with other agents' directions. If it happens, the agent recalculates the route to find another way. This brings revisiting the same location repeatedly and takes time for rerouting. It has difficulty coping in challenging environments and the planning time increases with the problem size [128].

Partial-Refinement A* (PRA*) [34] is an algorithm that reduces the cost of search by generating a path on an abstract level of the search space. These abstracted spaces (graphs) are built from the grid map [66]. The abstract level is selected dynamically. The A* algorithm is then used to execute a search with sub-goals on the abstract graph. The abstract path creates a corridor of states in the actual search space, through which the optimal path is found. The PRA* algorithm has often been used in the literature and it is a widely used approach with variations that have been described with different search techniques [141]. PRA* assigns a target with the closest distance, and not all targets might be chased, as some targets could be positioned at a further distance than others for all pursuers. It has been implemented in multi-agent scenarios [60] and was successfully adapted in the Dragon Age: Origins video game [34].

One way to find solutions for multi-agents is adapting strategic behaviour among single agents that are set to limit the number of agents to coordinate but independently commit their actions [142, 143]. Agents construct their paths on the individual planning graph while checking their actions on the public planning graph to avoid the path invalidation of other agents [144]. Each path is extracted and checked until all agents reach their target destinations [145]. Furthermore, the problem got formulated to Agent Decomposition Planner (ADP) [146] algorithm where planning gets automatically decomposed prior to the actual search with the help of heuristics. The ADP is an offline (must find the whole path even before committing to take the initial action), cooperative, totally centralised, and complete algorithm in which methods find solutions and effective paths for independent agents. However, because of the heuristic approach, the whole search

space can get explored [69].

Several agents are tasked to find a collision-free path to the static goal positions in the multi-agent pickup and delivery problems. Agents are allowed to move from the starting position to the pickup location, wait and then continue to the final location. A task to pick up from a location and deliver to a goal destination is a specific multi-goal MAPF problem that is referred to as a Multi-Agent Pickup and Delivery (MAPD) [147] problem. This process requires multiple paths and involves planning for multiple agents [148]. The Multi-Label A* (MLA*) [149] algorithm is able to provide a solution by computing multiple paths by using the A* algorithm and centralised heuristic value (h-value).

First, MLA* finds the shortest path for each agent and each path has a label of 1 or 2, where 1 is the distance from the initial position to the pickup location, and 2 is the distance from the pickup location to the delivery destination. Second, the assignment strategy uses heuristics to assign tasks to the agents using their h-values that are sorted in increasing order. The agent's h-value at the initial position is the sum of heuristic values from the current location to the pickup and from there till the delivery location. But, if the agent is already at the pickup location, then the h-value is from the pickup location to the delivery location. All successful assignments use the path found by MLA*. Although this modification is more complex with multiple goals, it allows for better decisions to assign tasks to the agents and find paths with shorter routes.

For real-life situations where multiple agents need to gather and choose a meeting point among all possible destinations, Multi-Agent Meeting (MAM) [59] arranges an optimal meeting point with the shortest paths from the starting positions. The distance towards the meeting position is minimised using two different costs, firstly the Sum-Of-Costs (SOC) and secondly, the Maximum Distance Cost (makespan). The solution for these problems is overcome with the Multi-Directional Meet in the Middle (MM*) algorithm that uses the best-first search method when finding the middle meeting point for several starting locations. MM* is the generalisation of the Meet in the Middle (MM) [150, 151] algorithm that is a bidirectional heuristic search guaranteed to meet in the middle. MM* with a unique priority function for SOC cost and makespan cost, the MM* algorithm promises an optimal path for MAM problems. Despite

the solution being found for the first SOC priority function, the search continues to find if any other solutions are available from another priority function, which in this case is makespan.

Robotics is one of the core applications used for search algorithms and the Efficient Path Planning (e*MIP*) [152] algorithm is one of them. It uses coordination among the robots with resource constraints, for example, path length or energy capacity. Robots monitor the environment to obtain maximum shared information. This mutual information analyses the most informative paths. Each robot's path is associated with a sum of sensing cost and travelling cost. The task is to find a path with a minimum cost and maximum information using joint effort. This is another useful method for assigning targets to the robots. The experiments are set on robotic boats in a lake.

2.3.2 Pursuers and Multiple Targets

Research has been conducted to investigate solutions for multiple pursuing agents in multi-target environments that use road networks as a different approach. The Flow Annotation Replanning (FAR) algorithm is based on annotating flow direction on the grid map, which was inspired by two-way roads [31]. The whole map is designed with a one-way direction for each column or row, avoiding head-to-head collisions. Each agent runs the A* search independently. The *Open* and *Closed* list is released from the memory and caches the computed path when it completes the search. The deadlocks happen quite often, block the planned paths and agents wait until the agents in the front move. Agents have solved it locally instead of replanning the whole step.

A more suitable method in a dynamic environment is to use a hierarchy that computes the abstract distances on demand. Hierarchical Cooperative A* (HCA*) [32] employs a simple hierarchy of abstractions which is a simple two-dimensional map. It creates a reservation table, previously computed and reserved paths, for future replanning to avoid collisions [86]. HCA* calculates the shortest path to the goal with no problem unless pushed away by other agents on the same path. When this happens, HCA* will lose its computed shortest path and recalculate until the goal is reached. HCA* solutions are

shorter compared to LRA*, however, HCA* performs worse when abstract distances increase.

The solution of Windowed Hierarchical Cooperative A* (WHCA*) [32] is to create a search in window space where agents cooperate among themselves and calculate the complete path in a large, three-dimensional state space. For a robust solution, each agent is dynamically prioritised with the highest search preference for a short period of time in the current *window*. Each *window* specifies a limited fixed depth in the cooperative search for the agent to move in the right direction. The agent calculates a partial path within the *window* space and moves steadily towards the goal. With improved heuristics, WHCA* is more concise, powerful, and efficient. Agents can distribute the processing time across each other. WHCA* is better than LRA* in finding short and successful routes with fewer cycles.

An algorithm can be complete if it finds a solution to the existing multi-agent pathfinding problem. An algorithm, such as a fully Distributed Multi-agent Path Planning (DMAPP) [153] algorithm, is incomplete as it fails to find a path although it exists. This occurs when DMAPP is prevented from finding a solution because of the priority order. However, it has been enhanced and proposes a complete Distributed Multi-agent Path Planning (DiMPP) [62] algorithm for multi-agent path planning. Agents may not have complete knowledge of all existing agents and the targets are allocated in advance. DiMPP computes a path independently for each agent, if a conflict occurs, it is resolved using priority assignment to the agents. The priority is decided based on the path length of individual agents. The longer the path, the higher the priority. DiMPP works in path planning, distributed decision-making, and plan restructuring phases.

A real-world domain such as an automated warehouse [154] is an environment where agents navigate between locations and interact with executing tasks. This is a MAPD instance, one of the MAPF problems [147] where agents constantly need to assign new assignments. The assignment is a task, that is to move from the initial position to the mid-point and from the mid-point to the final goal destination, where an agent picks up at a certain location and delivers to the allocated point. Tasks can appear in the application at any time and only an

agent that is not allocated to any tasks yet is assigned to one task. This agent plans a collision-free path using the A* algorithm from its current position, and then moves to the pickup position and delivers the task.

The recent research on MAPD has improved task planning, path planning and deadlock avoidance by introducing two offline search algorithms, Task Assignment and Prioritized (TA-Prioritized) and Task Assignment and Hybrid (TA-Hybrid) [155], that have experimentally proven to perform better and scale to a large number of agents and tasks. Additionally, a closely related solution and a different challenge to warehouse problems is the Multi-Goal Multi-Agent Path-Finding (MG-MAPF) [156] problem where each agent gets multiple target destinations assigned which needs to be visited at least once for successful output. Despite, the agents can navigate the high-quality path with no collision in automated warehouse environments, the challenge still exists in the implementation where agents, for instance, robots, are unable to adhere to the path accurately. Thus, the Action Dependency Graph (ADG) [157] framework proposes a solution that guarantees robustness. Furthermore, the authors in combination with the Rolling-Horizon Collision-Resolution (RHCR) [158], which has a limited planning timeframe, improved the framework to resolve the MAPF problems with a continuous flow of online tasks.

Two decoupled MAPD algorithms, Token Passing (TP) and Token Passing with Task Swap (TPTS) are introduced in [147]. TP is a simple, distributed algorithm that uses token passing, which stores data (task sets, assignments, and the path of agents) in a shared memory block. All non-assigned tasks are kept in a task set, and the task is assigned using the service time efficiency. The service time is the average of time steps required to complete the tasks. Each agent's path is planned one after another. At each time step, an agent chooses a token with the smallest h-value and no collision path at the pickup and delivery locations. An agent with no assignment stays at the last position of the path.

Unlike TP, the TPTS contains all tasks that have not been executed previously, not just non-assigned tasks. The difference is that an agent can assign the task that was previously assigned to another agent and can request a token swap if the task has not reached the pickup location and has a smaller cost. The TPTS algorithm is more effective in comparison to TP but scalability

is not guaranteed for many agents as TP does. TP's runtime is measured faster, although the computation time needs improvements [149]. These two algorithms operate in a known environment and are tested on a simulated warehouse-modelled map [23].

Conflict Based Search (CBS) [159] is the algorithm for MAPF problems that promises optimal solutions at the expense of computation by using a centralised approach, however, all pathfinding searches are single-agent which is similar to the decoupled approach [160]. CBS uses both high-level and low-level searches. At the high level, the search is structured to use the best-first search, and the arising conflicts need to be resolved for pairs of agents. The CBS algorithm uses a constraint tree (CT), with each node having constraints on time and location. At the low level, the A* based search is run only for the single agent, while disregarding the other agents, to find the optimal path under a set of constraints that are established at the high-level search [161]. CBS outperforms other optimal multi-agent pathfinding algorithms in the corridors and the areas with many obstacles, additionally, in the scenarios with many bottlenecks that have fewer conflicts. Also, CBS performs better in comparison to large open spaces that have many conflicts [33].

Although CBS solves MAPF problems optimally, still the worst-case performance needs to be reduced and therefore, CBS has been generalised into a new algorithm called Meta-Agent CBS (MA-CBS) [33]. This approach has been generalised to merge the conflicting agents into one compound as a meta-agent if the predefined conflict bound is met. Once the conflicting agents are merged, then this meta-agent is processed to find a path at the lower level. The complexity of the MAPF problem is exponential with the number of agents and maintaining a larger number of agents is a problem. To mitigate this issue and increase the performance, the conflicting agents are merged if the conflict bound is levelled in MA-CBS and this merging is always true for the Independence Detection (ID) [86] algorithm, unlike the CBS algorithm, which never merges agents.

It is possible to use the extensions of the A* algorithm such as Operator Decomposition (OD) and ID [86]. OD is an approach where branching factors are reduced for search algorithms. At each time step, OD considers possible

actions for one agent at a time, and with a fixed order continues to assign the action to the next agent. This process is called an *intermediate* state until all agents know their next move, which is a *full* state. If no actions, it is a *standard* state. This pruning method reduces the expansion of surplus states, and the solution is found relating to the single agent's actions [51]. For instance, if each agent has nine possible actions, including waiting with eight diagonal moves, then the branching factor is reduced from $(9^n)d$ to $9d$ while d is the depth of generated states [162]. Despite being optimal and complete, OD has expensive runtime, and paths can be conflicting if the runs are terminated prematurely [63].

The ID is another extension of the A* algorithm that divides the problem into smaller sub-problems identifying an independent group of agents. ID repeatedly merges agents into a group and finds paths using OD and this process continues until there is no conflict between the agents and they can independently find solutions without any conflicts and without sacrificing optimality [63]. Provided that the complexity of MAPF problems grows exponentially with the number of agents, the search of the largest group dominates the running time of solving the problem, though avoiding unnecessary merges can significantly increase performance [46]. The cost is denoted by the Sum of Individual Costs (SIC) heuristic for these algorithms [159].

The combined Target-Assignment and Path-Finding (TAPF) [163] problem is a different kind of MAPF problem. The number of agents is equal to the number of targets, and the agents are aimed first to assign all targets and then plan their path with no collision by minimising the makespan in the known environments. The agents are split into teams and each team is given the same number of targets to match the number of agents in the team. It is allowed for each agent within its team to swap the assigned targets but the agents from the different teams are not allowed to swap the targets with other teams. The solution for this problem is addressed with a Conflict-Based Min-Cost-Flow (CBM) [15, 163] algorithm that solves TAPF instances using anonymous (swappable target assignments) and non-anonymous (pre-determined target assignments) multi-agent pathfinding algorithms.

CBM is a hierarchical algorithm that uses a non-anonymous algorithm such

as CBS [161] on a high level and an anonymous minimum cost maximum flow algorithm [164] on a low level. At the high level, individual agents in each team are merged into one meta-agent, which uses the CBS algorithm to avoid collisions among these meta-agents and performs the best-first search on a binary tree. Each node on the tree has a set of constraints and paths for agents to observe the constraints. Makespan is used to find the minimum collision-free cost path. At the low level, CBM runs the minimum cost maximum flow algorithm over the time-expanded network. All agents in the same team are assigned to unique targets and no-collision paths are planned to comply with the constraints introduced on the high level. Moreover, edge weights are introduced on the low level to avoid possible collisions among meta-agents. The CBM algorithm has theoretically proven to be complete, correct, and optimal [163]. Furthermore, there are many improvements and enhancements introduced to the CBS algorithm [165, 166, 167] and different solutions explored for multi-agent pathfinding [168, 169].

2.4 Target Algorithms

This section introduces several existing target algorithms in the literature. The following is a brief description of each algorithm. Although there is plenty of research in the literature emphasising algorithms for pursuing agents, there are few studies that are conducted on algorithms for mobile, moving targets. The A* algorithm is a classic example that is implemented as an algorithm for many pursuing agents, as well as target algorithms [170].

TrailMax [78] is an intelligent algorithm that is based on a strategy. It generates a path for a target considering the pursuing agent's possible moves, i.e., it efficiently computes possible routes by expanding its current and adjacent neighbouring nodes and the agent's nodes simultaneously. The TrailMax algorithm aims to make the targets stay longer by maximising the capture time. The players can move on the map; thus, the target computes an action at every time step with new updated information about the players. It is for one-to-one player scenarios. The algorithm works as follows. To compute a path, an escape route that maximises its distance away from the agent, it checks the best cost of the neighbouring states against the pursuer's costs and expands nodes

accordingly. The algorithm expands nodes that are not yet expanded and not already occupied in the target closed list and not in the pursuer closed list. The node with the best cost is added to the target's closed list, which would generate the path afterwards. The first element in the path is an action for a target to take. This procedure is repeated from scratch at every time step. It is a state-of-the-art target strategy algorithm that performs the best against pursuing agents, aiming to make the targets less catchable or more difficult to capture [53].

When Minimax [171] is used as the target algorithm, it runs an adversarial search that alternates moves between the pursuers and the target. When the pursuing agent gets closer to the target state, then the target distances itself from the pursuing agent's state. To make the algorithm faster, Minimax is run with an alpha-beta pruning search, where alpha (α) and beta (β) are constantly updated to avoid the exploration of sub-optimal branches. The used depth is 5, i.e., the outcomes after at most 5 moves of each party are considered.

Dynamic Abstract Minimax (DAM) [171] is a target algorithm that finds a relevant state on the map environment and directs the target using Minimax with alpha-beta pruning in an abstract space. There is a hierarchy of abstractions. Higher levels might not provide enough information about the map and lose important details, such as an agent at the close by, and fine abstract levels might be very detailed and increase the computation costs. The search starts on the highest level of abstraction, an abstract space created from the original space. The Minimax algorithm runs a search at the highest level of abstract space and continues to the next low level of abstraction. It stops at the level where the target can avoid the capture. Then, on this level of abstraction, if a path exists, an escape route is computed using the PRA* algorithm (see Section 2.3.1). If the target cannot escape and there is no available move to avoid the capture on the selected abstract space, then the level of abstraction is decreased, and the whole process repeats until the target can successfully run away from being caught. The used depth is 5.

Another algorithm for targets is Simple Flee (SF) [60], which can be used to escape from the pursuing agents to the predefined states on the map. The SF algorithm works as follows. At the beginning of the search, the target identifies

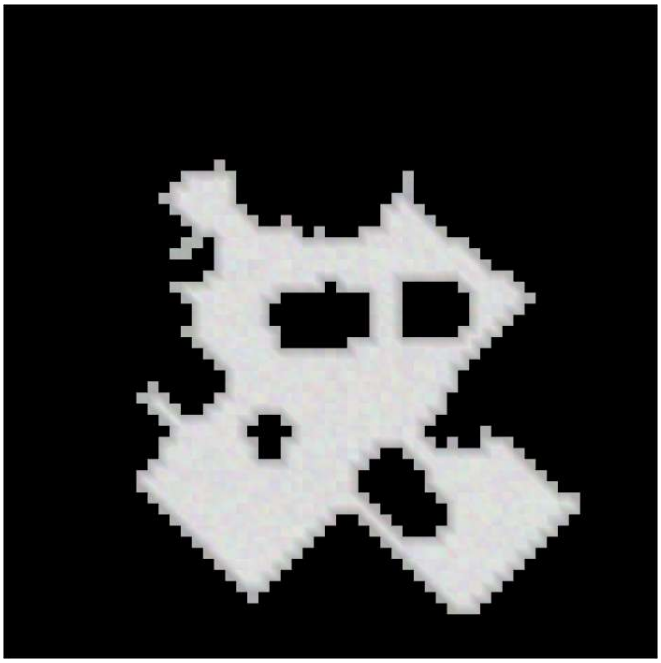


Figure 2.1: A sample AR0417SR map from Baldur's Gate video game used in the experiments.

some random locations on the map. When the target starts moving, it navigates to the furthest position, away from the pursuers, on the map among a set of random locations that are set before the simulation starts. During the escape, to disorient the pursuing agents, such as incremental heuristic algorithms, D* Lite [28] and MT-Adaptive A* [95], that can search from the target's state, SF can check if the selected location is still the furthest and possibly change direction after a pre-set number of steps, otherwise, it keeps moving to avoid capture. The number of locations on the map and the number of steps before the change are the parameters of the algorithm.

Greedy [172] is the standard algorithm that repeatedly makes the best local optimal choices that, in the hope, would lead to global solutions. This is a simple and fast approach to solving a problem that uses sub-optimal and easily computed heuristics. Greedy runs a cumulative Manhattan distance of maximising the gap towards the pursuers. It evaluates its options and moves to that state. Once it is at that point, it will stay until being captured, if any other maximum states are not available [60].

2.5 Design and Structure of the Experiments

Initially, this section introduces a precise problem characterisation of the multi-agents and multiple moving targets that are referenced throughout the thesis. Then, it follows by describing three existing assignment strategy algorithms that are considered as a basis for studying other different criteria. Finally, an experimental problem setting is presented which is referred to later in the following Chapters 3-7.

2.5.1 Problem Formulation and Description

An instance of the PAMT problem consists of m number of pursuing agents $P = \{p_1, p_2 \dots p_m\}$ and n number of targets $T = \{t_1, t_2 \dots t_n\}$, where $p \geqslant t$ on a finite, known undirected graph $G = (V, E)$. V denotes a set of vertices and E represents the set of edges between vertices. Vertices are the *states*, edges are *actions* and edge weights represent travel *distances*. The total cost of *distances* travelled by a pursuer from the initial state to the target state is called the *pathfinding cost*, i.e., the travelled total cost. A *heuristic function* (*distance heuristic*) estimates the *pathfinding cost*, and a pursuer then assesses the outcomes, evaluating the solutions with the lowest value as the most promising in contrast to the *cover heuristic* (covered area) with the maximum value [173].

The problem incorporates multiple players, pursuing agents and targets, and it is studied in a simulation environment. A scenario considers the players navigating to any adjacent vacant state as well as the pursuing agents traversing towards the target in a known or partially-known environment where the initial location of the targets and the obstacles are known. The communication is free, information is shared instantly and every player's state is known to all players, both pursuing agents and targets [174]. The initial states of the agents do not need to be unique, and agents can start from the same state or during the test runs can occupy the same state [21, 175]. Similarly, agents can share the same target if it is an optimal combination provided by the assignment strategy algorithm (see Section 2.5.2 and 4.2).

Coordination among the agents is achieved through the initial assignment of pursuing agents to targets. Although, the complete knowledge of the environment

is known, no prior information is given about the number of targets, their escaping route or destination. Targets are free to move to any point on the map and escape from the approaching pursuers, and no pursuer has control over their movement directions.

The environmental changes do not occur at the starting state, but at the next time step, the states of the targets can change; therefore, each agent must observe the new positions of the targets or whether they are caught. The pursuing agents must adjust to the dynamic changes and find a path from their current state to the state of the assigned target. Even though the path is computed on the search space, the only action is executed locally in the neighbouring states.

The success of the problem is when the target is reached and occupied in the same state by one of the pursuers. The success of a team when all targets are caught. However, the success of the target when it avoids the capture and manages to escape the approaching pursuers. When the target is not reached and the chase continues, the pursuers alternate two actions, planning and execution. The execution of an action, that is a moving action, interleaves with planning and planning cannot proceed during the moving. This method enables the simulation of the asynchronous chase scenarios that can also be found in the literature [22, 57, 176, 177].

The problem applies to any graph, however, empirical studies use grid-type graphs. Furthermore, in the empirical testbeds, which are commercial video game maps from Baldur's Gate [178], the maps are set on grid-based states with four connected neighbours, where each state is denoted as vacant or blocked by an obstacle. An agent can move orthogonally to the adjacent empty state or stay in the same state, i.e., a wait action. Although different speeds could be applied to each pursuer or target, they all have the same speed. The environment is deterministic as it is possible to infer the agent's next action based on the given current state and current action. Time is discretised into time steps.

2.5.2 Existing Criteria for Assignments

Many algorithms use the shortest distance criteria for assigning agents to pursue the targets. Others consider all or most possible assignments and use criteria

such as the sum of distances or the maximum distance (makespan) to select the optimal combination of assignments. Different or similar methods have been used for these cost criteria, including sum-of-costs and makespan [59], flowtime and makespan [179], service time and makespan [149] and the sum of path lengths and makespan [14], or the sum of the individual path costs (flowtime) and the maximum of the individual path costs (makespan) [180].

The work in this study builds on the criteria introduced by Xie, Botea and Kishimoto [53] and their research uses Summation-cost, Makespan-cost and Mixed-cost criteria. The Mixed-cost criterion is optimised by merging the other two criteria and it is the best overall in their study [53]. These three target assignment approaches, which consider a distance to evaluate the cost, are described in this section.

2.5.2.1 Summation-cost

A cost is the number of movements that measures the distance from a starting location to the goal location. Every agent, at the current position, estimates a distance cost towards the goal and not the future positions. Summation-cost is the cumulative distance sum of each agent [59]. Depending on the number of targets present in the environment, the pursuer computes a Manhattan distance from its current position towards the targets. Each combination cumulates the distance cost of each pursuer, and the Summation-cost criterion selects the combination with the lowest value.

Consider a two-dimensional game map AR0417SR (Figure 2.1) with obstacles in the black shade; Figure 2.2 is a portion of this map. There are four players present, two pursuing agents (A_1, A_2) and two targets (T_1, T_2). Each pursuing agent has an admissible path towards the targets; therefore, they have got a choice of which one of two targets to follow. Table 2.2 provides pre-computed individual distance costs, the *Distance* column, for each pursuing agent and the table contains *DistancesSum* column, which is the sum of two distance costs within each combination. The Summation-cost criterion compares the *DistancesSum* values and assigns targets based on the lowest combination cost. Table 2.2 displays distances from Figure 2.2, where (A_1,T_1),

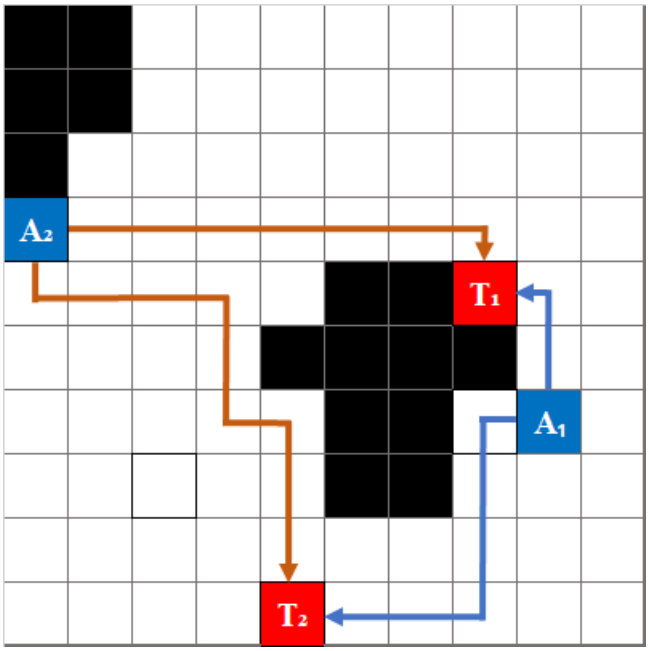


Figure 2.2: Demonstrating pursuing agents' (A_1 and A_2) possible directions towards the targets (T_1 and T_2) on the part of AR0417SR map, see Figure 2.1. Black shades are obstacles.

(A_2,T_2) has a cost of 3+10=13, which is optimal to (A_1,T_2), (A_2,T_1) with a cost of 7+8=15. In this case, Combination1 has the minimum value and it is preferred to Combination2.

2.5.2.2 Makespan-cost

This is similar to the Summation-cost criterion described above. The Makespan-

Table 2.2: The possible distance cost combinations for two pursuing agents. *DistancesSum* is used for the Summation-cost criterion and *MaxDistance* for the Makespan-cost and Mixed-cost criteria.

Combination	Agent to Target	Distance	DistancesSum	MaxDistance
1	$A_1 \rightarrow T_1$	3	13	10
	$A_2 \rightarrow T_2$	10		
2	$A_1 \rightarrow T_2$	7	15	8
	$A_2 \rightarrow T_1$	8		

cost criterion uses the maximum distance cost within the combination instead of its total [53]. The Makespan-cost has been called time steps too, as each move is equal to a one-time step. Thus, it focuses on the maximum time spent to reach the current position of the targets for all agents. This is important in many situations, for example, hot food delivery drivers might want to take their customers' orders to their destinations such that the maximum time is as low as possible.

Table 2.2 additionally shows this criterion in the column named *MaxDistance* where the combinations are compared with one another and the lowest value is selected. The assignment (A_1,T_2), (A_2,T_1) has a *MaxDistance(7,8)=8* which is optimal with respect to the makespan-cost criterion, whereas (A_1,T_1), (A_2,T_2) has a *MaxDistance(3,10)=10*. Therefore, according to these results, by selecting Combination2, which has the lowest value for the Makespan-cost criterion, the delivery drivers can have the optimal solution.

2.5.2.3 Mixed-cost

The Mixed-cost criterion uses the Makespan-cost criterion as the main component for its assignment strategy and it has been found to be the overall best criterion in some previous experiments [53]. To assign agents to the targets it takes the best value from the *MaxDistance*, i.e., the lowest value, as shown in Table 2.2. The distance cost used in the Mixed-cost criterion is the maximum time spent to reach the destination at the current position. If *MaxDistance* values are equal for both combinations, then the tie-breaker uses the *DistancesSum*.

2.5.2.4 Complexity analysis

The complexity analyses have been previously studied and they are provided for the above criteria. The problem in the Summation-cost criterion is to find the minimum total cost within the combinations and the time required to solve is $\mathcal{O}(n^3)$ [181], whereas Makespan-cost seeks to find the lowest value of the maximum cost in each combination and the solution for its time complexity is provided in $\mathcal{O}((m+n)\sqrt{n})$ steps which is $\mathcal{O}(n^{\frac{5}{2}})$ [182]. The Mixed-cost criterion employs Summation-cost as the tie-breaker, therefore, the worst-case is the same as the

Summation-cost [53].

2.5.3 Experimental Problem Settings

The experiments are adapted to a simulated gaming environment and set to run on commercial game industry maps, which are standardised benchmarked maps from Baldur's Gate video game [183]. The maps are two-dimensional, grid-based rectangular environments with four-connected grids and impassable obstacles. Commercial gaming maps are important for AI research, although they are complex and difficult in comparison with handmade or purposely-made artificial environments, these maps are significant in solving real-life problems [184] and improved to create realistic and engaging behaviour for non-human players [8]. It can be imagined as a scenario with cops (pursuing agents) and robbers (targets), where cops need to unite their forces to successfully catch robbers at a minimum cost.

The selection of experimental maps has prioritised the size, the existence of obstacles (static non-traversable cells) and the difficulty of navigation. Each map has its defined obstacles in the black shade and these obstacles shape the map. The white space is a traversable area. Although the maps are not already categorised, they are put into three different groups in Table 2.3 based on visual layouts as circle-shaped, corridors and large open spaces. The circle-shaped with resembling island obstacles in Figure 2.3(a) and those with narrow corridors in Figure 2.3(b) are the illustrated samples of Baldur's Gate video game.

To evaluate the performance of the algorithms, there are twenty-four benchmarked maps used for the experiments across all chapters and organised into groups from the same category. Each newly presented algorithm conducted experiments on circle-shaped and large open-space maps. Moreover, the environment with narrow passages and corridor-shaped maps were introduced for Chapters 4, 6 and 7. Additionally, five more purposely-made maps [60, 171] with varying navigation difficulties are included only for the experiments in Chapter 3.

Time is discretised into time steps. Only orthogonal moves are allowed where the travel directions could be horizontal (left or right) and vertical (up or down)

Table 2.3: The categorised testbeds used in the experiments for each chapter with their map identification, dimensions in nodes, and traversable states.

Map Names	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7	Height x Width (number of nodes)	Empty States to Expand
Circle-shaped							
AR0313SR		✓				64x60	946
AR0401SR				✓		59x52	1081
AR0402SR				✓		59x56	1075
AR0417SR	✓	✓			✓	54x52	833
AR0507SR			✓			54x52	739
AR0526SR				✓		80x72	833
AR0527SR	✓		✓		✓	54x52	531
AR0528SR	✓	✓				54x52	562
AR0531SR			✓			54x52	716
Corridors							
AR0016SR				✓		86x88	2103
AR0304SR		✓				86x80	1734
AR0413SR				✓		118x108	1014
AR0503SR					✓	75x68	745
AR0514SR		✓				64x64	1134
AR0712SR				✓		64x64	1163
Large Open Space							
AR0302SR		✓				86x80	1819
AR0311SR			✓			54x52	558
AR0332SR				✓	✓	59x56	973
AR0407SR	✓		✓			54x52	576
AR0508SR			✓			54x52	567
AR0509SR				✓		75x72	1503
AR0512SR			✓		✓	54x56	896
AR0607SR		✓		✓		54x60	1730
AR0707SR	✓		✓			59x56	974
Purposely-Made							
RoundTable09x09	✓					09x09	56
RoundTable39x39	✓					39x39	572
MTS1	✓					41x41	731
MTS2	✓					41x41	1277
Empty30x30	✓					30x30	900

(a) (b)

Figure 2.3: The standardised grid-based maps from Baldur's Gate video game are used and circle-shaped (a) and narrow passage corridors (b) are the samples.

with a cost of one each. This is the Manhattan distance that is the standard heuristics for rectangular grid environments [185] and Manhattan distance is very accurate [186]. Meanwhile, the diagonal heuristics are also used in [118, 125, 136] with a cost of $\sqrt{2}$ and it has been implemented for evaluation only in Chapter 4. If targets are not standing idle, then they have the same moving capabilities

Table 2.4: Experimental setup and the number of test runs for algorithms.

	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7
	STMTA*	Assignment Strategy - 1	MPTM	Assignment Strategy - 2	CDMTA*
Movement Direction	Manhattan	Diagonal	Manhattan	Manhattan	Manhattan
Number of Test Runs per Each Setup	30	30	20	20	20
Total Number of Individual Test Runs	21,600	12,600	20,480	56,700	56,000
Computer Configuration	CPU: 2.2 GHz RAM: 16 GB	CPU: 1.9 GHz RAM: 40 GB	CPU: 2.2 GHz RAM: 16 GB	CPU: 2.2 GHz RAM: 16 GB	CPU: 2.2 GHz RAM: 16 GB

as pursuers. Pursuing agents and targets have equal speed, and one action is performed at each time step. Each action is either move to the adjacent cell or stay put. Nevertheless, the approach is suitable to work in situations where there are different movement costs and speeds.

In the literature, there are many different settings outlined for the tests. For instance, in each configuration setting, which consists of a fixed or random starting position for pursuers and targets, every experimental result is averaged of the 5 [149], 10 [29, 62, 187], 20 [60, 152] or 30 [179, 188] test runs and variations may occur due to computational resource constraints [126]. The number of map environments used during the empirical experiments has been various, too. Despite the existence of 25 [189, 190], 50 [14] or 100 [66, 191] test runs, some algorithms, such as those used in warehouse settings [23, 192] or compressed path databases on static environments for point-to-point distances [114, 193], were only evaluated once for each state but with a different starting states on each individual run. The experiments in this research, as can be seen in Table 2.4, are the average of 20 or 30 test runs for each testing configuration setup. The total number of individual test runs is displayed in the same table.

To help to analyse the behaviour of the algorithms better, different sets of scenarios are conducted. Each given environment has various pursuer-to-target combinations. In single-agent scenarios one target is sufficient, and similarly, some multi-agent scenarios involve only one target. This research conducts multi-agent multiple-target experiments (the PAMT problem), which contain several combinations with 3, 4 or 5 pursuers and 2, 3 or 4 targets, where the ratio of the pursuers outnumbers the targets or is equal and no target is left unassigned. In the case of testing target algorithms in Chapter 5, these scenarios also help to study the actions and measure the performance when targets are outnumbered.

Each problem is defined by the starting position of the pursuing agents and targets. These positions provide a diverse set of scenarios where the behaviour and performance of the algorithms can be observed. Depending on the algorithm evaluated they have different sets of starting positions on each map, and they are positioned on preselected random locations in the known environment. All pursuing agents and targets have knowledge of the locations of others. The constraint is relaxed, and the pursuers are allowed to occupy the same state.

This helps in understanding how the pursuers utilise the use of surrounding and trapping the targets. Therefore, on some maps, the starting states included positioning pursuers in the same location on the map's corner, in the centre close to each other or spread out by the wall of the map. The targets are always positioned as far away on the opposite side of the map. Each chapter, Chapters 3-7, evaluate different pursuer-to-target combination and various starting positions.

The results presented evaluate the performance of algorithms and each chapter measures pathfinding cost and the success rate considering all pursuers. The pathfinding cost (the travelled total cost) is the number of steps taken before capturing all targets for successful runs, or the number of time steps until timeout for unsuccessful runs. On the contrary, as for the target algorithms in Chapter 5, this is a capture cost, meaning the number of steps of avoiding capture. The next is a success which is recorded when a target is captured, i.e., the pursuing agent occupies the same state as the target. In multi-agent scenarios, at least one pursuer's occupancy is enough to claim success. In the presence of multiple targets, success can be achieved when all targets are captured. For the target(s), success is the absence of pursuer success. Additionally, Chapter 6 and 7 measure minimum cost and runtime while Chapter 7 includes maximum cost, too. Having different-sized maps, the experiments are not limited to the fixed runtime, but the runtime is adjusted to the map size and runs out of time (timeout) at 10 times the height of the map. Statistical analysis has been provided for the significance of the results for all pathfinding costs in every chapter.

The base of the implementation [60] was kindly provided by Alejandro Isaza and it has been extended such that multiple targets and various pursuer-to-target assignment strategies could be tested. The simulation is implemented using C++. Table 2.3 displays that all obtained results from the experiments were conducted on a Linux machine on Intel® Core™ i7 at 2.2 GHz CPU with 16 GB of RAM, except those experiments which are conducted in Chapter 4 are on a Linux machine with a 1.9 GHz Intel® Core™ i7 and 40 GB of its RAM.

All the above-mentioned descriptions are the base settings of the experiments. Chapters 3-7 evaluate different algorithms, therefore, the pursuer-to-target ratio combination, the starting positions of the pursuing agents and the targets and

their compared algorithms are described in the experimentation section of each chapter.

2.6 Discussion

The related work presented in this chapter covers pathfinding search algorithms for single and multiple pursuing agents, alongside target algorithms. This review aims to identify the research problem of the multi-agent search algorithms towards moving targets. While previous sections investigate existing approaches to finding solutions to pathfinding and discuss how the presented algorithms differ from each other, this section is dedicated to highlighting the limitations of the literature and how the proposed approaches in this thesis improve the assignment strategy and the pathfinding for multiple pursuing agents as well as the escaping method for the target. The algorithms that are covered in this chapter are categorised in Appendix A.

Finding an optimal algorithm for a given environment is a difficult task, for instance, the FAR algorithm might be optimal for road networks while unsuitable for maze map environments. Therefore, finding an optimal algorithm for all environments is impossible in principle [194]. Therefore, multiple agents are required to collaborate and work collectively to achieve the goal which is moving to the stationary goal destinations or capturing all available targets in dynamic environments. Within this scope, computing the optimal solution is a challenging task, especially in complex environments. First, targets must be identified and assigned to the pursuing agents. Second, agents individually or teams of agents find paths to the assigned targets. Consequently, this is the area Chapter 2 is primarily investigating. The possible solutions in Section 2.3 mainly focus on multi-agent pathfinding problems with static targets and only a few consider moving targets. Although the review presented in this section discusses the pathfinding solutions for multiple pursuers, still computing the path is performed by autonomous agents. The various methods applicable to single-agent pathfinding algorithms are described in Section 2.2.

Pathfinding algorithms should find the shortest path, avoid obstacles, and reach the stationary or moving target within the time limit. Even in a simple

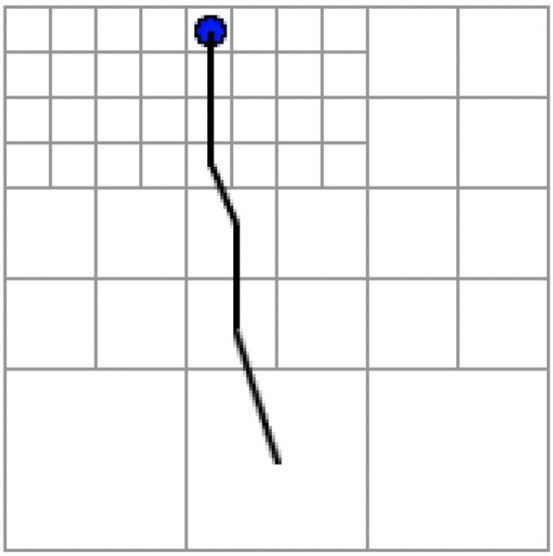


Figure 2.4: An environment with three different grid sizes as shown in [1]. A circle is an agent and its generated path.

environment, the pathfinding search algorithm faces several challenges and one of them is the varying grid size in the environment. Figure 2.4 illustrates grids closer to the agent with a higher resolution plan a detailed search and grids away at a distance with lower resolution create a rough plan and compute faster [1]. The key challenges are switching between different resolutions and representing dynamic obstacles. Moreover, grid maps, for instance, gaming maps from Baldur's Gate, are easier to navigate when compared to room-shaped or maze maps [97].

Another challenge is a large environment where the algorithm performs worse [32] and produces a movement path in a longer period [36]. If the environment is dynamic, not every algorithm can be used [124]. The design of the environment is important, it may lead to collisions or deadlocks [31].

When choosing a search algorithm for moving targets, one criterion should be considering memory consumption and computation efficiency [36, 89]. Cost-minimal movements are essential, and improvements demonstrated efficiency in G-FRA*, Field D* and TBAA*. Some IHAs generate a path from the stationary target location to the agent location. This is efficient for D* Lite as it can shift the map to find a quick path but makes it slower if the target moves in a dynamic environment. Then, D* Lite cannot reuse some previous search information [80].

Real-time heuristic algorithms search from the agent location to the target and MT-Adaptive A* can do it and vice versa [95].

One way to find a solution for the pathfinding problem for multiple agents is to have minimal interaction among the agents and use prioritised planning [5]. For instance, WHCA* [32] executes planning in the prioritised framework, where agents move in the window space to avoid conflicts. Although there can be valid solutions, it does not guarantee the completeness or optimality of the solutions [5]. The planning is much quicker with FAR [31] which uses less memory and performs even better with more pursuers [195].

Some multi-agent pathfinding algorithms were demonstrated whose performance could change depending on the grid map, size of the map, number of agents, and branching factor; and there is not any global best algorithm [33]. An algorithm such as CBS, an optimal solver for MAPF problems, can underperform in environments with open spaces and display worse performance than A* [159]. Similarly, A* searches might be a quicker solution than LPA* if the previous search tree is different from the current one [95].

The majority of the existing search algorithms aim to find the shortest path to the target location. Meanwhile, the CRA algorithm contradicts this to be the main objective, instead, it prefers working collectively to minimise choices of the target [60]. Although the shortest path is important, the timing is essential too as seen in real-time heuristic algorithms [36].

Multi-agent pathfinding approaches trade-off between coupled and decoupled approaches [192, 196, 197]. The coupled algorithms are complete and can be optimal, however, the computation is excessive, and they do not scale with the number of agents as the state space grows exponentially. Whereas the decoupled approach is effective and can work with many agents, on the other hand, the completion of the path, its runtime and the minimum cost are not guaranteed. There are a few methods that utilise these two approaches by joining together [162]. For instance, Biased Cost Pathfinding (BCP) [196] combines them to avoid expensive replanning, and collision among agents, and applies time constraints.

Section 2.4 presents a review related to the target algorithms where the work is based on escaping strategies. Target algorithms, without strategy but considering a pursuing agent's position, can escape from the pursuers but sometimes might

fall into the path of other pursuing agents. This causes an issue in multi-agent frameworks. To avoid this limitation, the study in this thesis considers every pursuing agent, and this new approach provides a winning strategy for the target.

The literature review of the algorithms indicates that there are many different solutions to solve a pathfinding problem. Initially, single-agent pathfinding problems were reviewed to demonstrate the variety of possible solutions. This further continued to the multi-agent algorithm, which indeed, uses some of the solutions that are provided for a single-agent. For instance, Adaptive A* with various versions developed to MPGAA*, or D* Lite being extended to MOPBD*, or different variations of A* such as LRA*, OD, ID, HCA* or WHCA* are implemented in multi-agent problems. However, these solutions do not cover every problem or scenario, and especially the complex problems are left with less attention.

The review indicates that research needs to continue for multi-agent algorithms, particularly in environments with moving targets. Table 2.1 categorises multi-agent algorithms with respect to their relation to single or multiple and static or moving targets. Moreover, with the increased number of players, the research implies the requirement for a good assignment strategy for pursuers, which has been an area of investigation. In the meantime, the current criteria for the assignment strategies are presented in Section 2.5.2. Similarly, it is inevitable to develop an effective strategy capable of providing competent solutions for the multiple pursuers in order to surround and capture the targets.

These highlighted points suggest new methods to improve and solve multi-agent multi-target problems. One solution is to use variants of A* with the current assignment strategy criteria. Meanwhile, this leads to investigating further other criteria options to assign targets, which have not been studied yet in the literature. Although the review of algorithms aims to quickly catch a target and concentrate on the shortest distance, only the CRA algorithm uses coverage areas to minimise target escaping options. This method was not studied with multiple targets, consequently, it leads to further investigation of how multi-agents could capture not only static targets but also moving multiple targets. Therefore, this develops a new multi-agent algorithm that is capable of capturing multiple moving targets in an efficient way. This solution needs to consider coupled or decoupled approaches.

The review presented in this chapter aims to identify the gap while considering the scope and its limitations. It provides an overview of the pathfinding search algorithms and discusses various prevalent methods and approaches for both, single and multi-agent algorithms. Additionally, the limitations of using these algorithms are also covered in this chapter, which motivates this thesis to find new approaches that address the gap. Alongside this, the area of research is illustrated in Figure 1.1 which displays the problem set. Hence, the efforts of this thesis are focused on exploring multi-agent pathfinding solutions to provide alternatives to the current approaches.

Chapter 3

Coordinating Multiple Agents with Assignment Strategy to Pursue Multiple Moving Targets

3.1 Introduction

The research aims to find pathfinding solutions for multiple Pursuing Agents and Moving Targets (PAMT), which is a challenging problem, especially in real-time applications. This type of problem is considered to be NP-hard [14, 198, 199] in the theory of computational complexity, that solving these problems efficiently is not known. Further complications can arise when one expects that agents are required to compute quickly enough to avoid slowdowns, deadlocks or running out of allocated time [95]. The agents need to coordinate their actions and move quickly while avoiding obstacles [200, 201]. They can detect and approach the target within a specified area and use a suitable heuristic algorithm that enables them to search, identify and catch the target [36].

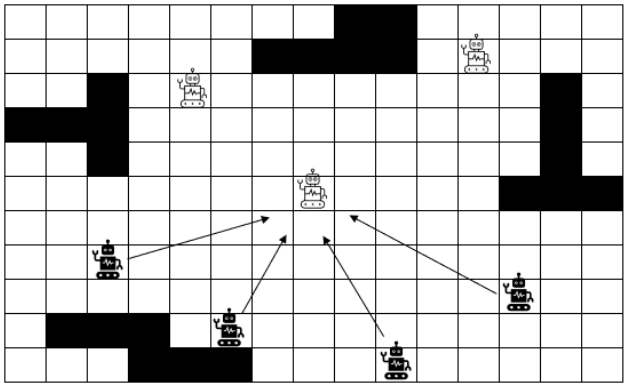
The PAMT is an important problem that has many applications, and yet, despite various methods introduced, there are still limitations to the existing algorithms either by the number of targets or the moving targets they chase or coordination in assigning the targets. For instance, if agents plan their actions independently, conflicts may arise while they execute their plans. Issues like

whether and how agents communicate and coordinate their actions must also be considered [37]. Hence, to overcome these limitations, in this chapter, the issues of coordinating among agents, target assignment, finding the shortest path for each agent and most importantly, pursuing multiple moving targets are addressed.

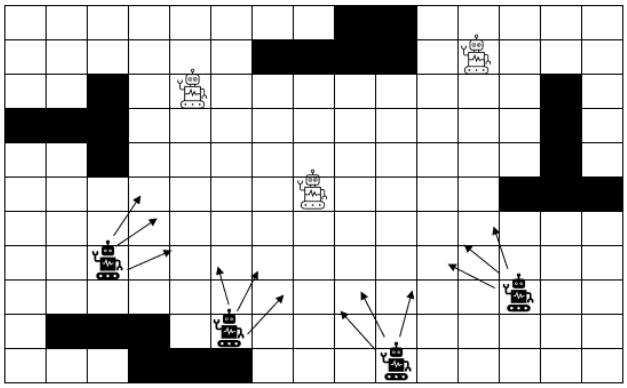
The objective is to develop a new method of solving the problem using assignment strategies for moving multiple targets. The problem can be divided into coupled and decoupled approaches. The first part of the algorithm proposed is called Assignment Strategy, which is the coupled approach that can identify targets, evaluate their positions, and coordinate pursuers. The algorithm runs all possible pursuer-to-target combinations at the initial position using the specified criteria and the optimal combination is chosen that assigns one target to each pursuer.

Figure 3.1 illustrates multiple robot agents aiming to catch multiple targets. Agents with a strategy to follow the closest target will chase only one closeby target and leave the rest as depicted in Figure 3.1(a) while in Figure 3.1(b) agents can chase any random target, which might cause disruption and chaotic environment. It is possible that all agents can start at the same position, and a naive assignment strategy causes all of them to follow the same path, which can cause boredom or a lack of immersion in game applications. The issue can be addressed with a combined force or trap strategy [202], which not only increases the difficulty but also makes the game more interesting. Figure 3.1(c) displays that agents with a given strategy can follow only one assigned target and this increases the chance of catching all targets quicker. Therefore, the assignment strategy displays an important step for multiple pursuing agents in identifying which target to follow which can help in finding an optimal path.

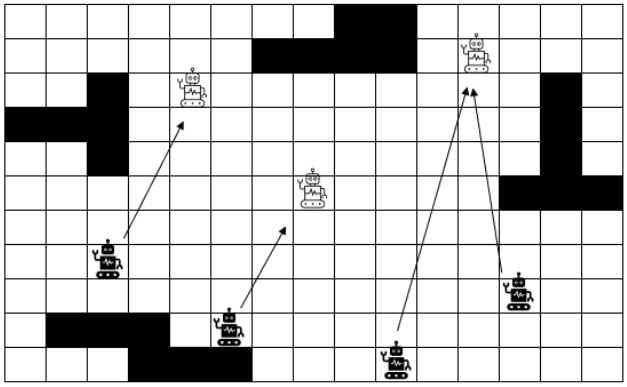
The main algorithm proposed is called Strategy Multiple Target A* (STMTA*). This is the decoupled approach where STMTA* collects information obtained from the assignment strategy, and it searches for a minimum-cost route at each time step for each pursuer and navigates it towards the moving target in dynamic environments. To find the optimal path, it independently computes all available paths starting from its neighbouring states towards the target and the path with the lowest cost is selected. If the target is caught by the assigned pursuer(s) on successful runs, then only these pursuers



a) closest target strategy, only one target approached



b) no strategy, random target selection, disorganised environment



c) smart moves with a given strategy

Figure 3.1: Optimisation with assignment strategies. Four black-shaded agents move towards three white-shaded targets that are a) closest, b) with random selection or c) with a given strategy.

are re-assigned with the existing, not caught yet, targets. This give-a-hand function helps other agents to chase and catch the remaining targets quicker. The hypothesis is that the STMTA* increases performance and produces better outcomes.

The implemented STMTA* and its variations are evaluated on standardised commercial gaming grid-based benchmark maps. Therefore, in this chapter, the main contribution is to evaluate the new method through the conduct of extensive and comprehensive experiments that extend the work significantly, by providing:

- A detailed explanation of the algorithm with illustrated examples.
- A detailed description of the assignment strategy and existing criteria.
- A comprehensive set of experiments.
- An increase in the number of maps including benchmark environments from the Baldur's Gate video game [183].
- A variety of pursuer-to-target combinations for experimentation, where the number of the pursuing agents $\geq$ targets.
- Different starting positions that help to evaluate the assignment strategy.
- Statistical analysis using Wilcoxon rank-sum test [203].

In the remaining parts of this chapter, Section 3.2 explains the assignment strategies and the STMTA* algorithm. Section 3.3 evaluates the algorithms empirically and provides the results of statistical tests. Section 3.4 concludes with areas that are left for future work.

3.2 Problem Formulation

There are many search algorithms available for multiple agent scenarios, but only a few can deal with multiple moving targets. Multiple agents need to navigate cooperatively, rapidly capture the moving targets and avoid obstacles. To study and evaluate the behaviour and the performance of multi-agent algorithms, the

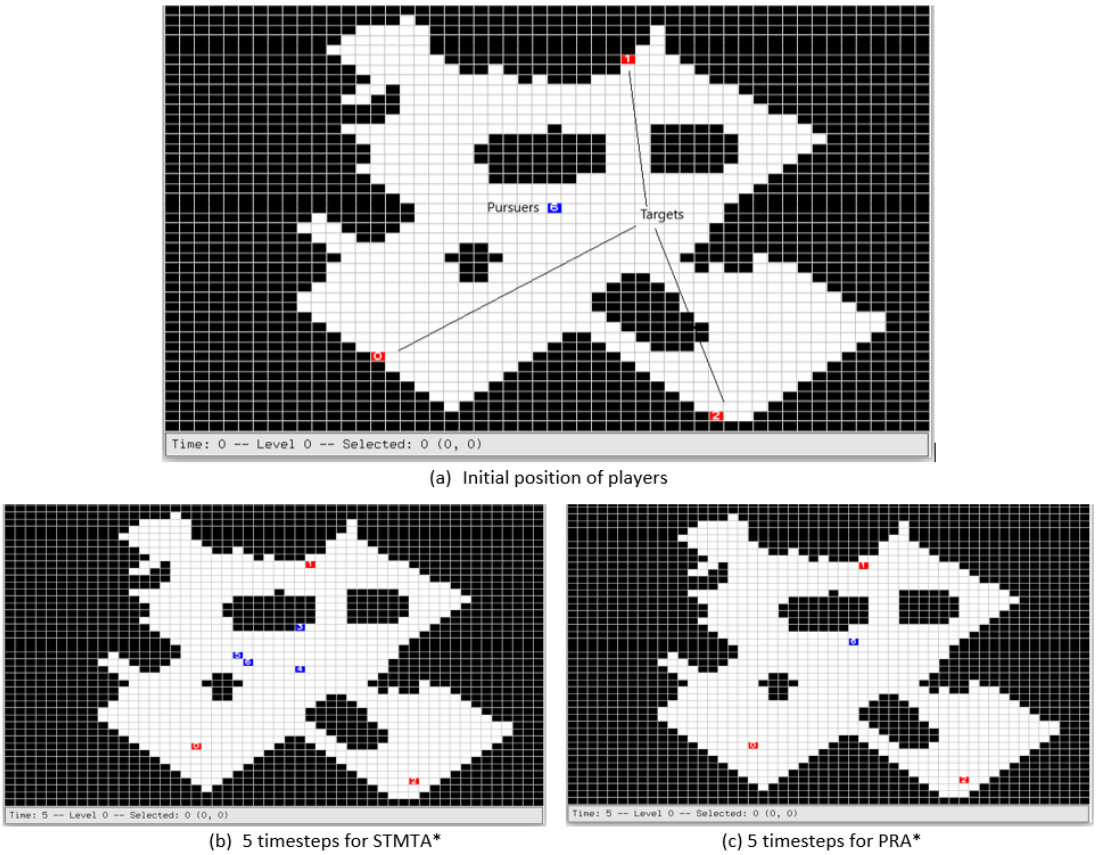


Figure 3.2: Position of 4 pursuing agents in the middle and 3 targets dispersed on the walls on the AR0417SR map (Figure 2.1). The initial state (a), the states after moving 5 steps for STMTA* (b) and PRA* (c). Black shades are non-traversable states.

problem is formulated using the idea of outmanoeuvring and surrounding the target [60]. The pursuing agent considers the position of the target at each time step and recomputes its path to the new state of the target if the target has moved otherwise it uses the previously computed path. The complexity of the problem for multiple cooperative agents is known to be NP-hard [14].

The common feature of the search algorithms that have been discussed in Chapter 2 is that their purpose is to get to the target as quickly as possible. In contrast, STMTA* often creates pursuit scenarios where agents do not necessarily take the shortest path but move in such a way that the target is trapped and outmanoeuvred. Firstly, at the initial state, before any moves

start, the assignment strategy algorithm allocates targets to agents after evaluating all possible combinations according to the set criterion. Then, in the next stage, once each pursuer is assigned a target to follow, the STMTA* computes a path and distance towards the target from its adjacent non-occupied neighbouring states. One of the actions (paths) that reach the target fastest is selected. As this is not a deterministic step, pursuers may not necessarily follow the same route towards the target even if they start from the same position, as displayed in Figure 3.2(a). However, a common strategy is to approach the closest target. For instance, the PRA* algorithm is one of them which independently and without coordination moves to the target. Unfortunately, all pursuers will follow the same target and leave the others free, unlike in the STMTA*. Even after 5-time steps when all players have repositioned and have new states. PRA* moves to the target at the top of the map as it is the closest in the distance as seen in Figure 3.2(c), while in Figure 3.2(b) STMTA* spreads out its pursuers to chase the targets that are previously assigned. This utilises resources for STMTA*. Another example that often leads to situations where the pursuers approach a target from different directions, leaving the target fewer or no opportunities to escape. For example, in maps that have round, circle-shaped obstacles similar to Figure 3.4(a). PRA* selects the same action for each agent, but STMTA* might select two different routes (clockwise and anticlockwise). This is because of the randomness it uses for its actions. Thus, targets are surrounded and trapped.

This study uses the PRA* algorithm as a baseline, and experiments are conducted against the STMTA* algorithm with its two variations, the Summation-cost criterion and Mixed-cost criterion. For the PRA* algorithm, a pursuer is simply assigned to the target that it is closest to, and the rest of the targets are pursued if the distance shortens. The performance is studied on complex maps with multiple pursuing agents and targets that have various starting positions.

For the statistical tests, the null hypothesis (H_0) is that the PRA* algorithm and the STMTA* algorithm have equal performance. It is aimed to refute it and show that the performance of the STMTA* algorithm is better.

Algorithm 1 Assignment Strategy Algorithm.

```
1: function assignTarget(target)
2:     agents ← getAllAgents();
3:     targets ← getAllTargets();
4:     for each agents a do
5:         for each targets t do
6:             d ← getDistanceCombinations(a, t);
7:         end for
8:     end for
9:     if d is not empty then
10:        c ← computeAssignmentStrategy(d);        ▷ get optimal combination
11:        p ← assignAll(c);
12:    end if
13:    return p;
14: end function
```

3.2.1 Assignment Strategies

The success of search algorithms is when an agent occupies the same location as the target within a given time space and has a shorter distance. These types of searches are easier if they involve one-to-one players or multiple agents to one target. The problem becomes complex when the number of players increases. The question arises of how searching agents know which target they need to chase before making the first move.

Algorithm 1 outlines the assignment strategy steps for multiple pursuing agents. The aim of the algorithm is to find the best possible combination for pursuing agents out of all available combinations within the given strategy that are computed between each agent and each target. The steps of the algorithm are as follows: initially, in line 2 all agents, *agents*, and in line 3 all targets, *targets*, are identified and placed in the lists. When all players are put in the relevant lists, the distance is computed by looping through each target, t, for each agent, a, to get the distance combination, d, at lines 4-8. If the computed distance combinations are not empty for d at line 9, then the best combination, c, at line 10 is selected based on the given assignment strategy (Summation-cost or Mixed-cost). The computed c will be assigned to the agents at line 11 to pursue relevant targets that return p at the final step in line 13. The

assignment strategies, Summation-cost and Mixed-cost, are used in this study and they are described in Section 2.5.2.

The algorithm results in a search for a path for a fixed number of pursuing agents. Since the algorithm is run in the coupled stage, the complexity is analysed where the runtime of each distance computation is in the worst-case exponential in the number of pursuing agents. Computing the distance between an agent and a target is the most time-consuming operation. Line 2 in Algorithm 1 calls a function to loop through all players which then identifies the agents and appends them to *agents* list. This operation requires n time. Similarly, line 3 calls a function to identify the targets and appends them to the *targets* list with n time. Lines 4-8 instruct two nested loops for agents and targets. Line 4 requires k operations for the pursuing agents and line 5 requires l operations for the targets. The total number of players is $k \times l$. However, line 6 calls a function to get the distance between an agent and a target that needs n times. Line 9 checks the condition and executes *one* time and then line 10 calls a function to get the optimal combination with $\mathcal{O}(n)$ time complexity. Line 11 is an assignment statement with *one* time. Therefore, the function in the worst-case makes the time complexity of $\mathcal{O}(kln)$. Although getting a distance between a pursuer and a target seems a straightforward computation, it can grow exponentially with many combinations for multiple pursuers and it makes it impractical for larger problems.

3.2.2 Strategy Multiple Target A*

As explained in Section 3.2.1, the algorithm starts by considering all possible assignments of pursuers to targets and choosing one based on criteria such as those described in Section 2.5.2. Then each pursuer computes the shortest path towards the target using heuristic methods. The common way is to compute the path from the current position but the STMTA* algorithm computes from its non-blocked neighbouring positions separately, and one of the paths with minimal distance becomes the route to take. Although the targets move away, the pursuers keep re-computing a path at each time step and thus chase their assigned targets until the capture or timeout. On successful runs when the target

Algorithm 2 Strategy Multiple Target A*.

```
1: function getPathForStrategy()
2:     assignedTarget ← assignTarget(target);
3:     p ← currentpursuerstate;
4:     t ← currentassignedTargetstate;
5:     get all neighbours_list for p;
6:     min_distance ← max();                                  ▷ max integer
7:     if neighbours_list > 1 then
8:         for each neighbours_list n do
9:             compute distance d at position n;
10:            if d < min_distance then
11:                min_distance = d;
12:                best_action ← n;
13:            end if
14:        end for
15:    else
16:        best_action ← neighbours_list[0];
17:    end if
18:    return best_action;
19: end function
```

is captured by one or more pursuers, instead of computing new assignments for all pursuers, the algorithm re-assigns only for these pursuers, which is sub-optimal but comparatively quick.

The pursuing agents have full knowledge of the map. Since the assignment or re-assignment is done from a global perspective rather than by individual agents, this is equivalent to a scenario where agents can communicate without noise, delays, or bandwidth limits. For STMTA*, the number of pursuers, m, and the number of targets, n, do not have to match, the scenarios for experimental tests were based on $m \geq n$ such that no target would be left unassigned. In those situations where the agent is left without a target assigned, the algorithm re-runs the *assingTarget()* function from the Assignment Strategy algorithm and provides a new target that has still not been caught by the other pursuing agents. Algorithm 2 displays the pseudo-code of STMTA*.

The function *getPathForStrategy()* finds the best possible move for the STMTA* algorithm. It starts with assigning the target at line 2 that was

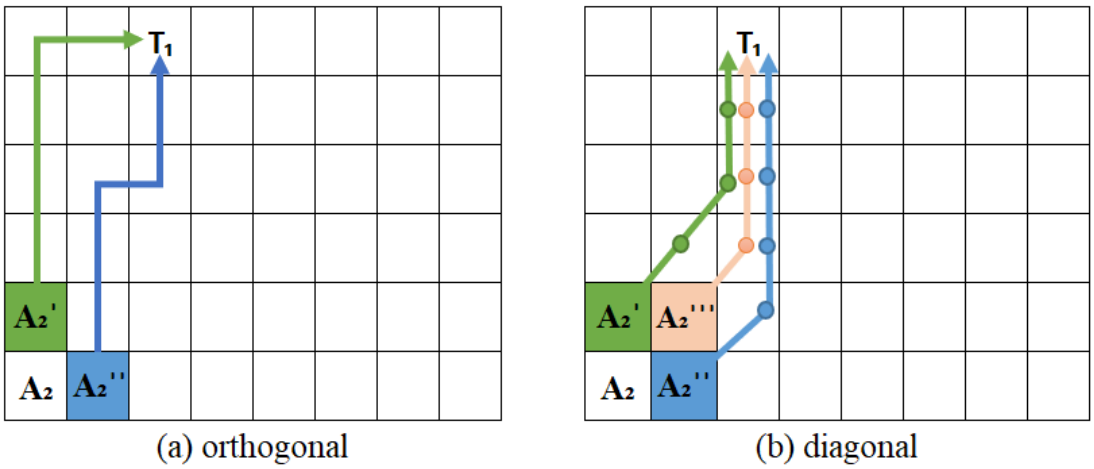
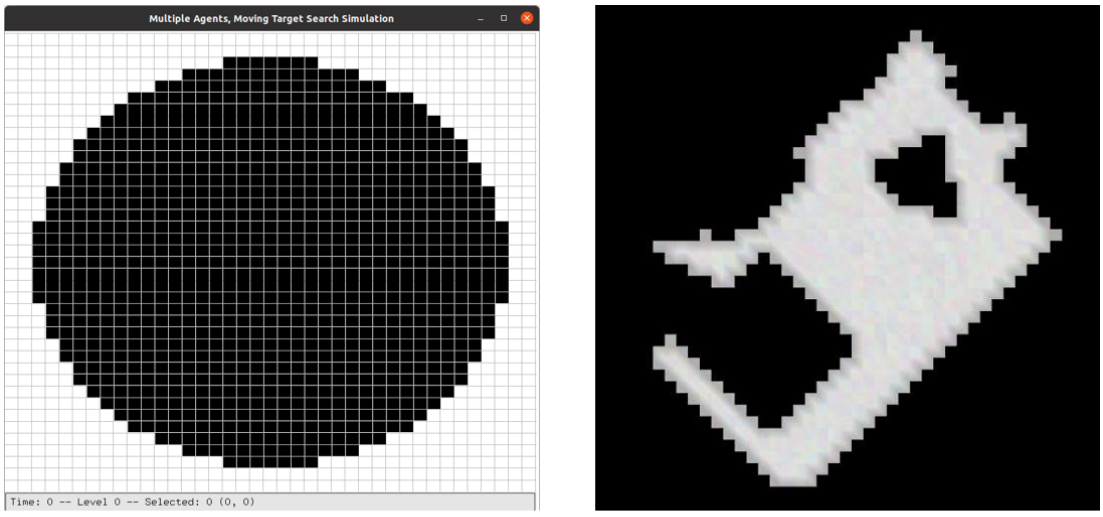


Figure 3.3: Pursuing agent A_2 towards a target T_1 on a 2-D map for (a) an orthogonal distance cost and (b) a diagonal distance cost.

identified using Algorithm 1. Lines 3 and 4 set the position p and t for each player, the pursuer and the target respectively. If the neighbours p' of p are traversable and there are no obstacles, they are inserted into the *neighbours_list* at line 5. This is also the list of possible actions. If the number of actions is one, then it is the *best_action*, line 16, else the actions are more than one. Starting from line 7, the algorithm loops through *neighbours_list* to find the shortest distance d towards t. The heuristic search algorithm A* is used to compute d at line 9, and it is compared with other p' within lines 10-13. The *best_action* for p is the p' with the lowest value, line 12.

The experiments in this study use the Manhattan distance, but it is possible to use a diagonal distance. To illustrate this, Figure 3.3 displays orthogonal neighbours with distance costs of 1 and diagonal neighbours with a distance cost of $\sqrt{2}$. From the position of pursuer A_2 towards target T_1, there are two neighbouring states that are added to the *neighbours_list* at step 5 of Algorithm 2. The possible actions (routes) with 5 steps from position A_2' and 6 steps from position A_2'' are displayed with arrows towards the target as shown in Figure 3.3(a). But Figure 3.3(b) has got three neighbouring states with additional diagonal movement and the possible actions with a cost of 4.8 from position A_2', 5.4 from position A_2'' and 4.4 steps from position A_2'''. The *best_action* for the orthogonal distance is 5 steps from the neighbouring position A_2', and the



(a) (b)

Figure 3.4: Sample maps used in the experiments, (a) round circle-shaped RoundTable39x39 and (b) benchmarked AR0527SR from Baldur's Gate video game.

best_action for the diagonal distance is 4.4 steps from the neighbouring position A_2'''.

Although the time complexity of Algorithm 1 depends on the number of pursuers, Algorithm 2 finds a path per pursuer where the planning is individual and there is no need to interact with other pursuers. Lines 2, 3, 4 and 6 are the assignment statements that generate *one* time. The pursuer has its state and four neighbouring positions, thus line 5 executes *five* times if not blocked. Line 7 checks the condition of whether there are more traversable neighbouring positions for the pursuer which requires *one* time, and then the algorithm performs a loop to find the shortest distance in line 8 with k time requirement. Similarly, lines 9 to 12 require k times of operations. Line 16 is an assignment operation that executes *one* time. The time complexity of the algorithm is $\mathcal{O}(k)$. However, if the distance is computed using the A* algorithm and its time complexity is $\mathcal{O}(b^d)$ where b is the branching factor and d is the depth of the solution [84]. Moreover, the difficulty of the map and the length of the path have a proportional effect on the complexity [204].

It is possible to use other parameters such as different speeds for players.

For example, MTS [131], Moving Target D* Lite [80], Incremental ARA* [113] and some other algorithms are successful in solving the problem if the target occasionally misses a turn. The empirical evaluation section demonstrates that it is possible to combine forces and trap targets while maintaining the same speed for all players, which makes the algorithm more powerful. The method might be slower overall (because of the overhead of computing assignment strategies) in comparison to "rush-in" methods, but it demonstrates a higher chance of being successful, i.e., catching all targets. STMTA* is capable of surrounding the target unless it ends in a deadlock or runs out of time steps in the scenarios where round circle routes exist, as shown in Figure 3.4(a). Therefore, to evaluate the algorithms in various scenarios and proceed with similar environment settings from the previous studies [60, 171], the experiments included purposely-made maps only in this Chapter 3 where the details of maps are displayed in Table 2.3.

3.3 Empirical Evaluation

Empirical results are presented in this section to demonstrate the efficiency of the proposed new STMTA* algorithms. First, the setup for the experiments is described, and then, the measurement of pathfinding cost for the solutions and the performance success is explained. Finally, the results are tested for statistical significance.

3.3.1 Experimental Setup

The general setup of experiments such as the map environment, player combinations, the movement direction of players, the cost of moves, the number of test runs for each configuration and the total number of individual tests as well as the computer settings are detailed in Section 2.5.3.

The performance of STMTA* with two criteria, Summation-cost and Mixed-cost, is tested and compared to the previous approach, the PRA* algorithm which has been used in the literature often [141]. PRA* uses a greedy approach that loops through all players and assigns a target with the closest distance, and not all targets might be chased, as some targets could be positioned at a further

distance than others for all pursuers. A similar situation is depicted in Figure 3.2. By using abstractions of search spaces, PRA* is quicker than A*.

The targets are not allocated to the pursuers in advance, instead, each agent algorithm identifies its target to follow. PRA* selects targets based on their nearby position on the map, whereas STMTA* assigns all targets to all pursuing agents using the assignment strategy algorithm, either the Summation-cost or the Mixed-cost criterion. The targets use the SF algorithm to avoid capture from the pursuing agents and a detailed description is in Section 2.4.

Figure 3.4 illustrates only two out of ten sample maps that are used to evaluate the performance of algorithms with different ratios of pursuers and targets and different starting positions. The combination of players can be seen in Table 3.1.

There are four different starting positions, and all players are located at randomly selected positions on the maps. The first starting positions, $state_1$, for the pursuing agents are on the right bottom corner, similar to previous experiments [29] on Figure 3.4(a), and the targets' positions are further away on the left side of the map. Pursuing agents and targets are aggregated in one position in the first test run [60]. All other starting positions, $state_n$, are

Table 3.1: The pathfinding cost (number of steps), lower is better, and success rate (%), higher is better, are displayed for each algorithm with player combinations on each row.

Player Combinations (Pursuer vs Target)	PRA*		STMTA*			
			Summation-cost		Mixed-cost	
	Number of steps	Success Rate	Number of steps	Success Rate	Number of steps	Success Rate
3 vs 2	79.52	88.63%	69.87	96.58%	66.76	97.50%
3 vs 3	116.89	82.22%	80.34	97.83%	80.53	97.47%
4 vs 2	84.69	87.46%	48.82	99.30%	47.4	99.63%
4 vs 3	94.79	85.64%	74.54	98.11%	73.35	98.50%
5 vs 2	88.41	86.25%	49.28	99.38%	49.97	99.46%
5 vs 3	79.36	89.86%	53.54	99.50%	51.4	99.61%
Average:	90.61	86.68%	62.73	98.45%	61.57	98.69%

dispersed and positioned as far away as possible or next to the map walls. The similar positioning is applied for all players, pursuers and targets, on all maps. These positions provide a diverse set of scenarios where the behaviour and performance of the algorithms can be observed.

3.3.2 Experimental Results

Performance analysis is conducted with respect to two key indicators; (i) the pathfinding cost is the number of steps taken before capturing all targets for successful runs, or the number of time steps until timeout for unsuccessful runs, and (ii) its success rate. Both measurements are averaged considering all pursuers. To evaluate the overall performance of the algorithms, the comparison table is displayed in Table 3.1 grouped by the number of pursuing agents against targets.

During the experiments, success for the pursuing agents is achieved when at least one pursuer reaches each target and occupies the same state as that target on the map. The 100% success is measured when all targets are captured. The pursuing agents cannot always reach the target before the timeout. But in all cases, a lower number of steps indicates a more successful algorithm.

The experimental results are run on ten different maps described in Section 2.5.3 and each row in Table 3.1 is the mean taken per player combination. It can be seen from the results that both variations of STMTA* manage to reach the targets quicker, i.e., with less pathfinding cost and have a better success rate compared with PRA*. The Mixed-cost criterion was previously reported that it is overall the best [53], and the results displayed in Table 3.1 support that.

When the difference of success rate is taken in each row for player combinations in Table 3.1, then the lowest is 8% for 3 vs 2 and the highest is approximately 16% for 3 vs 3 is seen when the Summation-cost criterion is employed. Although STMTA* Summation-cost has slightly better success rates in the 3 vs 3 and 5 vs 2 player combinations, this is because STMTA* Mixed-cost missed the target and went to timeout on a few occasions on various maps, but the time steps mean shows better results.

Comparing scenarios with different pursuing agent combinations shows that, as expected, when the number of pursuing agents increases, the pathfinding cost

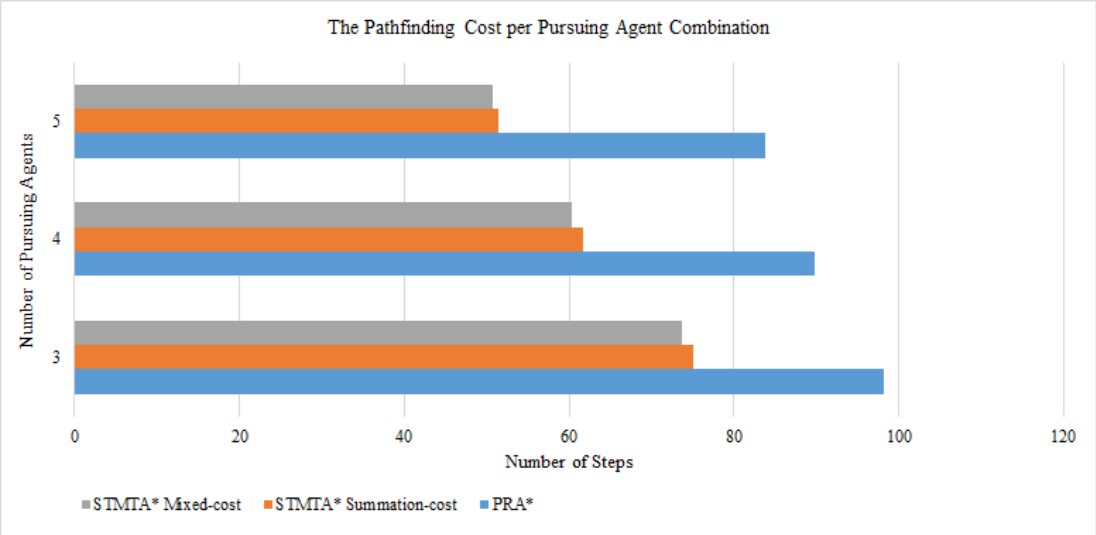


Figure 3.5: The pathfinding cost mean per pursuing agent combination for all algorithms.

tends to decrease, as illustrated in Figure 3.5, and the success of catching the targets tends to increase, see Figure 3.6.

Even though the results in Table 3.1 indicate that the STMTA* algorithm with its two variations has an overall success rate of over 98%, it performs weaker on round circle maps in comparison with other maps, but it still outperforms the PRA* algorithm which has 86%.

For example, the RoundTable39x39 map (see Figure 3.4(a)) is used, where the obstacles are shaped as a round table and players navigate by circling around it. Figure 3.7 illustrates the results for all player combinations for this map environment. There are two results displayed, the top half is a line chart for success rate and the lower part is for a number of steps using the bar chart. The maximum success rate for the PRA* algorithm is 75% and it performs worst in 3 vs 3 player combination with 58%. The reason is that PRA* rushes in towards the target and keeps chasing it, as all agents use the same speed, it hopes that the target makes a mistake by turning in the wrong direction, otherwise they loop continuously until timeout. This has been the main issue for PRA*. On the other hand, the STMTA* algorithm is more successful because the pursuers approach the target from different directions. The

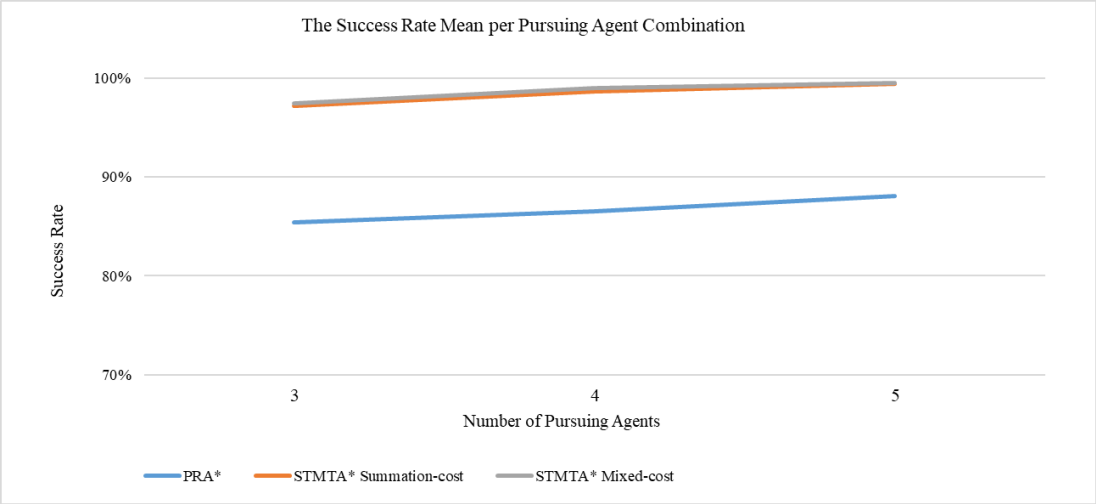


Figure 3.6: The success rate mean per pursuing agent combination for all algorithms.

difference in the number of steps is displayed on bar charts. The performance of PRA* is similar against agent combinations when two targets are present but shows significant change with three targets when the pursuers increment. Only in 3 vs 2 player combination, PRA* performs noticeably well against STMTA* Summation-cost but still demonstrates poorer results in all other test runs. In contrast, the success rate of STMTA* with both variations is higher, and there are fewer steps until targets are caught.

PRA* displays similar lower results on other maps too. The success rate is 79.6% on a smaller RoundTable09x09 map and 72.3% on the MTS2 map and it has the worst performance on the Empty30x30 map with 43.1%, whereas the STMTA* algorithm has over 94% of success for the RoundTable09x09 and the MTS2 maps. The environment without any obstacles, the Empty30x30 map, has a success rate of 100% for STMTA*. The STMTA* Mixed-cost demonstrates overall the best results.

Despite the fact that STMTA* was successful in most of the test runs, it displayed a drop in relative performance on the AR0407SR and AR0528SR maps. All three algorithms have a 100% of success rate but PRA* was slightly quicker to catch the targets on these map environments.

Next, statistical tests are used on the pathfinding costs to find out which of

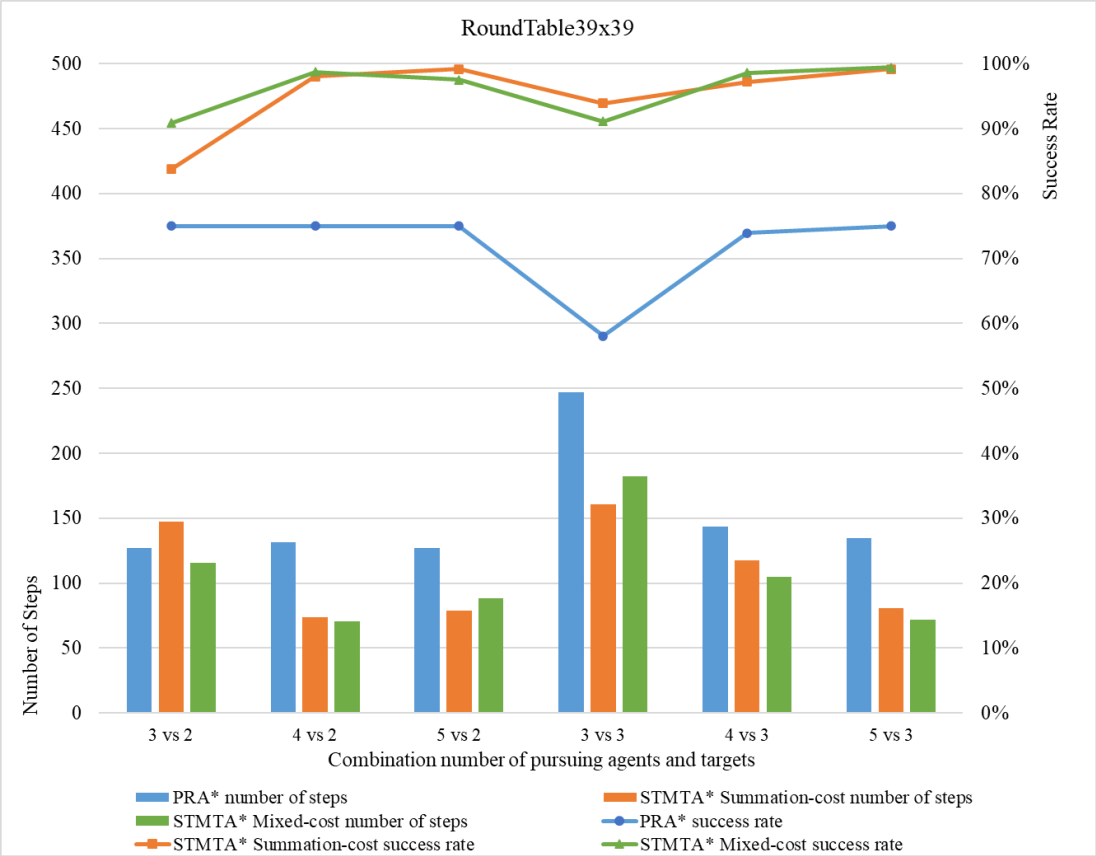


Figure 3.7: The performance analysis of three algorithms on a RoundTable39x39 map, measuring the pathfinding cost (number of steps) and success rate.

the results are significantly different. The pathfinding costs are not normally distributed; therefore, the statistical results are obtained using the Wilcoxon rank-sum tests. The level of significance used is 0.05.

Table 3.2 provides a heat map for the values obtained from the statistical tests. The table compares the p-values for each initial starting position for PRA* with STMTA* Summation-cost (left column) and for PRA* with STMTA* Mixed-cost (right column) algorithms. $State_1$ is the aggregated and $state_n$, $n > 1$, are the dispersed positions for pursuers.

From this table, the majority of the results display statistically significant differences. Most of the $state_1$ aggregated positions show significance, in contrast to $state_n$ dispersed positions. What stands out in the table is a significant difference in round table maps (RoundTable09x09 and

Table 3.2: The statistical analysis is used between PRA* and STMTA* algorithms and the p-value obtained using the Wilcoxon rank-sum test. The results are grouped by the starting position for each test run on all maps and player combinations. The p-values below 0.05 have no shades.

Players configuration	Maps	p-value: PRA* vs STMTA* summation cost				p-value: PRA* vs STMTA*mixed cost			
		State1	State2	State3	State4	State1	State2	State3	State4
3 Pursuers vs 2 Targets	RoundTable09x09	3.8E-08	2.4E-08	0.3438	4.5E-13	3.6E-08	9.8E-11	0.4733	4.5E-13
	RoundTable39x39	0.0001	8.0E-10	4.0E-11	4.8E-10	1.3E-05	0.2028	1.7E-10	4.4E-09
	MTS1	0.0341	0.0007	0.0002	0.0017	0.0323	0.0021	0.0002	0.0002
	MTS2	8.8E-07	1.7E-12	6.9E-12	9.3E-13	1.7E-08	4.5E-11	4.6E-11	7.4E-13
	Empty30x30	1.2E-12	1.2E-12	NA	1.3E-08	1.2E-12	1.2E-12	1.6E-09	1.2E-10
	AR0407SR	0.4486	0.01293	6.4E-09	0.1217	0.0245	0.0449	0.1416	0.0002
	AR0417SR	0.0001	7.9E-07	0.1582	3.8E-09	0.0030	3.0E-06	0.1296	5.8E-10
	AR0527SR	0.2136	6.7E-06	0.6565	0.0143	0.9646	2.8E-05	0.6890	0.6461
	AR0528SR	4.7E-05	4.0E-05	2.2E-08	0.0040	4.0E-07	1.2E-10	1.0E-08	0.0023
	AR0707SR	0.0164	9.9E-09	0.1689	2.7E-08	9.2E-05	8.3E-11	0.0273	5.3E-06
3 Pursuers vs 3 Targets	RoundTable09x09	7.2E-13	9.7E-13	0.4602	0.0004	4.0E-11	1.0E-12	0.7587	0.0076
	RoundTable39x39	6.1E-14	0.0005	0.0195	5.7E-12	4.9E-13	0.0013	0.0112	2.3E-11
	MTS1	1.1E-12	0.0175	4.8E-05	2.2E-05	1.1E-12	0.0054	2.8E-06	0.0020
	MTS2	0.0110	1.3E-08	0.8155	5.8E-10	0.3337	1.1E-12	0.9701	2.0E-11
	Empty30x30	1.2E-12	1.2E-12	9.9E-13	1.2E-12	1.2E-12	1.2E-12	8.3E-13	1.2E-12
	AR0407SR	0.0270	0.2863	7.6E-05	7.7E-05	0.0070	0.9409	3.4E-06	0.0004
	AR0417SR	1.9E-07	0.0209	1.5E-06	0.3286	9.3E-05	0.6252	0.0073	0.5385
	AR0527SR	0.0368	0.0123	0.6198	0.7058	0.1386	0.0079	0.7956	0.0944
	AR0528SR	2.1E-07	1.5E-11	2.7E-07	0.0934	4.2E-08	3.5E-08	9.1E-09	0.0670
	AR0707SR	0.0100	0.0007	0.5655	0.5004	0.0011	0.0031	0.2841	0.7111
4 Pursuers vs 2 Targets	RoundTable09x09	3.8E-10	5.3E-12	0.0014	1.7E-14	3.9E-12	5.0E-12	0.0014	1.7E-14
	RoundTable39x39	1.3E-10	1.2E-10	0.0001	0.0002	1.6E-11	1.2E-10	1.1E-05	1.5E-05
	MTS1	0.0027	0.0002	1.3E-12	0.0034	0.0046	0.0212	1.3E-12	0.0008
	MTS2	1.7E-14	1.2E-11	1.2E-11	1.6E-13	1.7E-14	1.5E-11	1.3E-11	2.4E-13
	Empty30x30	1.2E-12	1.2E-12	NA	1.1E-12	1.2E-12	1.2E-12	NA	1.2E-12
	AR0407SR	0.1508	0.8756	0.0307	1.2E-09	0.0179	0.0335	0.1162	3.9E-10
	AR0417SR	0.0082	2.1E-07	0.0012	1.5E-07	0.0094	7.3E-06	0.0004	8.5E-07
	AR0527SR	0.0210	3.4E-05	0.0145	1.2E-05	6.4E-05	0.0005	0.2149	3.8E-06
	AR0528SR	7.4E-09	0.0007	0.3386	0.6420	3.5E-08	1.1E-05	0.0761	0.7902
	AR0707SR	0.0280	1.6E-10	0.1048	2.6E-09	0.1892	5.0E-06	0.0073	5.0E-09
4 Pursuers vs 3 Targets	RoundTable09x09	1.2E-12	1.2E-11	4.2E-07	0.0262	2.0E-13	1.3E-11	2.0E-06	0.0413
	RoundTable39x39	5.1E-11	0.0175	4.8E-10	0.2465	4.2E-12	0.0020	1.7E-08	0.9228
	MTS1	4.6E-05	3.6E-05	2.6E-05	4.1E-06	0.0002	0.0731	0.0061	1.9E-10
	MTS2	1	1.5E-07	1.7E-14	0.0044	0.6354	1.5E-07	1.7E-14	1
	Empty30x30	1.2E-12	1.2E-12	9.2E-13	1.2E-12	1.2E-12	1.2E-12	8.3E-13	1.2E-12
	AR0407SR	0.0002	0.0006	0.7775	0.0010	0.0002	0.0431	0.2648	0.0007
	AR0417SR	2.2E-06	0.0006	3.4E-07	0.0001	7.9E-07	1.3E-06	2.9E-07	0.0057
	AR0527SR	0.0220	0.0902	0.1550	0.0071	0.0067	0.0079	0.0139	0.0007
	AR0528SR	1.0E-07	1.6E-05	0.6879	2.3E-05	1.2E-08	2.1E-05	0.4611	0.0008
	AR0707SR	0.0212	9.0E-06	0.0514	2.8E-07	0.0911	7.1E-07	0.7865	0.2071
5 Pursuers vs 2 Targets	RoundTable09x09	2.4E-10	1.7E-14	3.4E-07	2.4E-13	6.4E-13	1.7E-14	3.4E-07	4.5E-13
	RoundTable39x39	1.5E-11	0.0001	1.5E-08	1.5E-12	5.3E-11	0.0005	3.6E-08	3.0E-12
	MTS1	0.3716	0.6348	1.2E-09	0.0425	0.3889	1	1.6E-12	4.5E-08
	MTS2	3.4E-13	4.0E-08	4.1E-08	9.0E-13	2.9E-13	5.5E-07	4.1E-06	9.4E-13
	Empty30x30	1.2E-12	1.1E-12	3.8E-08	1.2E-12	1.2E-12	1.1E-12	1.2E-07	1.2E-12
	AR0407SR	0.0136	0.0004	0.9466	2.4E-08	1.7E-05	7.0E-05	0.3337	1.2E-07
	AR0417SR	2.1E-06	1.9E-08	3.0E-07	4.8E-08	0.0004	7.2E-08	1.2E-06	7.4E-09
	AR0527SR	0.1525	0.1258	0.2946	3.7E-07	5.0E-05	0.0307	0.1468	2.0E-06
	AR0528SR	2.7E-06	9.1E-06	0.0015	3.3E-08	3.1E-06	1.1E-09	0.0009	5.5E-05
	AR0707SR	0.0427	6.4E-10	0.4266	4.5E-10	0.4804	6.9E-07	0.0256	6.6E-09
5 Pursuers vs 3 Targets	RoundTable09x09	1.6E-13	0.6012	0.0007	2.8E-12	1.5E-12	1.2E-06	2.8E-05	2.2E-12
	RoundTable39x39	4.1E-12	0.0769	1.0E-07	1.4E-11	4.2E-12	0.9285	5.3E-09	1.6E-10
	MTS1	7.0E-11	0.9941	2.4E-05	1.7E-08	9.1E-12	0.7104	0.0760	0.0002
	MTS2	1.2E-07	1.3E-08	3.8E-08	4.6E-13	1.7E-09	5.4E-07	9.3E-13	4.4E-13
	Empty30x30	1.2E-12	1.2E-12	0.0003	1.2E-12	1.2E-12	1.2E-12	NA	1.2E-12
	AR0407SR	0.0168	0.0195	3.4E-11	0.9881	2.3E-05	3.1E-07	1.7E-08	0.9468
	AR0417SR	5.0E-08	8.7E-12	0.0003	1.9E-07	3.2E-08	6.0E-12	6.5E-05	2.0E-07
	AR0527SR	0.0634	0.6705	0.1057	9.9E-12	0.8459	0.6607	0.3968	9.6E-12
	AR0528SR	7.5E-06	1.3E-07	0.0799	8.6E-05	2.4E-05	3.2E-08	0.0282	0.0007
	AR0707SR	0.6190	6.5E-08	0.9270	0.0005	0.0035	1.7E-08	0.6007	1.3E-07

RoundTable39x39) at $state_1$. If the experimental results produce the same number of steps for all test runs, then the Wilcoxon rank-sum Test returns nothing for p-value as displayed for the $state_3$ position on the Empty30x30 map.

Overall, most of the results are significantly different, while the exceptions are not distributed very regularly.

3.4 Conclusion

In this chapter, the new STMTA* algorithm is proposed to find the solution for multiple pursuers in a dynamic environment while chasing multiple moving targets. The presented algorithm is divided into two approaches, coupled and decoupled. The coupled approach coordinates all pursuers to find the combination using the assignment strategy algorithm which runs all possible pursuer-to-target combinations at the initial position using the given criteria, and the optimal combination is selected that assigns one target to each pursuer. In the decoupled approach, the pursuers independently compute their path towards the moving target at each time step.

The STMTA* algorithm has been investigated more thoroughly in this thesis on multi-agent and multi-target scenarios using benchmark environments. Detailed experiments were carried out using ten purposely made and commercial gaming benchmark testbeds with six different player combinations. During these experiments, the number of steps and success rate were measured, and for statistical analysis, the Wilcoxon rank-sum test was applied. This study has established that the proposed algorithm captures targets quicker and produces a higher rate of success, approximately by 16%, in scenarios with multiple moving targets. This algorithm, therefore, provides a useful approach in dealing with scenarios where multiple moving targets are present.

Chapter 4

Multi-Agent Path Planning Approach Using Assignment Strategy Variations in Pursuit of Moving Targets

4.1 Introduction

A simple strategy to find a low-cost distance towards the target and move to its position can promise a capture if the speed of agents is faster than the target. An example is the real-time strategy video game, Company of Heroes, where a team of soldiers move quicker [60]. In an environment with multiple players, the search for the path from agents toward the targets is more complex and requires well-thought, rigorous task planning. The most straightforward solution for the agents is to follow the nearest target, however, it might not be the best option in the presence of multiple targets. Figure 4.1 is an example where pursuing agents will follow only Target1 as it is the closest among all other targets. If all agents choose to follow the target that is closest in the distance, then targets that are not chased can affect the total cost, success, and runtime. The challenge increases when the targets are not stationary, and their number increments.

Multiple agents can benefit from two stages, the combination of coupled and

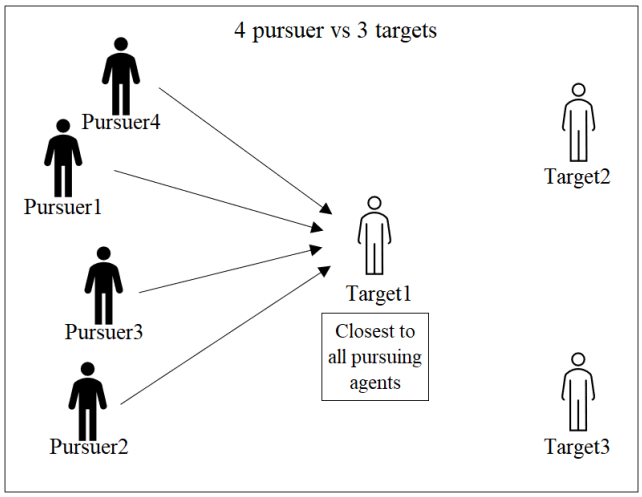


Figure 4.1: A possible scenario where one target is positioned closer than others. Target1 is chased if the strategy for the pursuers is to follow the closest target.

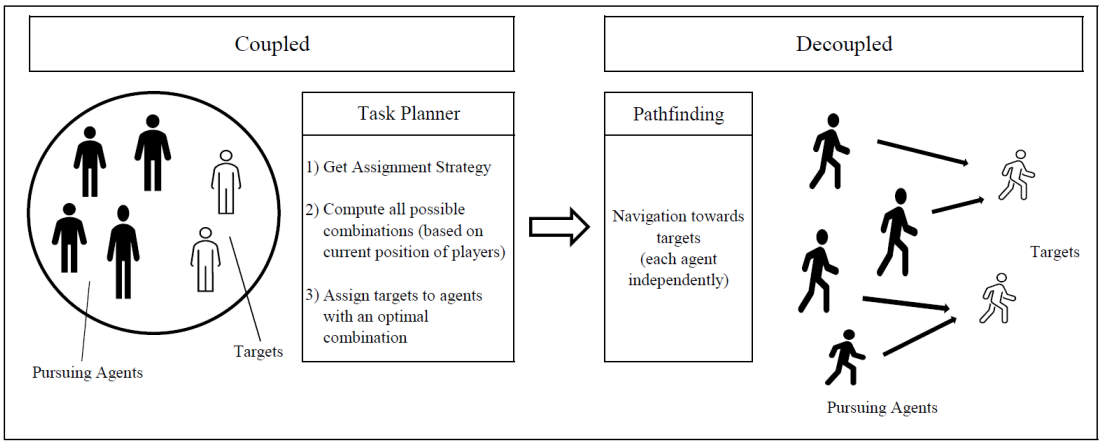


Figure 4.2: Two stages for multiple agents, first planning the task with the best combination and then navigating each agent independently towards the targets.

decoupled pathfinding algorithms, as illustrated in Figure 4.2. The assignment strategy algorithm with the given criterion initially computes all possible combinations for all pursuers in the task planning stage. The combination with the lowest value gets all targets assigned to the pursuers, before the move, and none of the pursuing agents should be idle. Once the targets are assigned, the next stage starts, where all pursuers search their path independently and navigate themselves towards the moving targets using heuristic algorithms.

Pathfinding problem variation and complexity depend on the map environments, the existence of obstacles, a target being able to move or wait until the rescue arrives [26], the number of players and the combination of these. There are many search algorithms that use the classical and standard scenario, where a single agent's goal is to find the shortest path; for example, the A* algorithm is a well-known solution to many applications, as are MTS, D*Lite, RTAA* or Abstraction MTS to name a few. Moreover, these issues have been extended to PAMT or MAPF [51] problem scenarios. The problem of multiple agents is known to be an NP-hard to solve optimally [14, 199]. Alongside this consider a scenario where five pursuing agents are present, their adjacent states are not occupied, and each agent can make a move in orthogonal directions. The possible joint actions at a one-time step are equal to $4^5 = 1024$ [24].

One of the alternating approaches to solve pathfinding problems for multiple pursuers is to use an assignment strategy algorithm prior to navigating towards the targets. The distance measures pose a very high impact on the performance level of the strategy, while an efficient measure would help the agents fast approach the targets. The study in this chapter investigates more effective distance measures encompassing multiple criteria. A variety of cost functions have been considered in this study to ascertain the efficiency of each proposed composite criterion subject to various environmental configurations. Twin-cost, Cover-cost and Weighted-cost criteria are investigated in this chapter. It is found that these criteria are more efficient than the existing best-known criterion, which has been commonly used in the literature.

The rest of this chapter is organised as follows. In Section 4.2, the proposed methods for assignment strategies are presented in the context of multi-agent algorithms and Section 4.3 follows with a description of the experimental setup and a discussion of the analysis. Finally, Section 4.4 draws conclusions from the results obtained.

4.2 Proposed Assignment Strategies

One possible solution to coordinate multiple pursuing agents is to consider the assignment strategy algorithm which is introduced in Section 1.1.3. The

Algorithm 3 Twin-cost Algorithm.

```
1: function computeAssignmentStrategy(combinations)
Input: DistancesSum and MaxDistance
2:     m ←DistancesSum;
3:     n ←MaxDistance;
4:     for each n do
5:         c_t ← m × n;
6:     end for
7:     return c_t;
8: end function
```

pursuers are composed as one entity and a task is delivered that helps to navigate towards the targets quicker and capture them for a successful outcome. Existing assignment strategies, such as Summation-cost and Makespan-cost, are detailed in Section 2.5.2. New proposed methods for the assignment strategy algorithms are introduced in this section. First, Twin-cost and Weighted-cost are presented that use distance for their criteria. Then, it follows to introduce the Cover-cost assignment strategy approach which uses an area of coverage for computing an optimal solution.

Except for the Cover-cost criterion, all existing and new proposed assignment strategies use distance as their measurement. Table 4.1 displays 3 pursuing agents versus 3 targets and all possible six combinations are listed. The *Distance* column is measured while pursuing agents and targets are stationary at the initial position. Column for *DistancesSum* is the sum of all distances while column *MaxDistance* is the maximum distance in each combination, which at the same time represent Summation-cost and Makespan-cost, respectively. The results for Twin-cost and Weighted-cost criteria are described in the sections below.

4.2.1 Twin-cost

Similar to the Summation-cost and Makespan-cost criteria as described in Section 2.5.2, the Twin-cost criterion uses distance cost to obtain the best value for its new assignment strategy. The equation for the Twin-cost criterion is the product cost that uses the values that were computed for the Summation-cost, *DistancesSum*,

and the Makespan-cost, *MaxDistance* from Table 4.1. The result of this product, *DistancesSum* times *MaxDistance*, is the cost for each combination. In situations where the combination costs are equal, then the average of *DistancesSum* and *MaxDistance* is taken. The equation for Twin-cost, c_t, is:

$$c_t = DistancesSum \times MaxDistance \tag{4.1}$$

Algorithm 3 is the pseudo-code for the Twin-cost criterion. When the Algorithm 1 in Section 3.2.1 calls the function *computeAssignmentStrategy()* on line 10, then Algorithm 3 with pre-computed *DistancesSum* and *MaxDistance* performs

Table 4.1: The sample scenario of 3 agents versus 3 targets and agents' distance towards the targets. There are six possible combinations, and each has the sum of distances (*DistancesSum*) and maximum distance (*MaxDistance*). The Twin-cost is *DistancesSum* times *MaxDistance* and Weighted-cost uses a parameter value of 0.5 to *DistancesSum* and 0.5 to *MaxDistance*.

Combi-nation	Agent to Target	Distance	DistancesSum	MaxDistance	Twin-cost	Weighted-cost 50/50
1	$A_1 \rightarrow T_1$	8	39	16	624	27.5
	$A_2 \rightarrow T_2$	15				
	$A_3 \rightarrow T_3$	16				
2	$A_1 \rightarrow T_2$	4	37	17	629	27
	$A_2 \rightarrow T_1$	17				
	$A_3 \rightarrow T_3$	16				
3	$A_1 \rightarrow T_2$	4	40	22	880	31
	$A_2 \rightarrow T_3$	22				
	$A_3 \rightarrow T_1$	14				
4	$A_1 \rightarrow T_1$	8	41	22	902	31.5
	$A_2 \rightarrow T_3$	22				
	$A_3 \rightarrow T_2$	11				
5	$A_1 \rightarrow T_3$	20	48	20	960	34
	$A_2 \rightarrow T_1$	17				
	$A_3 \rightarrow T_2$	11				
6	$A_1 \rightarrow T_3$	20	49	20	980	34.5
	$A_2 \rightarrow T_2$	15				
	$A_3 \rightarrow T_1$	14				

the steps on lines 4-6 and returns, c_t, the best result on line 7. Lines 2 and 3 are assignment operations, therefore they are required *one* time, however, line 4 is a loop that needs k operations. Line 5 is the statement inside the loop and it requires the same k time. Since the algorithm has only one loop to run its results for each combination, the frequency of the execution of the statement displays the complexity of $\mathcal{O}(k)$.

Consider a case where only Combination4 and Combination6 from Table 4.1 are present. For the Makespan-cost criterion, the *MaxDistance* with the lowest cost value of 20 would be the result. But, when the *DistancesSum* and *MaxDistance* are multiplied, the results are 902 for Combination4 and 980 for Combination6. Although Combination6 has a lower *MaxDistance* and it is the best choice for the Makespan-cost criterion, in multiplication Combination4 has a better result. Therefore, for the cost of two-time steps, the optimal choice is Combination4 with respect to the Twin-cost criterion.

In the situations, when there is a tie-breaker needed, then the average of *DistancesSum* and *MaxDistance* is taken. Imagine a conflicting result where the first combination has 20 steps for *DistancesSum* and 9 steps for *MaxDistance* while the second combination has 30 steps for *DistancesSum* and 6 steps *MaxDistance*. Now it is clear that the product of both combinations returns the same result of 180 steps for the Twin-cost criterion. However the average of *DistancesSum* and *MaxDistance* in the first combination is 14.5 and in the second combination is 18. Therefore, the first combination returns the results.

4.2.2 Weighted-cost

This is the criterion relevant to the problems where both *DistancesSum* and *MaxDistance* costs need to be considered. When all distances are computed and their combination values are obtained for each agent, then the Weighted-cost criterion allocates predefined weight values for both *DistancesSum* and *MaxDistance* depending on the ratios provided by the assignment strategy. For instance, it is possible to allocate $w_1 = 0.25$ and $w_2 = 0.75$, where the total of w_1 and w_2 is equal to one. The equation for the Weighted-cost criterion, c_w, is:

Algorithm 4 Weighted-cost Algorithm.

1: **function** COMPUTEASSIGNMENTSTRATEGY(combinations)
Input: DistancesSum and MaxDistance
2: initialise w_1 and w_2 ▷ w is a weight parameter
3: $m \leftarrow$DistancesSum;
4: $n \leftarrow$MaxDistance;
5: **for each** n **do**
6: $c_w \leftarrow (m \times w_1) + (n \times w_2)$;
7: **end for**
8: **return** c_w;
9: **end function**

$$c_w = (DistancesSum \times w_1) + (MaxDistance \times w_2) \tag{4.2}$$

The overview of the Weighted-cost criterion approach is presented in Algorithm 4. The pseudo-code provides the steps to compute the best assignment strategy for agents based on the weight inputs on line 2. The equation on line 6 returns the values which are compared to get the lowest result for this criterion. Similar to Twin-cost, this criterion requires two operations with *one* time in lines 3 and 4. Then it iterates the loop for k times at line 5, and so does the statement in line 6. Thus the complexity is processed in $\mathcal{O}(k)$.

One example can be a taxi driver who has a plan to drive fast to get to the goal destination quicker, the Makespan-cost criterion, but needs to consider shorter routes to get there at the same time, the Summation-cost criterion. Another example to compute the cost of equal ratio 50/50 is shown in Table 4.1 where it displays the results for the Weighted-cost criterion with 0.5 for *DistancesSum* and 0.5 *MaxDistance* in the last column. In contrast to the Makespan-cost criterion (*MaxDistance*), where the results show that Combination1 is the best option, the Weighted-cost criterion displays slightly better results for Combination2.

4.2.3 Cover-cost

All previous criteria, existing ones, the new Twin-cost criterion and the Weighted-cost criterion use a distance to compute the best assignment strategy for the multiple agents. This Cover-cost criterion proposes a different approach which

Algorithm 5 Cover-cost Algorithm.

```
1: function computeAssignmentStrategy(combinations)
2:     compute state to cover                          ▷ labels "covered"
3:     mark state as agent covered
4:     get costValue for marked states
5:     append costValue to coverCost list
6:     for each coverCost c do
7:         c_c ← max(c);                ▷ gets maximum combination cover cost
8:     end for
9:     return c_c;
10: end function
```

is not to use the cost of distances, but instead, use the area of coverage. Each pursuer, while idle, before the test run starts, expands all possible states before it successfully reaches the target at the current position. Then each expanded state on the map is marked as "covered" for the pursuer and all other states are "uncovered" if not an obstacle. This is similar to the expansion of the states in the breadth-first search algorithm [205] or the Dijkstra algorithm [206]. When these areas are computed, every pursuer's covered area value is averaged per combination, and the combination with maximum coverage area is the optimal combination to assign targets to the pursuers.

The Cover-cost Algorithm 5 gives the pseudo-code for these steps. Similar to

Table 4.2: The scenario of two pursuing agents, A_n, versus two targets, T_n, and the individual expanded states that are labelled "covered" by the pursuers towards the targets on the AR0509SR gaming map. There are two possible combinations, and each has a percentage of covered area (*CoveredArea*), and the results are averaged within the combination (*CombinationCoverage*).

Combination	Agent to Target	Covered Number of States	CoveredArea	Combination-Coverage
1	$A_1 \rightarrow T_1$	274	18.2%	18.4%
	$A_2 \rightarrow T_2$	279	18.6%	
2	$A_1 \rightarrow T_2$	490	32.6%	39.5%
	$A_2 \rightarrow T_1$	696	46.3%	

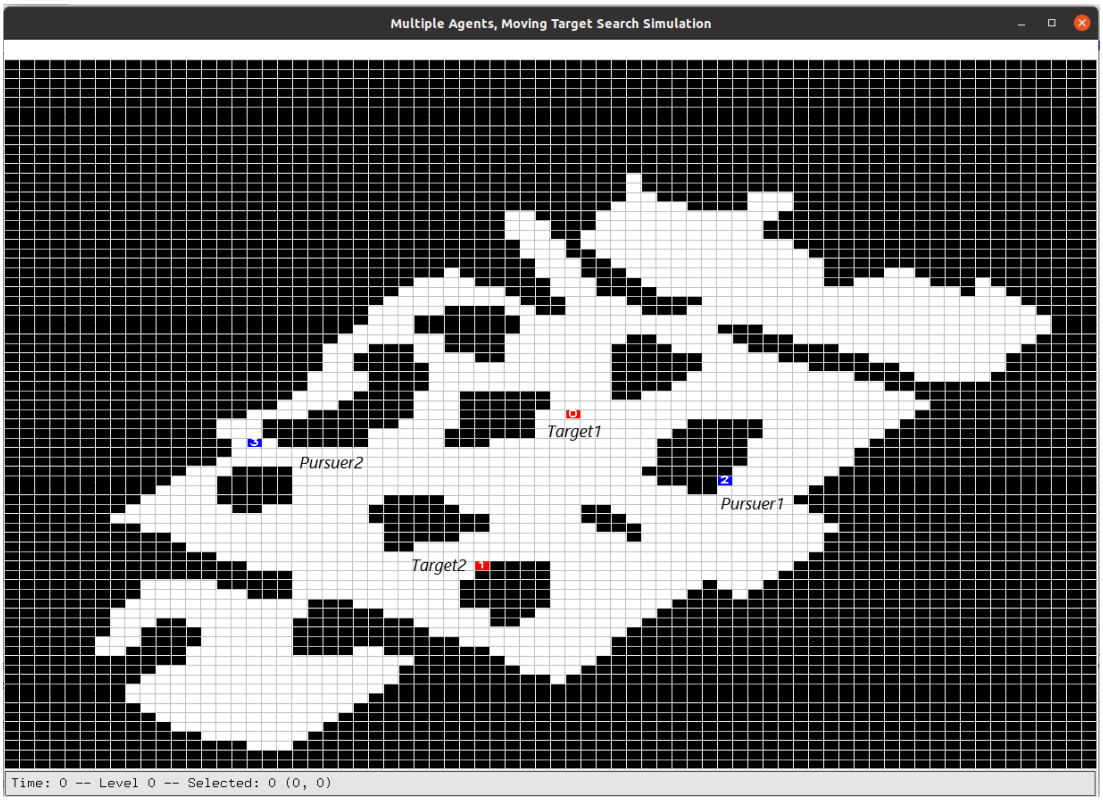


Figure 4.3: A benchmarked AR0509SR map from Baldur's Gate video game. There are two pursuing agents and two targets dispersed on the map.

the previous criteria, when Algorithm 1 calls the function on line 10, *computeAssignmentStrategy()*, then Algorithm 5 returns the optimal result from all possible combinations. First, it computes all available states for each pursuer in line 2 and marks each state as it is covered by this pursuing agent in line 3. Then, in line 4, the percentage of covered areas is computed and added to the list in line 5. To return the final output, lines 6-8 loop through the list and provide the maximum coverage.

The time complexity of Algorithm 5 is analysed. Line 2 is the expansion of the states which has a linear complexity [60] and requires k times. Similarly, lines 3 and 4 need k time and line 5 is the statement that generates *one* time. However, line 6 is the loop that does not compute distance, instead focusing on the covered area which requires k times. The assignment statement inside the loop, in line 7, needs k times, too. Therefore, the Covered-cost criterion displays

the same complexity of $\mathcal{O}(k)$ as the Twin-cost and the Weighted-cost criteria.

Table 2.5.3 in Section 2.5.3 details map environments and displays the total number of empty states to expand for each map in the *Empty States to Expand* column. For instance, the AR0509SR map illustrated in Figure 4.3 contains 1503 empty states and has 4 players (2 pursuing agents vs 2 targets). Table 4.2 displays possible two combinations for these two pursuers. The table contains the sum of all states that are labelled "covered", *Covered Number of States*, for each pursuer (A_n,T_n). The *CoveredArea* is the percentage of the states on the map, that is Covered Number of States divided by the total of empty states *Empty States to Expand*. This results in the equation:

$$CoveredArea = \frac{Covered\ Number\ of\ States \times 100\%}{Empty\ States\ to\ Expand} \tag{4.3}$$

The last *CombinationCoverage* column is the average for each combination. The combination with the highest *CombinationCoverage* percentage is assigned to the pursuers. Combination2 has the highest value in this instance, and hence it is chosen over Combination1.

4.3 Experimentation and Discussion

This section contains empirical results for the existing and proposed approaches. The experimental setup is described first and followed by performance results for the proposed new methods.

4.3.1 Experimental Setup

The general setup of experiments such as the map environment, player combinations, the movement direction of players, the cost of moves, the number of test runs for each configuration and the total number of individual tests as well as the computer settings are detailed in Section 2.5.3.

All new assignment strategy approaches are compared against the existing overall best assignment strategy, the Mixed-cost criterion. The Twin-cost or the Cover-cost criteria do not take any parameters, however, the Weighted-cost criterion employs pre-defined weight parameters. For the experiments in this

study, the ratio of 75/25, 25/75 and 50/50 was used for *DistancesSum* and *MaxDistance*, respectively. In the first setting for the ratios, the strategy considers more weight on the *DistancesSum* than the *MaxDistance*, while the second setting is the opposite strategy where the *MaxDistance* considers more weight than the *DistancesSum*. The third setting considers the equal weight strategy for both values. These weight parameters are analysed to compute the best possible combination for pursuing agents.

The pursuing agents use the STMTA* algorithm and the targets use the SF algorithm. The initial scenario is configured with 4 vs 2. The number of players is increased by 1 for each side in the next combination.

There are five different starting positions for agents and targets. The first starting position is the same state for agents and at a far distance, it is the same state for targets. The second starting position is to locate all agents in the centre of the map, if possible and spread the targets around the corners of the map. The position of agents on other sets was randomly selected on the opposite side of each other.

4.3.2 Performance Analysis

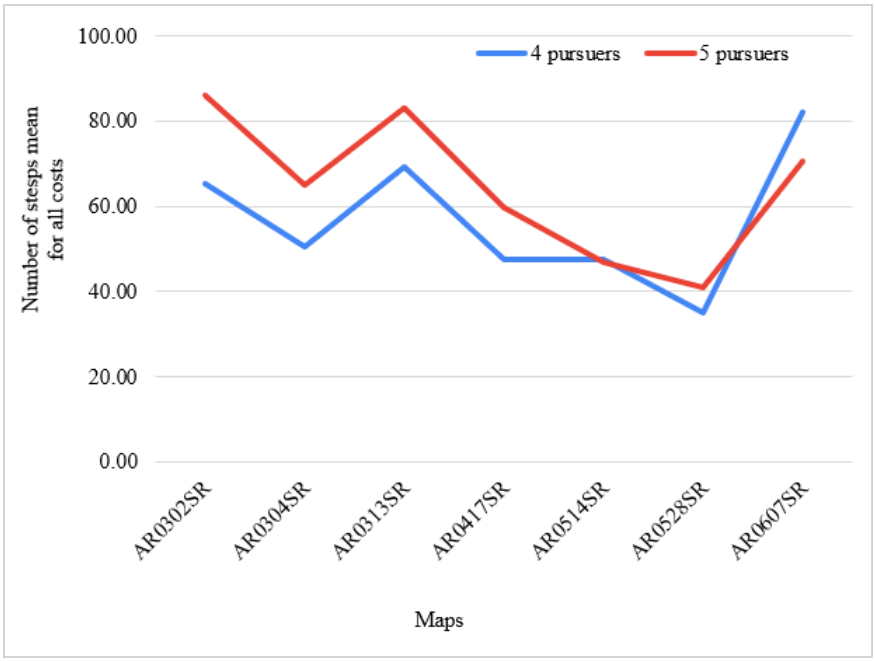
The results are presented as a comparison of assignment strategies for multiple agents. Performance measures the average number of steps travelled for all agents on a single map and the success of completeness of the path. The timeouts are excluded from the results, instead, the percentage of the *hit rate* is provided. The results are compared for Mixed-cost, Twin-cost, Cover-cost and Weighted-cost criteria.

The evaluation of assignment strategies and their comparison is displayed in Table 4.3. The *means* represent the number of steps travelled for each test set and the *hit rate* indicates the percentage of successful runs, meaning at least one of the agents occupying the state of the targets before the timeout. At the bottom of the table, the results are averaged for all maps. Although there is not a big difference, the Cover-cost and the Twin-cost criteria perform slightly better in the 4 pursuers vs 2 targets (4 vs 2) scenario and only Twin-cost shows positive results for the 5 pursuers vs 3 targets (5 vs 3) scenario.

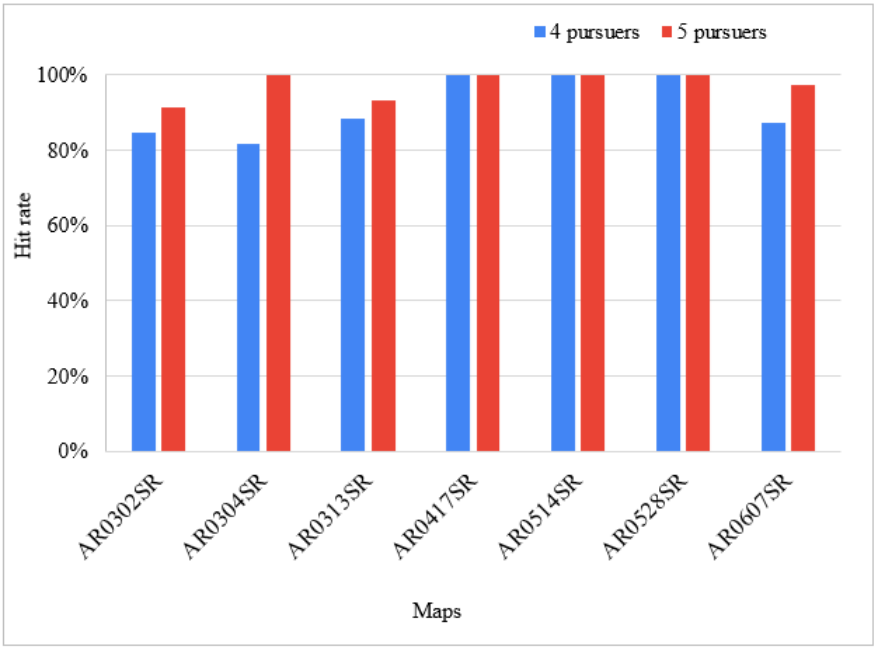
In Table 4.3 the results show that the Mixed-cost criterion has been outperformed in every case on each map. The overall *means* for the number of steps travelled display that the Weighted-cost criterion is successful with the lowest results, although with a different set of parameters for each scenario, Weighted-cost 75/25 on 4 vs 2 and Weighted-cost 25/75 on 5 vs 3.

Table 4.3: The *means* for the number of steps travelled, the ratio of successful test runs and their standard deviations for all maps in all configuration settings. The bottom of the table is the average results of all maps.

	4 pursuers vs 2 targets						5 pursuers vs 3 targets					
	Mixed-cost	Twin-cost	Cover-cost	Weighted-cost 75/25	Weighted-cost 25/75	Weighted-cost 50/50	Mixed-cost	Twin-cost	Cover-cost	Weighted-cost 75/25	Weighted-cost 25/75	Weighted-cost 50/50
Map	AR0302SR											
Means	67.55	65.62	64.93	65.03	65.54	62.84	90.11	87.77	73.93	89.35	89.80	85.44
Std. dev.	9.66	10.35	5.38	9.66	9.95	9.20	14.41	12.35	10.56	12.85	15.42	15.84
Hit Rate	84%	87%	90%	80%	83%	83%	93%	92%	88%	94%	91%	91%
Map	AR0304SR											
Means	51.44	52.20	52.43	41.57	53.13	52.61	64.92	64.11	69.73	64.24	63.74	64.08
Std. dev.	4.21	5.05	5.46	4.54	4.96	4.52	8.05	7.21	5.77	7.31	6.42	6.69
Hit Rate	83%	84%	79%	79%	82%	83%	100%	100%	100%	100%	100%	100%
Map	AR0313SR											
Means	69.33	68.31	69.91	68.65	69.45	70.39	83.27	81.92	85.26	79.77	84.47	83.90
Std. dev.	14.65	13.09	10.15	12.61	14.21	12.63	17.65	14.19	15.87	17.09	14.13	14.40
Hit Rate	89%	87%	86%	91%	87%	91%	91%	94%	93%	92%	95%	95%
Map	AR0417SR											
Means	47.10	44.97	46.55	49.38	47.77	48.95	63.99	64.51	58.28	60.61	56.06	55.04
Std. dev.	14.39	8.91	15.65	11.33	9.71	12.27	24.22	37.50	19.68	22.23	20.65	17.15
Hit Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Map	AR0514SR											
Means	49.27	48.47	43.08	49.39	48.81	47.23	46.57	48.12	49.03	46.11	45.28	45.79
Std. dev.	9.09	8.89	3.77	10.19	8.82	8.46	9.42	8.29	6.91	7.63	6.95	7.89
Hit Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Map	AR0528SR											
Means	34.75	34.91	33.29	36.24	35.56	34.91	40.47	38.32	46.36	42.97	38.16	39.90
Std. dev.	7.26	7.45	2.69	7.41	7.50	7.30	11.09	9.44	6.58	10.81	7.92	6.80
Hit Rate	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Map	AR0607SR											
Means	80.78	83.14	83.82	78.00	82.88	84.95	65.42	65.30	80.83	73.91	65.55	71.78
Std. dev.	18.67	23.49	21.02	18.51	20.29	21.83	15.66	18.45	27.00	23.55	16.20	22.90
Hit Rate	83%	85%	91%	89%	88%	89%	99%	100%	93%	97%	99%	95%
Means of all	57.17	56.80	56.29	55.47	57.59	57.41	64.97	64.29	66.20	65.28	63.29	63.70
Std. dev. of all	11.13	11.03	9.16	10.61	10.78	10.89	14.36	15.35	13.20	14.49	12.53	13.10
Hit Rate of all	91%	92%	92%	91%	92%	92%	98%	98%	96%	98%	98%	97%



(a)



(b)

Figure 4.4: The illustrated graphs display the number of steps mean for all assignment strategy costs per map (a) and the success rate of completed test runs per map (b).

AR0302SR, AR0304SR, AR0313SR and AR0607SR maps are either larger in dimensions or have more nodes to expand. These maps are more difficult to navigate and provide lower rates of success, i.e., capturing the targets before the timeout, see Figure 4.4(b). The results suggest that increasing the number of agents from 4 to 5, increases the success rate and it never drops. The great improvement is seen on maps: AR0304SR with an average *hit rate* from 82% to 100% and AR0607SR with an average *hit rate* from 87% to 97%, as depicted in Figure 4.4(b). The graph illustrated in Figure 4.4(a) demonstrates the improvement in the number of steps, averaged per map when compared with 4 vs 2 and 5 vs 3 players. More experiments with scenarios on 4 vs 3 and 5 vs 2 test combinations for each assignment strategy have been conducted, but due to space limitations, and similarities in the outputs, the results are not outlined in this study.

Standard deviation is taken as the third metric alongside *means* and *hit rate* to indicate the spread of data with which the particular mean value is calculated. The lower the standard deviation, the better the performance is. The results in Table 4.3 show the standard deviation remains steady and helps differentiate the performances from one another.

Overall, the experimental results show that the proposed new assignment strategies help to succeed and in some individual cases improve by approximately 23% especially when the *means* are the same. The reduction is seen in the number of steps, even if the number of agents increases.

4.4 Conclusion

This chapter has investigated and identified more new alternative methods for assignment strategies in multi-agent scenarios in order to increase efficiency. The proposed methods such as Twin-cost, Cover-cost and Weighted-cost criteria have been experimented with, and performance analysis measures the number of steps travelled and the success of completed test runs on grid-based gaming maps. These findings highlight the potential usefulness of these methods and evaluate the strengths and weaknesses during the experiments. It was suggested that the use of the proposed criteria makes the algorithms more efficient than those

currently used including the Mixed-cost criterion. The results suggest that the Weighted-cost criteria depending on parameters have performed the best.

Chapter 5

A Strategy-Based Algorithm for Moving Targets in an Environment with Multiple Agents

5.1 Introduction

There has been a large amount of research on search algorithms for many years. The study and development of search algorithms were based on the basic scenario of a single agent that is tasked with finding a target or goal state on a graph within minimal time. Each search algorithm has its own purpose and need. Even in a simple, static environment, the pathfinding search algorithm faces several challenges. In complex environments, more challenges arise. With various assumptions of this single agent with a single target, the scenario can be relaxed, leading to more difficult problems: there can be several pursuing agents that need to coordinate their search, assigning strategy to the agents before following targets, there can be multiple targets, all of which need to be caught, and targets can move on the graph over time rather than be in a fixed position.

Besides a more standard pathfinding search for a single agent pursuing a single target on a static map, the case could be complicated by an increase in the

number of agents or dynamic changes in the environment. For example, in the scenarios with moving targets, the target algorithms also play an essential role in developing multi-agent scenarios, but they are less studied. The goal of such algorithms is to evade capture as long as possible.

Consider a pursuit and evasion game, where players could be human or computer-controlled. Other examples are video games such as Grand Theft Auto and Need For Speed where both sides of players can be controlled by the algorithms or a flight simulation application where computer-controlled targets are needed to catch or shoot [207]. To make the game more interesting, intriguing, and challenging, the targets need to behave intelligently. Therefore, good target algorithms are an essential factor in improving the gaming experience.

Target algorithms that exist usually have strategies such as maximising the escaping distance [53], random movements to selected, unblocked positions to evade the capturer [3] or, in a state-of-the-art approach called TrailMax, maximising the survival time in the environment by considering the potential moves of pursuing agents on each time step [78].

MAPF problems have been analysed in detail in the literature [170]. These problems are known to be NP-hard [14]. An example of such a problem in a video game is when all non-player agents need to navigate from a starting location to the goal location on a conflict-free route in a static or dynamic environment [62]. Algorithms developed for moving, in other words escaping targets, can make the empirical study of MAPF problems more meaningful, useful, and challenging. Thus, how the existing ones can be improved? An algorithm is introduced based on TrailMax that can be used for multiple moving targets to flee from multiple agents in a dynamic environment. A good design of such an algorithm can help targets to escape more intelligently, rationally and in a human-like manner.

This study considers more testing scenarios against more pursuer strategies, target algorithms, benchmarked maps, player combinations and improving the cost while the target expands pursuers' nodes. Empirical evaluations report different performance metrics, such as capture cost, success rate, computation time and statistical analysis for the significance of the findings.

In the remaining parts of this chapter, Section 5.2 describes the new approach

to the problem. Empirical comparisons are described in the subsequent Section 5.3 which follows the discussion in Section 5.4 and the conclusion is derived in Section 5.5.

5.2 Multiple Pursuers TrailMax: Proposed Approach

The proposed new target algorithm is described in this section. First, the introduction is given for the algorithm, then it follows with pseudo-code and the section finalises with further improvements.

When the problem was described in Section 5.1, it was stated that a smart target algorithm is very useful. In simple scenarios where a single agent pursues one target, the target would know from which agent it needs to escape, as there is only one. Some of the strategies to run away from the agent have been discussed in Section 2.4. But if a situation is considered where multiple targets need to escape from the current state and move to the safest destination in the dynamic environment, how would targets know which pursuing agent they need to avoid for a successful run? For example, the SF algorithm can flee from the closest pursuer but sometimes could run into other pursuers. What would be a smart move for a target while avoiding capture if there are many pursuers?

Although the TrailMax algorithm is a state-of-the-art algorithm, it has been designed to work with only one agent, meaning a target does not have any strategy to escape from one pursuer and avoid another approaching pursuer at the same time. For this specific reason, a target algorithm that would be able to identify approaching multiple agents and escape from all pursuers, a novel algorithm, called Multiple Pursuers TrailMax (MPTM), is developed.

The MPTM algorithm uses a similar methodology as TrailMax but is enhanced for MAPF or PAMT problems. There are two possible benefits that could come from extending TrailMax to multi-agent pathfinding problems. First, the target can identify the state location of other targets and collaborate with them. Second, it can ensure the escape is not only from one pursuing agent but from any approaching invading agents. Here the focus is on the second

issue. It is exhaustive, meaning it considers all possible moves from the agents. Therefore, it is relatively computationally intensive and provides a solution if one exists.

5.2.1 The MPTM Algorithm

The pseudo-code for the MPTM algorithm is depicted in Algorithm 6. First, the current locations of all players (pursuers and target) need to be initialised in line

Algorithm 6 The Multiple Pursuers TrailMax Algorithm.

```
1: function MultiplePursuersTrailMax()
2:     initialise position for all players (pursuers and target)
3:     initial cumulative cost c ← 0 for each player
4:     add target to target_node_queue
5:     add pursuers to pursuer_node_queue
6:     target_caught_states ← 0
7:     if target is not captured then
8:         while target_node_queue not empty do
9:             c_t ← get c from target_node_queue
10:            c_a ← get c from pursuer_node_queue
11:            if (c_t ≤ c_a) then
12:                remove target from target_node_queue
13:                if target not in target_closed and pursuer_closed and parent node not in pursuer_closed then
14:                    insert target into target_closed
15:                    append target neighbours onto target_node_queue
16:                end if
17:            else
18:                for each p_i of players do
19:                    get state s_i for p_i
20:                    if s_i is pursuer then
21:                        c_a ← get c on pursuer_node_queue
22:                        remove p_i from pursuer_node_queue
23:                        if p_i not already in pursuer_closed then
24:                            insert p_i into pursuer_closed
25:                            if p_i in target_closed then
26:                                increment target_caught_states
27:                                if target_caught_states is equal to size of target_closed then
28:                                    return true
29:                                    append p_i neighbours onto pursuer_node_queue
30:                                end if
31:                            end if
32:                        end if
33:                    end if
34:                end for
35:            end if
36:        end while
37:    end if
38:    generate target_path
39:    if target_closed not empty then
40:        reverse target_closed
41:    end if
42: return target_path
43: end function
```

2. The next step is to group all players according to their role and append their positions into the relevant queues, all pursuers to the *pursuer_node_queue* and a target to the *target_node_queue*. At this point, all players will have a cumulative cost of zero, lines from 3 to 5. To make it easier to follow the code, each movement cost will be equal to one unless it is in wait action, then it is zero. This is with the assumption that there is no octile distance. However, the algorithm works with different speeds and distances.

The algorithm has four different lists. The first two lists, *target_node_queue* and *pursuer_node_queue*, contain expanded and visited nodes, such as the current state or neighbouring states for both target and pursuers. The next two lists, *target_closed* and *pursuer_closed*, contain states that are already visited and occupied by players.

Since this is the target algorithm, in line 7, it starts first to check if it is already caught or not. Then loops through if there are any target nodes in the *target_node_queue*. As this is the first step, it only contains the target's current position. Then, it computes the cumulative cost c, the highest value, for target c_t and pursuers c_a at lines 9 and 10. If the c_t is lower or equal to the c_a, then the target expands its nodes, line 11.

During the expansion of nodes for targets in lines 12–15, first, the target node is removed from the *target_node_queue* and placed inside *target_closed* if it is not already in the list and not in the *pursuer_closed* list. It also checks if the target's parent node is not in *pursuer_closed*. The target loops through its available adjacent neighbours and adds them to the *target_node_queue*. These steps are iterated until no state is left to expand. The nodes are expanded like in breadth-first search [205], first-in-first-out.

When the target c_t is higher than c_a, the condition on line 11, the pursuers take the turn and start to expand their nodes. The main part of this algorithm is the lines between 17 and 35, where each pursuer loops through its state and expands its nodes independently from other pursuers. The target needs to know the position of pursuers' states and loops through each player. If it is a pursuing agent, then this particular agent will be removed from *pursuer_node_queue* and placed inside *pursuer_closed* if not already in. The neighbours will be added to the *pursuer_node_queue*. Any state visited by the pursuer exists inside the

target_closed list, then *target_caught_states* is incremented and compared to the size of *target_closed*, which returns true if equal.

Lines 38–42 generate a path. The last element in *target_closed* is the furthest state that the target could move to. This list is reversed to identify the route, and the first element in the list is the action that the target takes. The function repeats every time step to find the best action for the target.

This turn-based expansion goes to the point where all states on the map have been occupied either by the target or the pursuers. The target could only win if its state is not taken by any pursuers until the timeout. For multiple targets, the algorithm runs on each target, and normally, each will get a different outcome based on its location. The result will be the same if they are all in the same state. Even if the starting position is different, the targets could join their path if that is the optimal option.

Although the MPTM algorithm is similar to breadth-first search in expanding its states, it additionally requires computing the cost and comparing each iteration to the cost of the pursuers. First, line 2 requires k times to initialise the position of pursuers and targets. Line 3 is an assignment statement to each pursuer which needs k time operations. Lines 4, 5, and 6 are the statements that require *one* time. Next, line 7 is a conditional statement that needs *one* time, however, it contains the main part of the algorithm. This follows with a while loop in line 8 and this requires k time operations. Line 9 and line 10 are the assignment statements which also need k times. Then another conditional statement to compare the costs and needs k time in line 11. The statements in lines 12, 13 and 14 require k time operation, however, line 15 generates l operations for target neighbours, therefore, the required time is kl. When the condition in line 11 is not met, then line 18 is checked and requires l operations for each player which makes its complexity of kl times. The statements in lines 19 to 28 all require kl time of operations, too. Meanwhile, line 29 needs n operations for each neighbour of the pursuer and this makes kln time in total operations. Lastly, the line 38 generates a path with k time. Line 39 is a conditional statement with *one* time and line 40 requires k time operations. The final result is generated with *one* time in line 42. Hence, the worst-case time complexity of the MPTM algorithm is $\mathcal{O}(kln)$.

5.2.2 Further Improvements

The strategy of TrailMax works for one-to-one agent scenarios, and getting the best cost from the list for each player is straightforward. But this is not the case for the MPTM algorithm as it considers many pursuing agents in one search. The *pursuer_node_queue* contains information for all pursuers and their moving directions with costs.

It has already been discussed that the initial cost is zero for all players. When line 11 is called, it will be true, and the target will take turns to expand and increase its cost by one. On the next iteration, this condition will be false, as the cost for the target is 1, and all pursuers' cost is still zero. The expansion takes place for pursuers. As there are many pursuers, line 21 will request the first pursuer's cost from the *pursuer_node_queue*. Then this pursuer will expand and increase its cost to 1. There is a problem here because TrailMax requests the best cost for each iteration. It would have been fine if there was only one pursuer, but this is an issue with multiple pursuers. If the best cost was considered for multiple pursuers, then only the first pursuer would be expanded as only its cost would be incremented. This leads to the fact that only the same pursuer is requested with the best cost and all other pursuers are left without expansion with an initial cost of zero.

To fix the above problem, the cost requested on lines 10 and 21 is not the best cost but a cost for each pursuer in order of from the *pursuer_node_queue*. This gives a greater opportunity for a target to evaluate all pursuers' moves and make decisions more accurately. Another enhancement is that MPTM does not only consider and run away from the closest pursuing agent but takes into consideration all pursuers on the map by checking each pursuer's state on line 18.

5.3 Empirical Evaluations

The empirical results will be presented in this section to demonstrate the efficiency of the proposed algorithm. First, the experimental setup will be described, and then, the performance results of the MPTM algorithm described in the previous Section 5.2 will be reported.

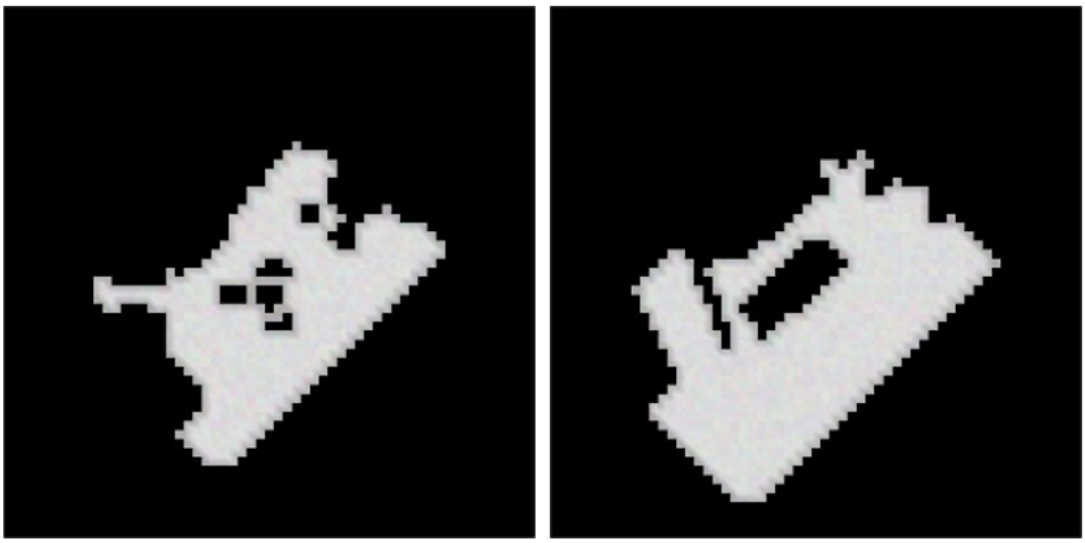


Figure 5.1: The experimented sample maps, (a) AR0311SR and (b) AR0507SR, are used in the Baldur's Gate video game.

5.3.1 Experimental Setup

The general setup of experiments such as the map environment, player combinations, the movement direction of players, the cost of moves, the number of test runs for each configuration and the total number of individual tests as well as the computer settings are detailed in Section 2.5.3.

The evaluation of the MPTM algorithm is tested and compared against Greedy, Minimax and SF algorithms. The pursuing agents use PRA* and STMTA* algorithms. The assignment strategies algorithms such as Twin-cost, Cover-cost and Weighted-cost 50/50 criteria are introduced in Section 4.2 and they are employed for STMTA*.

The scenarios were chosen to have multiple targets, and for the experiments, initially, two and later, three targets were tested. The combination of pursuers versus targets is displayed in Table 5.1. These scenarios help to understand the behaviour of the MPTM algorithm when targets are outnumbered.

All players are placed at different randomly selected locations on each map. There were two different sets of starting positions. The first set has all pursuers in the same location and all targets in the same location, and targets are positioned

at the farthest distance from pursuers. The second set has all players randomly positioned in disperse, on various walls of the map.

Table 5.1: The average number of steps (the capture cost) for each target algorithm against pursuer algorithms. A larger number is better as it avoids the capture by the pursuing agents.

Player Combinations (Pursuer vs Target)	Target Algorithms	Pursuer Algorithms			
		PRA*	STMTA*		
			Twin-cost	Cover-cost	Weighted-cost (50/50)
4 vs 2	SF	50.44	51.22	51.94	52.73
	Greedy	53.32	50.44	49.80	50.77
	Minimax	73.11	57.38	58.3	57.86
	MPTM	117.30	119.45	125.62	117.92
4 vs 3	SF	52.78	55.98	57.14	55.33
	Greedy	60.78	52.02	51.22	51.63
	Minimax	68.23	59.33	60.43	61.08
	MPTM	130.27	138.97	146.26	132.28
5 vs 2	SF	48.31	49.6	50.00	49.53
	Greedy	52.70	49.55	50.20	50.00
	Minimax	67.59	56.26	55.79	55.05
	MPTM	106.72	100.77	108.36	104.81
5 vs 3	SF	51.31	52.94	54.33	53.45
	Greedy	58.57	54.04	51.66	54.94
	Minimax	63.59	56.36	56.93	56.00
	MPTM	126.92	123.55	133.58	112.23
Mean for all combinations	SF	50.71	52.44	53.35	52.76
	Greedy	56.34	51.51	50.72	51.84
	Minimax	68.13	57.33	57.86	57.50
	MPTM	120.30	120.69	128.46	116.81

5.3.2 Experimental Results

Performance analysis is conducted with respect to three key indicators: (i) the number of steps taken for each target algorithm before being caught, (ii) its success rate and (iii) computation time. The first two measurements are averaged considering all targets, and the time is normalised per step.

During the experiments, each test run finishes when all targets are caught or there is a timeout. If some pursuers already caught their assigned targets, the chase continues as long as there are still uncaught targets. Success for pursuers is achieved when all targets are caught, and the number of steps, until the targets have been caught, is recorded. The success of the targets is to avoid capture or stay on the map for as long as possible.

Capture Cost. To evaluate the MPTM algorithm, a comparison with SF, Minimax and Greedy is displayed in Table 5.1. This measures the performance in terms of the number of steps for all targets. The numbers indicate the mean of steps for target algorithms on all maps.

Table 5.1 displays results for different target algorithms. Each value is the mean of eight tested maps. The proposed MPTM target algorithm offers a much longer stay on the maps for all combinations. This indicates that it avoids capture and demonstrates smarter decisions. The higher number is better.

Some maps have island-type obstacles that allow the targets to escape from pursuers more easily, see Figure 5.1. Although each map has many states to explore, as detailed in Table 2.3, all algorithms managed to find an escape route. SF and Greedy both display similar capture times and their results are close to each other. Minimax is better than SF and Greedy but still not as good as MPTM.

The results compared in Table 5.1 show that for all player combinations, the MPTM algorithm managed to escape all pursuing agents two times longer than Minimax. The same algorithm when compared against SF or Greedy, the results display that on average MPTM manages to run away from the pursuers 2.3 times longer. The graph in Figure 5.2 provides a visual comparison of the times to capture between MPTM and the other three target algorithms.

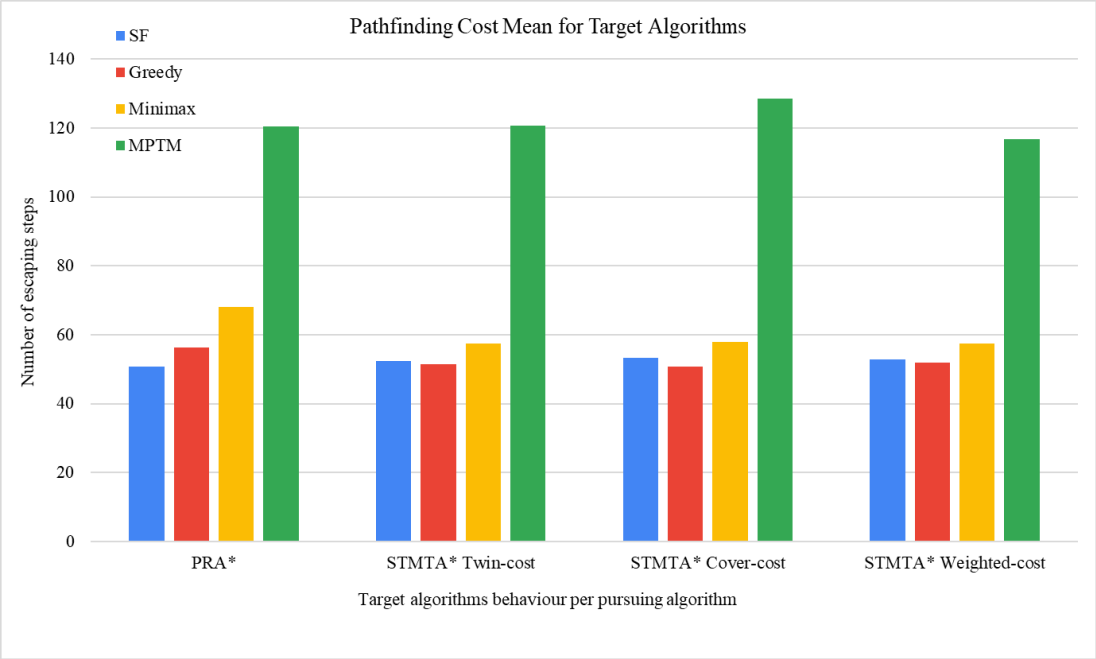


Figure 5.2: The overall comparison of the MPTM algorithm with other target algorithms per a pursuing agent algorithm. The graph displays the mean for all maps and all player combinations.

Comparing scenarios with different pursuing agents and target numbers shows that, as expected, when the pursuer-to-target ratio increases, capture times tend to decrease, while when the pursuer-to-target ratio decreases, capture times tend to increase.

The evidence shows that the new MPTM algorithm outperforms SF, Minimax and Greedy algorithms in the number of steps in all test configurations.

While the experiments were designed to study target algorithms, it is also interesting to note that the STMTA* algorithm with its assignment strategy variations performs overall better than PRA*.

Statistical tests are also used on the capture costs to find out which of the results are significantly different. The proposed MPTM algorithm is compared against existing SF, Greedy and Minimax algorithms. Only the STMTA* Weighted-cost algorithm's results are used for the comparison as it has shown overall the best results among other pursuer algorithms as shown in Table 5.1. The capture costs are not normally distributed; therefore, the statistical results

are obtained using the Wilcoxon Rank Sum tests. A significance level of 0.05 is used. The values obtained from the statistical tests are provided on a map in Table 5.2.

Table 5.2 displays p-values for all eight maps and four different player combinations separately that are used during the experiments. There were two starting positions on each map. Each set of players was aggregated in the first position and in the second position, and all players were dispersed. The results in the table display p-values individually for each starting position.

From these results, it can be seen that the majority of the results display statistically significant differences. p-values presented in Table 5.2, the results below 0.05 indicate significant differences, while there are results that are below 0.01 in the level of significance. Although some results are close significant. Most of the aggregated positions show significance, in contrast to dispersed positions.

It can be concluded that the results of the experiments for capture cost are significant for 0.01 on most of the tests. The findings should make an important contribution to the field of target search algorithms.

Success Rate. Success for the agents is achieved when a pursuing agent gets to the position of the target. In multitarget scenarios, success is achieved when all targets have been captured. For the target(s), success is the absence of agent success. The success rate for algorithms is shown in Table 5.3. The results presented in the table are for four target algorithms against four pursuing agent algorithms for all sets of combinations.

From this Table 5.3, the SF and Minimax algorithm performs the worst, and they always get caught by pursuing agents in any tested combination. The Greedy algorithm shows being caught in every possible test against the STMTA* algorithm and its variations. It also failed against PRA*, but only in one instance, where it managed to succeed when the deadlock occurred. It happened on the 5vs3 player combination. In this particular example, when the pursuers caught one target, instead of approaching and catching the remaining targets, the pursuers kept moving one step back and forward until timeout.

On the other hand, MPTM shows better results in comparison with SF, Greedy and Minimax. Overall, the performance of MPTM is quite well, yet

Table 5.2: Wilcoxon Rank Sum test results (p-values) for MTPM compared against SF, Greedy and Minimax algorithms.

Player combinations (Pursuer vs Target)	Target Algorithms	Maps used from Baldur's Gate Video Game (p-values)															
		Map AR0311SR		Map AR0407SR		Map AR0507SR		Map AR0508SR		Map AR0512SR		Map AR0527SR		Map AR0531SR		Map AR0707SR	
		Position1	Position2	Position1	Position2	Position1	Position2	Position1	Position2	Position1	Position2	Position1	Position2	Position1	Position2	Position1	Position2
4 vs 2	SF	0.0005	0.0006	0.0838	0.0386	3.6E-07	0.3907	0.0010	2.3E-07	4.6E-05	0.3284	0.4404	0.0532	0.0021	0.0630	0.0594	0.0009
	Greedy	0.8597	5.3E-08	0.1866	4.0E-05	3.4E-07	0.0600	7.8E-09	3.9E-08	0.6926	0.0366	7.9E-09	0.7811	5.9E-09	0.1084	9.6E-06	5.2E-09
	Minimax	0.0395	0.0009	0.6398	5.3E-05	4.6E-07	4.3E-06	3.7E-06	5.2E-06	0.1500	0.0045	2.3E-07	0.5473	1.5E-08	0.0003	0.0075	5.7E-08
4 vs 3	SF	1.6E-05	0.0236	0.6354	0.0121	5.6E-07	0.1853	6.6E-06	1.3E-05	0.4311	0.2133	0.0080	0.3486	0.0065	0.0130	0.0528	0.0060
	Greedy	0.0050	0.0013	9.8E-05	2.4E-06	5.7E-07	0.0363	7.9E-09	3.5E-07	0.1892	0.3248	7.8E-09	0.3353	6.8E-09	0.0028	0.0002	0.0134
	Minimax	0.1842	0.0004	0.0110	0.0118	2.8E-07	1.0E-05	2.5E-08	0.0004	0.2319	0.0160	1.5E-08	0.3353	1.3E-08	0.3939	0.0110	3.1E-07
5 vs 2	SF	0.0002	1.2E-05	0.0068	0.0007	4.6E-08	0.7649	1.1E-06	3.5E-08	4.0E-07	0.9455	0.6161	0.0024	0.2908	0.0025	0.3639	5.8E-08
	Greedy	0.0250	4.6E-07	0.0015	0.2622	5.5E-08	0.0872	7.8E-09	1.3E-08	0.9891	0.0858	7.5E-09	0.7806	7.8E-09	0.5570	0.0006	6.3E-05
	Minimax	0.4322	0.0477	0.0070	0.01364	1.3E-08	0.0920	5.9E-08	8.8E-07	0.0202	0.0997	1.1E-07	0.0497	6.7E-08	6.2E-07	0.0156	1.0E-04
5 vs 3	SF	1.8E-06	0.0005	0.5956	0.0289	3.0E-07	0.2380	0.0002	9.6E-06	0.0369	0.8374	0.3935	0.4524	0.0066	0.4874	0.1164	1.1E-05
	Greedy	0.0400	0.0001	0.0004	0.0162	9.5E-07	0.0053	7.9E-09	1.6E-05	0.1549	0.1025	7.9E-09	0.0005	5.9E-09	0.0091	2.1E-05	1.1E-06
	Minimax	0.0197	6.2E-05	0.0018	0.6448	1.1E-07	0.4472	3.0E-08	0.0773	0.0209	0.1036	7.9E-09	0.0005	8.4E-09	0.0016	0.0008	1.1E-06

there are a few instances where eventually gets caught 100% of the time. The graph in Figure 5.3 illustrates how MPTM performed for all test combinations on all maps.

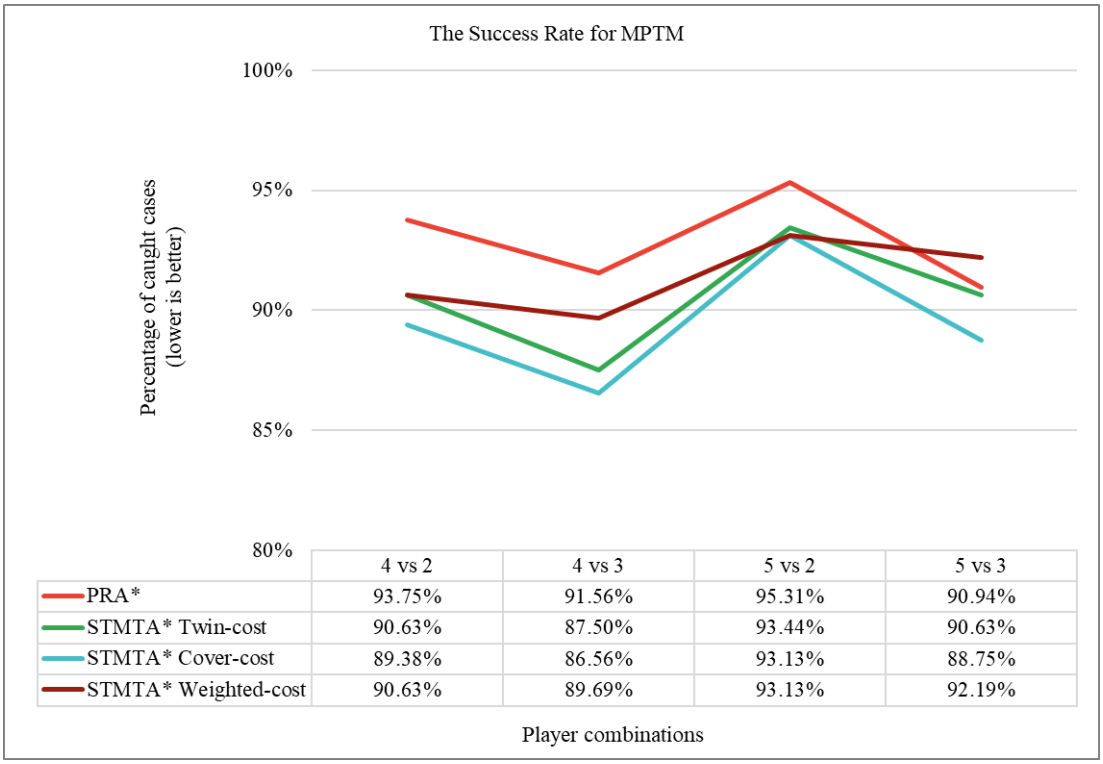
Like capture costs, success rates are also dependent on pursuer and target ratios. The success was proportional to the number of pursuers and targets. More pursuers for the same number of targets increased the captivity. The success rate was increased when the number of targets incremented versus the same number of agents, as displayed on the graph, see Figure 5.3.

The behaviour of the MPTM algorithm is better on the maps that have obstacles that could be navigated around, for example, the maps illustrated in Figure 5.1. These types of maps may be suitable for adaptive target algorithms as they offer opportunities for escape but may be difficult for the pursuing agent algorithms if they do not have strategies such as the trap strategy [202]. The maps AR0311SR, AR0527SR and AR0707SR have dead-ends or blind alleys and thus make it more difficult to find an escape route, leading to lower target performance on these maps.

With some algorithms, pursuing agents sometimes fail to catch the targets, although these are outnumbered. They might catch one target but fail to catch the other, or keep following the target, or end in a deadlock until timeout. This is commonly seen in PRA* as there is no assignment strategy before starting the move, unlike STMTA*.

Table 5.3: The overall success rate of capture for all scenarios. For targets, the lower is better.

Target Algorithms	Pursuer Algorithms			
	PRA*	STMTA*		
		Twin-cost	Cover-cost	Weighted-cost (50/50)
SF	100.00%	100.00%	100.00%	100.00%
Greedy	99.92%	100.00%	100.00%	100.00%
Minimax	100.00%	100.00%	100.00%	100.00%
MPTM	92.89%	90.55%	89.46%	91.41%



	4 vs 2	4 vs 3	5 vs 2	5 vs 3
PRA*	93.75%	91.56%	95.31%	90.94%
STMTA* Twin-cost	90.63%	87.50%	93.44%	90.63%
STMTA* Cover-cost	89.38%	86.56%	93.13%	88.75%
STMTA* Weighted-cost	90.63%	89.69%	93.13%	92.19%

Figure 5.3: The performance rate of success for the MPTM target algorithm for all test configurations and maps. Lower is better.

On average, over all maps per player combinations, the success rate can be 13% better than Minimax, Greedy and SF.

Timing. This section measures the time taken for each algorithm during the same tests that measured the capture cost and the success rate. Each experiment is recorded in seconds and averaged over all tests.

Table 5.4 provides the results for each target algorithm. SF, Greedy and Minimax do not do as much computation as MPTM prior to moving, therefore, their results are smaller and closer to each other in comparison to MPTM, which has greater differences.

To find the best possible action, the MPTM algorithm computes all possible moves for the target and all pursuers on the map, therefore, the computation time is much higher.

5.4 Discussion

Results presented in Section 5.3 show that the MPTM algorithm has a greater chance of escaping from multiple pursuing agents, which has been the main focus of this study. The MPTM algorithm can estimate the possible future movements of the pursuers and therefore, MPTM can function smartly by avoiding capture and fleeing as far as possible until it runs out of all options. This could be similar to cop and robber situations, where the robber is a villain and escapes from the

Table 5.4: The computation time (in seconds) per step for each target algorithm.

Player Combinations (Pursuer vs Target)	Target Algorithms	Pursuer Algorithms			
		PRA*	STMTA*		
			Twin-cost	Cover-cost	Weighted-cost (50/50)
4 vs 2	SF	0.00037	0.00038	0.00038	0.00038
	Greedy	0.00019	0.00008	0.00008	0.00009
	Minimax	0.00166	0.00155	0.00154	0.00156
	MPTM	0.15913	0.16537	0.16234	0.16284
4 vs 3	SF	0.00057	0.00055	0.00055	0.00055
	Greedy	0.00020	0.00011	0.00011	0.00011
	Minimax	0.00141	0.00129	0.00140	0.00138
	MPTM	0.24819	0.24975	0.24949	0.23835
5 vs 2	SF	0.00036	0.00038	0.00038	0.00037
	Greedy	0.00014	0.00008	0.00009	0.00009
	Minimax	0.00171	0.00163	0.00163	0.00160
	MPTM	0.15882	0.15846	0.16146	0.15964
5 vs 3	SF	0.00054	0.00054	0.00054	0.00054
	Greedy	0.00019	0.00013	0.00012	0.00012
	Minimax	0.00151	0.00136	0.00143	0.00136
	MPTM	0.24287	0.25920	0.24776	0.24295

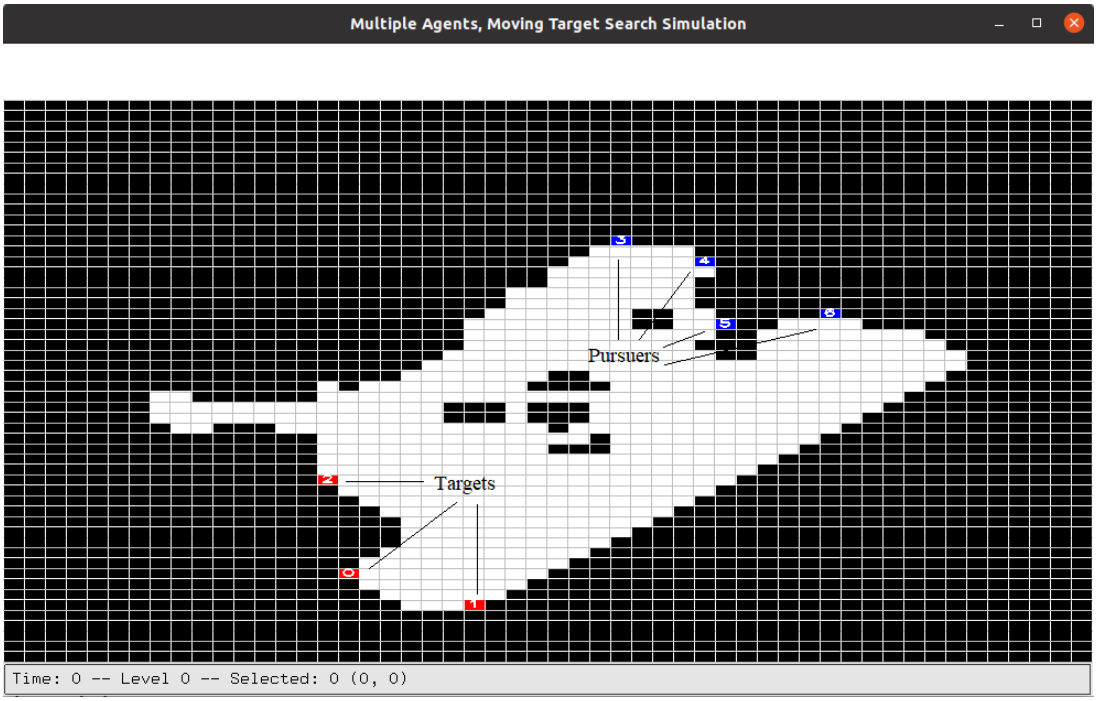


Figure 5.4: The Baldur's Gate benchmarked gaming AR0311SR map with pursuers the targets at the initial position.

cops as illustrated in the simulation gaming map from Baldur's Gate in Figure 5.4. The simulation displays the initial position of four cops (pursuers) and three robbers (targets) on the map.

The proposed MPTM algorithm is measured and compared against SF, Greedy and Minimax algorithms. MPTM offers better results by staying much longer on the maps and managing to escape the pursuing agents. The number of steps is the capture cost, where in some cases the MPTM avoids capture by 2.6, 2.9 and 2.4 times longer than SF, Greedy and Minimax, respectively. Moreover, these results were statistically tested using the Wilcoxon Rank Sum test to establish the significance of the findings. Table 5.2 displays the p-values and with a 95% level of confidence, most of the results indicate significant differences. Another key measurement is the success rate that exceeds expectations for MPTM with 91.08% of being caught, the lower is better, whereas SF and Minimax get caught 100%, and Greedy with 99.98%.

Based on different maps and various player combination settings, the

suggested new algorithm allows for functioning efficiently. Despite MPTM's success rate and outsmarting pursuers, further research is needed to improve the computation process. To avoid exhaustive and intensive computation with larger player combinations and to speed up the search, it might be more beneficial to have a branching factor or window-based search.

5.5 Conclusion

The aim of this chapter was to provide a solution for MAPF or PAMT problems and develop a target algorithm that would consider multiple pursuers and make a smart escape. Numerous interesting studies have been conducted on search algorithms, and among them are solutions to the MAPF frameworks. Only a few studies have been carried out on target algorithms, especially in multi-target environments.

This research shows that TrailMax is a successful algorithm for the control of targets if developed further for dealing with multiple pursuers. The amendments proposed to the TrailMax algorithm made it work as a strategy for multi-agent multi-target search problems in dynamic environments.

The resulting MPTM algorithm has outperformed other target algorithms for the same scenario, and that can make pursuit and evasion scenarios in computer games more challenging, meaningful, and interesting. The results clearly show that the MPTM algorithm performs far better, with at least doubling capture cost and escaping success by 13% on the gaming maps used for benchmarking.

Chapter 6

Adaptive Weighted-Cost Assignment Strategy for Efficient Multi-Agent Path Planning

6.1 Introduction

The necessity and the importance of assignment strategy algorithms were discussed in Chapter 4. In this regard, the sample problem of how pursuing agents might only follow one target in the presence of multiple targets was illustrated in Figure 4.1. The discussion provided solutions and presented three new different criteria, including Twin-cost, Weighted-cost and Cover-cost assignment strategies, in Section 4.2. The Weighted-cost criterion performed the best, and in this study, it is the baseline for the new algorithms. However, there are limitations to the Weighted-cost algorithm as it needs to predefine the weight parameters before the test run. The experiments are conducted on the benchmarked gaming maps [178] and in Section 2.5.3 categorised into three different groups: maps with narrow corridors, circle-shaped obstacles, and large open spaces. Although solving every problem with one algorithm may not be possible, the solutions provided might be sufficient for these configurations.

This chapter's main contribution is a new assignment strategy algorithm, Adaptive Weighted-cost, for multiple pursuers. First, the issues with predefined

weight parameters are avoided and instead, each situation is resolved by adapting dynamically so that the Adaptive Weighted-cost algorithm evaluates and computes the weight parameters independently. The computation of the same weight parameters can be adjusted with a cross multiplication of values for better utilisation of makespan in multi-agent situations. The second contribution is the Joint Weighted-cost and the Joint Twin-cost algorithms, which use the distance and the covered area of agents to find a combination that helps pursuers assign targets to capture them quicker. There are two possible solutions, one is to use the weight formula, and the other one is to use multiplication between the variables. In some situations, because of their position on the map, the targets can have more area covered, more area to escape, and the effort of joining distance and a covered area to find the best combination for pursuers can lead to better task planning. Third, each assignment strategy algorithm can navigate towards the assigned target in the *static assignment mode (s-mode)* or periodically (predefined parameter) stop for an evaluation and re-assign targets in the *dynamic assignment mode (d-mode)*. In the *d-mode*, the pursuers might change the targets, if needed, based on the current position of the players [189].

The rest of this chapter is organised as follows. Section 6.2 briefly discusses the problem and the existing approaches and then introduces new methods for assignment strategies in Section 6.3. A description of the experimental setup, a discussion of analyses and the display of evaluated results are provided in Section 6.4. Finally, Section 6.5 draws conclusions from the findings obtained with suggestions for future works.

6.2 Pathfinding Problem and Current Methods for Assignment Strategies

This section refers to the problem and then describes the pursuer-to-target assignment with suggested enhancement methods. The second part of this section briefly describes existing assignment strategies with their approach and methods used.

6.2.1 Pathfinding Problem for Multiple Agents

Given that the multi-agent pathfinding problem is introduced in Section 2.5.1, it is possible for pursuing agents to follow any target based on their closest distance to the agent as used with PRA* [34]. However, thorough planning for pursuers can increase performance by reaching the targets quicker and successfully capturing them.

The pursuing agents move towards the targets using the STMTA* algorithm which relies on the Assignment Strategy Algorithm 1 to assign targets. Once the targets are assigned to the pursuers, each pursuer computes its route towards the target. The pursuers constantly observe the new positions of the targets and whether any of the targets are caught. This process repeats each time step. If the target is caught by the assigned pursuer(s) on successful runs, then to find quicker sub-optimal solutions, only these pursuers are re-assigned with the existing, not caught yet, targets. This give-a-hand function helps other agents to catch the remaining targets quicker. The STMTA* algorithm has been enhanced and it can change previously assigned targets if it meets the given assignment strategy's criteria. The improvement offers *s-mode* or *d-mode*. In *s-mode*, the targets are assigned only at the beginning of each run and not changed until the capture, whereas in *d-mode*, the pursuing agents are able to re-assess the position of all players and re-assign after a predefined number of steps.

6.2.2 Existing Assignment Strategy Methods

Multiple pursuing agents and multiple targets are present in a given scenario. The pursuers coordinate among themselves using assignment strategy algorithms to assign each target. These assignment strategies then provide the optimum combination to assign targets in order to enable the pursuers to achieve their goal of catching targets most cost-effectively. The research has been taken to provide a solution to find an optimal combination of assignments and the most common methods that have been used are the cumulative distance sum or the maximum time step (makespan). Although Summation-cost and Mixed-cost criteria and their similar approaches are commonly used in the literature as mentioned in Section 2.5.2, there are other new approaches to finding the best cost-effective

combination for multiple pursuers. Twin-cost, Cover-cost and Weighted-cost are the current methods that were introduced in Chapter 4 and the experiments were conducted in comparison with the existing Mixed-cost criterion. The Twin-cost and the Weighted-cost criteria use the distance measurement to obtain the best values, while the Cover-cost criterion uses an area of coverage. Section 4.2 details these proposed assignment strategies and Section 4.3.2 evaluates their performance.

6.3 Proposed New Methods for Assignment Strategies

In the following section, three new assignment strategy algorithms are introduced. These new methods aim to find the best combination to assign targets to multiple agents which leads to better task planning and increases the performance in successfully capturing targets. The first algorithm is the Adaptive Weighted-cost that disregards predefined weight parameters and computes the cost by dynamically evaluating the parameter values. The second is the Joint Weighted-cost and the third is the Joint Twin-cost algorithms that use the sum of distances and the covered area to find a combination. Finally, the section introduces two different navigation modes, the *s-mode* and *d-mode*, for pathfinding search once the experiment starts.

Table 6.1: The scenario of two pursuing agents, A_n, versus two targets, T_n, and the distance from the pursuers towards the targets on the AR0509SR gaming map (Figure 4.3). There are two possible combinations, and each has the sum of distances (*DistancesSum*), and maximum distance (*MaxDistance*).

Combination	Agent to Target	Distance	DistancesSum	MaxDistance
1	$A_1 \rightarrow T_1$ $A_2 \rightarrow T_2$	21 28	49	28
2	$A_1 \rightarrow T_2$ $A_2 \rightarrow T_1$	25 26	51	26

Algorithm 7 Adaptive Weighted-cost Algorithm.

```
1: function computeAssignmentStrategy(combinations)
Input: DistancesSum and MaxDistance
2:     m ←DistancesSum;
3:     n ←MaxDistance;
4:     w1 ← m/(m+n);
5:     w2 ← n/(m+n);
6:     for each n do
7:         csa ← (m × w1) + (n × w2);
8:     end for
9:     return csa;
10: end function
```

6.3.1 Adaptive Weighted-Cost

In the study of assignment strategies in Chapter 4, the Weighted-cost criterion was successful and performed better results in comparison with the approaches tested. The experiments used predefined weighted parameters and the ratios were 0.25/0.75, 0.50/0.50 and 0.75/0.25 for the Summation-cost and the Makespan-cost values (*DistancesSum/MaxDistance*), during the test runs for the Weighted-cost criterion. Although the experimental results displayed promising outcomes, further work is undertaken to improve and optimise the algorithm such that, instead of predefining the parameter values each time, the algorithm adapts this approach and dynamically identifies the weight cost based on the values for *DistancesSum* and *MaxDistance*.

Algorithm 7 outlines the Adaptive Weighted-cost algorithm for multiple agents. Lines 2 and 3 require combination values for *DistancesSum* and *MaxDistance*. The main part of the algorithm is the computation that happens in lines 4 and 5 for w_1 and w_2 parameters, where w_1 is *DistancesSum* divided by the sum of *DistancesSum* and *MaxDistance* and w_2 is *MaxDistance* divided by the sum of *DistancesSum* and *MaxDistance*. The iteration finds the values for all combinations in lines 6-8 and returns the outcome in line 9 for the Adaptive Weighted-cost algorithm. Table 6.1 has testing results from the AR0509SR gaming map (Figure 4.3) for Combination1 where *DistancesSum* is 49 and *MaxDistance* is 28. The dynamically computed ratio for w_1/w_2 is 0.64/0.36.

Although the method differs in how the weight values are determined, it actually has a similar time complexity as Algorithm 4. Thus, lines 2 and 3 are the repeats of the operations. However, lines 4 and 5 are the additional operations that compute weight cost values and since they are assignment operations, they require *one* time. Therefore, the number of steps in the worst-case does not change and has a complexity of $\mathcal{O}(k)$.

The mentioned Algorithm 7 displays a standard method of computing two variables, *DistancesSum* and *MaxDistance*, with dynamically identified weighted parameters. There can be situations with five pursuing agents on the map and they all might have a similar distance towards the targets with 20 time steps each. The Summation-cost for these five pursuers is 100 steps, but the makespan is 20 steps. Because there is a huge ratio difference between these values, it can be adjusted in such a way that the dynamic weight parameters can be swapped. Therefore, the computation for the standard Adaptive Weighted-cost, c_{sa}, in line 7 for Algorithm 7 is:

$$c_{sa} = (m \times w_1) + (n \times w_2) \tag{6.1}$$

This equation will normalise the result and utilise more *MaxDistance*. So, the result for the standard Adaptive Weighted-cost criterion is $(100 \times 0.83) + (20 \times 0.17) = 86.40$, while the adjusted Adaptive Weighted-cost criterion, c_{aa}, is $(100 \times 0.17) + (20 \times 0.83) = 33.60$, with an equation:

$$c_{aa} = (m \times w_2) + (n \times w_1) \tag{6.2}$$

This adjusted method helps to balance off the values especially if there is a huge difference between two values and provides a better combination for pursuing agents in assigning the targets. The Adaptive Weighted-cost criterion's both standard and adjusted methods are implemented and tested during the experiments.

Algorithm 8 Joint Weighted-cost Algorithm.

```
1: function computeAssignmentStrategy(combinations)
Input: DistancesSum and CombinationCoverage
2:     initialise w_1 and w_2                          ▷ w_n is weight parameter
3:     m ←DistancesSum;
4:     n ←CombinationCoverage;
5:     for each n do
6:         c_jw ← (m × w_1) + (n × w_2);
7:     end for
8:     return c_jw;
9: end function
```

6.3.2 Joint Weighted-Cost

Until now, the distance measurement (the sum of distances or makespan) or covered area has been used to assign targets to the pursuing agents and all the above-mentioned Weighted-cost criteria have used distance values for their assignment strategies. Although the distance is an important measurement, it is possible to use the covered area together with the distance as the parameter to compute an optimal solution for assignment strategies. This is a new approach to the assignment strategy where the pursuing agents' distance and covered area join together and use the weight parameter.

The combination of the shortest distance criterion and large covered area criterion is used in this new approach. The pursuing agents need to evaluate the lowest cost of the *DistancesSum* and also consider the covered area together. The Summation-cost criterion already provides the *DistancesSum* value, and the Cover-cost criterion provides the *CombinationCoverage* value. The Joint Weighted-cost criterion is similar to the Weighted-cost criterion mentioned in Section 4.2.2 and it uses the predefined weight parameters that help to compute the best option among all combinations for multiple pursuers. The criterion requires two measurements, *DistancesSum* and *CombinationCoverage*, as displayed in lines 3 and 4 of Algorithm 8. The ratio of the weight parameters in line 2 for this criterion is used at 0.5/0.5 on both variables. Algorithm 8 iterates all possible combinations that are available for the pursuers in lines 5-7 and the best combination with the minimum cost value is returned in line 8.

In each assignment computation *DistancesSum* and *CombinationCoverage* values are required and also have initialisation of weight parameters. Although each of these lines 2, 3 and 4 is *one* time operation, the iteration in lines 5 to 7 generates k number of operations for the optimal assignment combination. Since the manual input of weight parameters does not change the complexity, the time complexity of the Joint Weighted-cost criterion is the same as the Weighted-cost criterion, which is $\mathcal{O}(k)$ per step.

6.3.3 Joint Twin-Cost

When the number of targets is more than one and multiple pursuers require assignment strategies, another way to solve the assignment problem can be using an interaction effect between two variables [208]. The Joint Twin-cost criterion uses two measurements, the sum of distances and the covered area, as in the Joint Weighted-cost criterion above and multiplies them to find the combination value. This approach is similar to the Twin-cost criterion that is mentioned in Section 4.2.1 but instead of the sum of distances and makespan, this criterion uses a joint between the agents' sum of distances, *DistancesSum*, and their covered area, *CombinationCoverage*. Algorithm 9 provides steps for this criterion. Line 2 and line 3 require agents' sum of distances, *DistancesSum*, and the cost of the area covered, *CombinationCoverage*. The product of two variables performs in lines 4-6 and the result is returned in line 7. The drawback of using the *CombinationCoverage* value in Joint Weighted-cost and Joint Twin-cost algorithms is that it increases the computation time as it computes the breadth-first search by visiting every state until it reaches the target.

This newly introduced Algorithm 9 has the same order of steps and the function is similar to the Algorithm 3 that is described in Section 4.2.1. The assignment operations in lines 2 and 3 require *one* time. The number of operations in lines 4 to 6 is not changed, therefore, it is possible to conclude that the complexity of this algorithm is equal to $\mathcal{O}(k)$.

Algorithm 9 Joint Twin-cost Algorithm.

```
1: function computeAssignmentStrategy(combinations)
Input: DistancesSum and CombinationCoverage
2:     m ←DistancesSum;
3:     n ←CombinationCoverage;
4:     for each n do
5:         c_jt ← (m × n);
6:     end for
7:     return c_jt;
8: end function
```

6.3.4 Combinations and Navigation Mode

The above-mentioned algorithms can be tested on a gaming AR0509SR map (Figure 4.3). The experiment was conducted on a configuration with 4 agents vs 4 targets (4 vs 4) and possible 24 combinations are listed in Table 6.2. Pursuing agents are numbered 4-7 and targets are numbered 0-3, and each pursuer is assigned one target with the relevant cost in parenthesis (the first example is the distance cost from pursuing agent P_5 to target T_0 is measured in 26 steps). In the first row, if four distance steps in the parenthesis are summed up, then *DistancesSum* is equal to 129 and the maximum step distance for *MaxDistance* is 43 for the first-row combination. Three algorithms, Weighted-cost, Joint Weighted-cost and Adaptive Weighted-cost, use already generated values (*DistancesSum* and *MaxDistance*) and compute their cost based on their criterion. The results for *CombinationCoverage*, Joint Weighted-cost and Joint Twin-cost use the maximum combination cost values, while the rest use the minimum combination cost. Six possible combinations could be applied during the test runs before the chase starts depending on the provided criterion. The empirical evaluation of the experimental results is in Section 6.4.

The algorithms, Weighted-cost, Adaptive Weighted-cost with the standard and the adjusted method of computation, Joint Weighted-cost and Joint Twin-cost, all assign targets to the pursuers at the *s-mode* before the navigations start and the target does not change until it is caught. The *d-mode* has a predefined, changeable number of steps parameter, that the targets can be re-assigned during the test run if the computation of the assignment strategy suggests a different

combination setting. The behaviour of changing targets in *d-mode* will be tested and compared with its *s-mode* configuration in the next Section 6.4.

6.4 Experimentation and Discussion

This section presents the empirical results of the newly proposed approaches for the assignment strategies. Initially, the setting for the experiments is described

Table 6.2: The AR0509SR map (Figure 4.3) positions dispersedly players (4 vs 4) at the starting state-4 during the experiments. There are 24 possible assignment combinations. Each criterion has its optimal combination. *DistancesSum* (DS), *MaxDistance* (MD) and *CombinationCoverage* (CC) have been shortened.

#	Pursuer-to-Target assignments are in this order of combinations:	DS	MD	CC	Weighted-cost			Joint Weighted-cost	Joint Twin-cost	Adaptive Weighted-cost	
					25/75	50/50	75/25	50/50	$m \times n$	Standard	Adjusted
1	5->0(26), 7->1(43), 6->2(32), 4->3(28),	129	43	0.387	64.50	86.00	107.50	64.69	49.91	107.50	64.50
2	5->0(26), 6->1(37), 7->2(38), 4->3(28),	129	38	0.370	60.75	83.50	106.25	64.68	47.68	108.29	58.71
3	5->0(26), 4->1(15), 6->2(32), 7->3(30),	103	32	0.340	49.75	67.50	85.25	51.67	34.98	86.17	48.83
4	6->0(37), 7->1(43), 5->2(31), 4->3(28),	139	43	0.331	67.00	91.00	115.00	69.67	45.96	116.32	65.68
5	6->0(37), 5->1(44), 7->2(38), 4->3(28),	147	44	0.327	69.75	95.50	121.25	73.66	48.02	123.27	67.73
6	7->0(35), 5->1(44), 6->2(32), 4->3(28),	139	44	0.326	67.75	91.50	115.25	69.66	45.36	116.16	66.84
7	5->0(26), 4->1(15), 7->2(38), 6->3(24),	103	38	0.325	54.25	70.50	86.75	51.66	33.44	85.48	55.52
8	7->0(35), 6->1(37), 5->2(31), 4->3(28),	131	37	0.313	60.50	84.00	107.50	65.66	41.01	110.30	57.70
9	6->0(37), 4->1(15), 7->2(38), 5->3(21),	111	38	0.301	56.25	74.50	92.75	55.65	33.38	92.38	56.62
10	7->0(35), 4->1(15), 6->2(32), 5->3(21),	103	35	0.300	52.00	69.00	86.00	51.65	30.94	85.75	52.25
11	5->0(26), 7->1(43), 4->2(18), 6->3(24),	111	43	0.295	60.00	77.00	94.00	55.65	32.72	92.01	61.99
12	5->0(26), 6->1(37), 4->2(18), 7->3(30),	111	37	0.292	55.50	74.00	92.50	55.65	32.46	92.50	55.50
13	6->0(37), 4->1(15), 5->2(31), 7->3(30),	113	37	0.283	56.00	75.00	94.00	56.64	32.03	94.25	55.75
14	4->0(41), 7->1(43), 6->2(32), 5->3(21),	137	43	0.277	66.50	90.00	113.50	68.64	37.96	114.54	65.46
15	6->0(37), 7->1(43), 4->2(18), 5->3(21),	119	43	0.271	62.00	81.00	100.00	59.64	32.22	98.83	63.17
16	7->0(35), 4->1(15), 5->2(31), 6->3(24),	105	35	0.268	52.50	70.00	87.50	52.63	28.15	87.50	52.50
17	4->0(41), 6->1(37), 7->2(38), 5->3(21),	137	41	0.260	65.00	89.00	113.00	68.63	35.59	114.89	63.11
18	4->0(41), 5->1(44), 6->2(32), 7->3(30),	147	44	0.256	69.75	95.50	121.25	73.63	37.61	123.27	67.73
19	7->0(35), 6->1(37), 4->2(18), 5->3(21),	111	37	0.253	55.50	74.00	92.50	55.63	28.10	92.50	55.50
20	6->0(37), 5->1(44), 4->2(18), 7->3(30),	129	44	0.250	65.25	86.50	107.75	64.62	32.19	107.38	65.62
21	4->0(41), 7->1(43), 5->2(31), 6->3(24),	139	43	0.245	67.00	91.00	115.00	69.62	34.03	116.32	65.68
22	4->0(41), 6->1(37), 5->2(31), 7->3(30),	139	41	0.243	65.50	90.00	114.50	69.62	33.71	116.68	63.32
23	4->0(41), 5->1(44), 7->2(38), 6->3(24),	147	44	0.241	69.75	95.50	121.25	73.62	35.41	123.27	67.73
24	7->0(35), 5->1(44), 4->2(18), 6->3(24),	121	44	0.234	63.25	82.50	101.75	60.62	28.34	100.47	64.53

and then follows the presentation of the performance results for all algorithms detailed in Section 6.3. The runtime to assign targets to agents, the pathfinding cost, and the success of the run, are all measured for performance. Finally, statistical tests are conducted for the significance of the results.

6.4.1 Experimental Problem Settings

The general setup of experiments such as the map environment, player combinations, the movement direction of players, the cost of moves, the number of test runs for each configuration and the total number of individual tests as well as the computer settings are detailed in Section 2.5.3.

While the pursuing agents employ the STMTA* algorithm during the experiments, the multiple-moving targets use the SF algorithm. Each assignment strategy algorithm is tested under the same configurations in the same environment. All newly proposed approaches, Adaptive Weighted-cost, Joint Weighted-cost and Joint Twin-cost algorithms, in Section 6.3 are evaluated against the baseline Weighted-cost algorithm, which has predefined weighted parameters with ratios of 0.25/0.75, 0.50/0.50 and 0.75/0.25. Altogether, including the *s-mode* and *d-modes*, 21 algorithms will be tested.

For the given environment, the initial scenario for the first tests is to have four pursuing agents and two targets (4 vs 2). To help analyse the algorithms' behaviour better, the second and the third set of tests were conducted with the number of targets increased by one and two, i.e., the pursuers outnumber targets or are equal. Each problem is defined by the starting position of the players. Players have five different sets of starting positions on each map, and they are placed randomly at preselected locations.

6.4.2 Performance Analysis

The performance of the assignment strategy algorithms is evaluated, and the results are presented for the pathfinding cost, the success of reaching targets, the time it takes to assign targets and the minimum cost. The pathfinding cost is measured for the number of time steps taken before capturing all targets for successful runs or until timeout for unsuccessful runs. The runtime is measured

in seconds. Mean is taken for all measurements considering all pursuers and configurations.

Each assignment strategy algorithm is tested for *s-mode* and *d-mode*, and in *d-mode*, the pursuers can assess the position of players and if necessary, then change previously assigned targets with a new configuration after every 10 steps for one set and 20 steps for the other set.

6.4.2.1 Pathfinding Cost

The pathfinding costs for assignment strategy algorithms are measured and displayed in Table 6.3. The table is split into three-player combinations, pursuing agent versus targets (4 vs 2, 4 vs 3 and 4 vs 4), and each player combination contains three map groups. The table shows the mean of three maps for the pathfinding cost and minimum cost [183, 189, 209, 210] within each group. The minimum cost is obtained from the tests run twenty times for each starting state. The overall mean for the pathfinding cost and the minimum cost of all results are shown at the bottom of Table 6.3.

Predefined weight parameters for the Weighted-cost algorithm display similar performance but the dynamic method of identifying weight parameters for the Adaptive Weighted-cost algorithm shows better results. The Joint Twin-cost algorithm performed the worst, but the Joint Weighted-cost algorithm performed better and in some individual cases displayed the best results. It is seen in Table 6.3 that the Adaptive Weighted-cost with the adjusted approach resulted in having the lowest cost for each map group with two exceptions, where Adaptive Weighted-cost with the standard on circle group maps for 4 vs 2 and Weighted-cost (50/50) on corridor group maps for 4 vs 2, but the overall result displays the best performance for Adaptive Weighted-cost with the adjusted computation.

The results in Table 6.3 suggest that dynamically adapting the weight parameter for the assignment strategy algorithm produces better results but to identify its significance the statistical tests are used on the pathfinding costs. The data obtained from the test runs are not normally distributed and because there are many algorithms to compare the Friedman test is used for statistical analysis. First, the ranking is used in the Friedman test where each data set is

Table 6.3: The comparison of assignment strategy algorithms includes the mean for pathfinding cost and minimum cost.

	Weighted-cost									Joint Weighted-cost			Joint Twin-cost			Adaptive Weighted-cost					
	25/75			50/50			75/25			50/50			$m \times n$			Standard			Adjusted		
	Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic	
		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step
Circle																					
Cost mean	80.0	77.4	82.7	79.5	77.6	78.2	78.4	78.8	78.9	81.7	76.5	78.6	95.8	158.2	130.5	76.9	77.3	78.7	75.1	77.2	79.8
min. cost	37.7	35.8	35.6	36.9	35.7	35.1	39.2	35.1	35.2	37.0	34.6	35.0	44.2	47.8	48.6	36.6	35.4	35.7	36.3	36.1	35.7
Corridors																					
Cost mean	76.9	77.0	77.3	77.0	77.7	77.0	75.8	76.7	77.5	77.2	78.4	77.5	80.2	104.0	92.6	75.5	76.7	77.8	75.49	78.0	76.9
min. cost	59.0	59.4	58.1	58.9	60.3	58.8	59.1	58.5	59.0	59.5	59.6	58.7	62.8	66.1	63.4	58.2	58.1	58.8	58.8	59.5	59.8
Large Open Space																					
Cost mean	64.7	66.2	64.6	64.12	66.5	64.9	64.9	65.4	64.9	65.5	66.8	64.9	65.8	79.5	71.4	65.0	66.2	64.7	64.16	65.2	64.5
min. cost	46.8	46.7	46.9	47.1	46.6	46.3	47.1	46.4	46.1	47.1	47.0	46.0	47.3	51.8	48.8	46.0	46.8	45.2	46.0	46.1	44.8
Average for all maps																					
Cost	73.9	73.6	74.9	73.6	73.9	73.3	73.0	73.7	73.8	74.8	73.9	73.7	80.6	113.9	98.2	72.5	73.4	73.7	71.6	73.5	73.8
min. costs	47.8	47.3	46.9	47.7	47.5	46.7	48.5	46.7	46.8	47.9	47.1	46.54	51.4	55.2	53.6	47.0	46.8	46.57	47.0	47.3	46.8

ranked separately for each algorithm and the algorithm with the best performance is ranked 1, as shown in Table 6.4. The ranking is obtained for all algorithms per map and added up according to their group category on each player combination. To obtain the overall ranking results, all ranking values are added to get the total sum and the final ranking as displayed at the bottom of Table 6.4. Second, the Friedman test uses these ranking results to obtain p-values. The 0.05 is used for the level of significance and the results are displayed in Table 6.5. The evidence suggests that 4 vs 3 and 4 vs 4 player configuration results display statistically significant differences and the results for the 4 vs 2 player combination show significance only in most cases.

6.4.2.2 Minimum Cost

As for the minimum cost for the test runs on each state, both Weighted-cost and Adaptive Weighted-cost algorithms with their variations displayed the best results in some individual cases throughout the experiments. Table 6.3 shows that every player combination has a different algorithm that performs the best. The results highlight that the Joint Weighted-cost algorithm produces the best outcome on 4 vs 2, the Adaptive Weighted-cost (standard) algorithm outperforms on 4 vs 3, the Weighted-cost (50/50) algorithm on 4 vs 4 and the overall best performance is displayed by the Joint Weighted-cost which outperforms Adaptive Weighted-cost (standard) by a very small margin. The table suggests that joining the distance and the covered area may find the lowest costs. Also, the evidence indicates that the algorithms in the *d-mode* with re-assessment in every 20 steps display the minimum cost in all player combinations and for the best-performing algorithm. Comparing the results for different player combinations shows, as is expected, that the pathfinding cost and the minimum cost increase with the number of targets.

6.4.2.3 Success Rate

Success is measured when the targets' positions are occupied by at least one pursuing agent. Although in these experiments, in two out of three pursuer-to-target combinations, the pursuing agents outnumber the targets, it is possible

Table 6.4: Ranking of 21 algorithms based on pathfinding costs with three player combinations, three map groups and 300 problems each. The bottom of the table displays the overall ranking. The best performance is ranked no. 1.

	Weighted-cost									Joint Weighted-cost			Joint Twin-cost			Adaptive Weighted-cost					
	25/75			50/50			75/25			50/50			m * n			Standard			Adjusted		
	Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic		Static	Dynamic	
		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step		10step	20step
4 vs 2																					
Circle	3551	3012	2961	3305	3005	3223	3235	3098	3141	3408	3027	3116	3772	4185	4182	3090	2969	3335	3338	2985	3366
Corridors	3070	3436	3260	3114	3392	3285	2899	3142	3141	3436	3242	3237	3493	4095	3876	2905	3457	3331	2859	3325	3311
LargeOpenSpace	3306	3247	3385	3189	3303	3277	3298	3095	3064	3442	3370	3150	3144	3920	3545	3282	3369	3194	3304	3089	3334
Total Rank Values	9927	9694	9605	9607	9699	9784	9431	9334	9345	10285	9639	9502	10409	12199	11603	9277	9795	9859	9500	9398	10011
Rank	16	11	8	9	12	13	5	2	3	18	10	7	19	21	20	1	14	15	6	4	17
4 vs 3																					
Circle	3249	3199	3366	3144	3075	3116	3079	2930	2999	3358	3202	3144	3917	4264	4261	3295	3309	3154	3042	3126	3076
Corridors	3089	3122	3269	2870	3159	3062	2932	3019	2902	3164	3645	3598	3974	4694	4480	2848	3131	3073	3043	3291	2940
LargeOpenSpace	3064	3321	3219	3058	3355	3069	3002	3305	3232	3138	3568	3125	3391	4633	3741	3031	3351	3329	3090	3281	3001
Total Rank Values	9401	9642	9853	9071	9589	9246	9013	9253	9132	9660	10414	9867	11282	13591	12482	9173	9790	9556	9175	9697	9017
Rank	9	12	16	3	11	7	1	8	4	13	18	17	19	21	20	5	15	10	6	14	2
4 vs 4																					
Circle	3465	3008	2968	3163	3152	2986	3383	3073	2946	3146	2904	3071	4175	4831	4723	3297	3081	2850	3066	2907	3110
Corridors	3234	3280	3167	3164	3431	3214	3036	3221	3322	3015	3441	3158	3536	4239	3996	2889	3242	3188	2930	3333	3268
LargeOpenSpace	2955	3428	3321	2749	3497	3130	3016	3156	3148	3009	3558	3329	3222	4864	4148	3042	3523	2956	2788	3338	3130
Total Rank Values	9653	9716	9455	9075	10080	9329	9435	9450	9416	9169	9902	9557	10932	13934	12867	9227	9845	8994	8784	9577	9508
Rank	14	15	10	3	18	6	8	9	7	4	17	12	19	21	20	5	16	2	1	13	11
The Sum of All Combinations & Ranking																					
Total Sum	28981	29052	28913	27753	29367	28359	27879	28037	27893	29114	29954	28926	32622	39724	36951	27677	29429	28408	27459	28672	28535
Rank	13	14	11	3	16	7	4	6	5	15	18	12	19	21	20	2	17	8	1	10	9

for pursuers to fail and if at least one target manages to escape the capture before the timeout this is recorded as an absence of success for all pursuers. Figure 6.1 displays the success rate for each player combination while the assignment strategy is in *s-mode* or in *d-mode*. It is evident from the graph that the pursuers who get assigned a target in the *s-mode*, do not change their target until it is captured, and are 100% successful in all maps and all combinations. Large open space maps are easier to navigate and have more options to approach the targets, therefore, the same 100% success rate can be seen in these maps where the pursuing agents catch the targets successfully in all tested situations. Similar results are displayed for corridor type of maps that all algorithms achieve their intended results by capturing all targets, apart from the AR0016SR map for the Joint Twin-cost algorithm in the *d-mode* with re-assessment every 10 steps. This only happened on the map with two different starting states for 4 vs 3 and 4 vs 4 player combinations, this slightly affected the performance. The circle-shaped maps are the most difficult ones to navigate and if the speed of targets and pursuers are the same, which is the case, then it makes it more challenging. The

Table 6.5: p-values display the significance of pathfinding costs on each map per player combination.

Maps	Player combinations		
	4 vs 2	4 vs 3	4 vs 4
Circle			
AR0401SR	1.2E-09	1.6E-04	8.2E-28
AR0402SR	2.4E-23	1.0E-28	3.8E-37
AR0526SR	1.8E-05	4.5E-08	1.7E-29
Corridors			
AR0016SR	3.9E-22	5.5E-24	9.4E-19
AR0413SR	2.6E-05	2.0E-42	4.1E-03
AR0712SR	0.0868	5.8E-18	2.8E-34
Large Open Space			
AR0332SR	0.0778	2.1E-21	7.0E-18
AR0509SR	0.0856	5.4E-05	2.9E-12
AR0607SR	4.1E-14	8.1E-14	1.2E-45

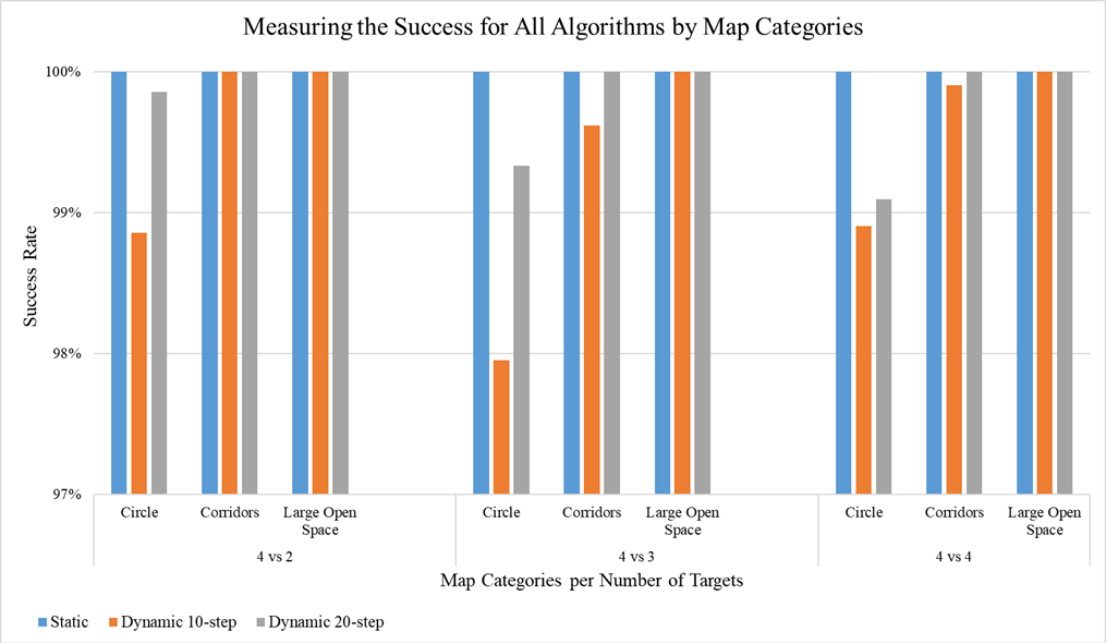


Figure 6.1: The success rate for pursuing agents using the *s-mode*, and *d-mode* with change in 10 and 20 steps.

pursuing agents can only succeed with these types of maps only if they split and surround the target otherwise the tests end with a continuous run until timeout. The AR0402SR map in this group has been the most difficult to navigate and succeed, and the lowest performance is displayed by the Joint Twin-cost algorithm with 78% on 4 vs 3 and 4 vs 4 player combinations. All algorithms in the *d-mode* with re-assessment of 10 steps or 20 steps with various combinations and settings manage to achieve an overall result of over 99.5% and 99.8%, respectively, in the circle-shaped maps. All algorithms achieve 100% of success when their navigation is in *s-mode*, but when the *s-mode* and *d-mode* are averaged, the most successful algorithm in capturing the targets is Adaptive Weighted-cost with the adjusted method at 99.9%.

6.4.2.4 Assignment Runtime

The time spent for assignment strategy algorithms to identify and assign targets to the pursuing agents is measured in seconds. Each algorithm recorded the time from the start of the test run until all agents were provided with the most

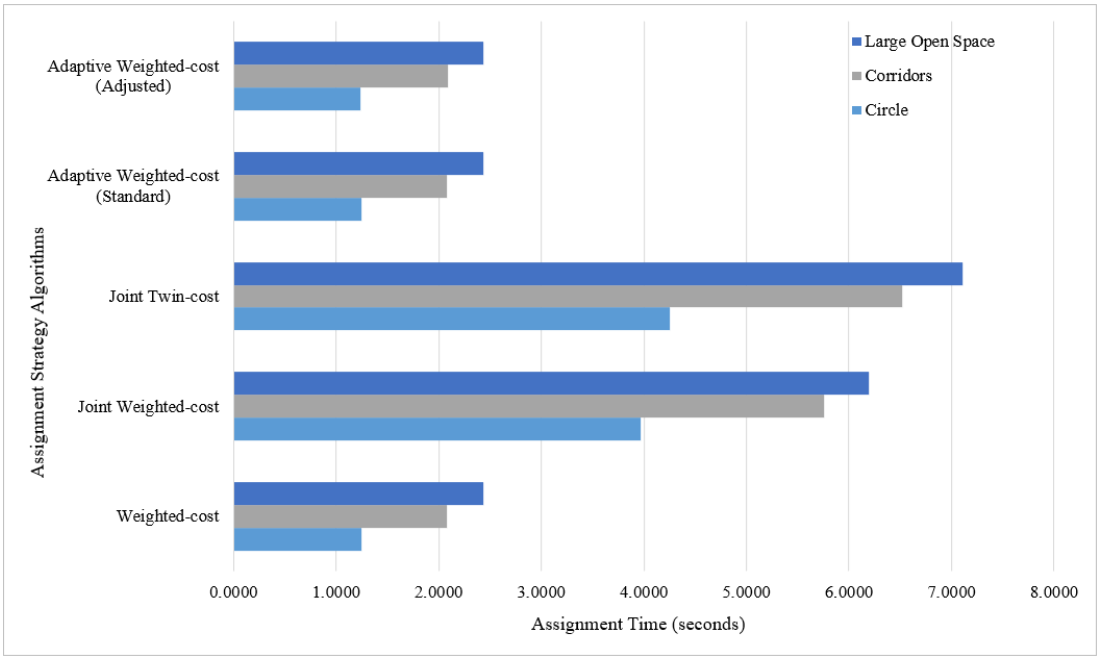


Figure 6.2: Comparing the assignment runtime in seconds for assignment strategy algorithms on three different map groups.

optimal combination set and assigned all available targets on the map. Figure 6.2 illustrates the results for all algorithms and their behaviour on map groups. Because all five starting positions are fixed on every test run and the time to assign targets is measured before the navigation starts, each algorithm's *s-mode* and *d-mode* with 10 steps and 20 steps of the assignment are averaged. The graph depicted in Figure 6.2 presents the circle-shaped maps that are quicker to assign targets with slightly over a second in all algorithms. Although corridor-shaped maps are larger and have more empty cells to occupy in comparison with large open-spaced maps, the algorithms are quicker to assign targets. The Joint Weighted-cost and Joint Twin-cost algorithms use cells to label covered for each pursuer and as expected, this reflects on the higher computation cost for each assignment runtime. The number of pursuers is not altered in player combinations, only the number of targets increases. Therefore, the number of possible combinations remains the same for the pursuers and the increase in the number of targets does not escalate the computation time in assigning targets. Dynamically adapting the weight parameter on each test

outperforms the predefined weight parameters and the Adaptive Weighted-cost algorithm with the adjusted approach displays the quickest assignment time.

6.5 Conclusion

In this study, novel approaches have been proposed for coupled planning in developing assignment strategy algorithms for multiple pursuing agents in an environment with multiple moving targets and static obstacles. The proposed methods including Adaptive Weighted-cost, Joint Weighted-cost and Joint Twin-cost algorithms provide combinations that help to increase the performance by reaching the targets most cost-effectively. To find the optimal combination, the Adaptive Weighted-cost algorithm is based on the sum of costs, *DistancesSum*, and makespan, *MaxDistance*, whereas the Joint Weighted-cost and Joint Twin-cost algorithms are based on the *DistancesSum* and covered area, *CombinationCoverage*, of the pursuers. These newly introduced algorithms were analysed and compared against the existing overall best approach, the Weighted-cost algorithm.

The experiments were conducted on the benchmarked Baldur's Gate gaming maps and performance was evaluated for the pathfinding cost, that is the number of steps travelled before reaching the targets, the minimum cost per test set, and the success of the completed test runs and the computation time for assigning targets. The findings from the empirical evaluations suggest that the runtime was high for Joint Weighted-cost and Joint Twin-cost, this is due to computing the covered area, *CoveredArea*, for each pursuer, and this is reflected in the pathfinding costs, although Joint Weighted-cost occasionally displayed the minimum cost per test run. The results highlight that using dynamic weight parameters for the Adaptive Weighted-cost algorithm with the adjusted method displayed the best overall performance. Furthermore, it succeeded with a rate of 99.9% and was proven to be statistically significant. To conclude, the new proposed approach is more efficient and gives better performance over other assignment strategy algorithms.

Chapter 7

Increasing Covered Area to Capture Moving Targets in a Dynamic Environment

7.1 Introduction

A heuristic function to compute the distance between an agent and a target is used as the Manhattan distance (four-connected grids) or Euclidean distance (eight-connected grids). The heuristics do not overestimate the solution cost; however, the presence of obstacles may result in poor solution quality. A solution to the problem aims to minimise the distance between the agent and the target and eventually catch the target. This is the approach where the problem may not be minimised if the target is moving and all players have the same speed, then most likely, the chase can continue without capturing the target. Multiple agents adopting the same strategy will follow the target as if they are grouped into one instead of benefiting from surrounding and trapping the target. The objective of the introduced solution is to outmanoeuvre the target [60]. First, the covered area is computed for each player then the action of each agent is defined either to increase the covered area or to attack the targets to reduce the distance. It is also possible that the action that increases the covered area can reduce the distance. This problem characterises the real-time algorithms that interleave planning and

execution and it is prevalent in computer video game environments [34, 106].

In this chapter, the problem scenario is extended from single to multiple targets and the original CRA algorithm [60] is enhanced to adapt to multiple targets. The idea of the covered area approach, to trap and outmanoeuvre the target, is used in the proposed new algorithm, Cover Dynamic Moving Target A* (CDMTA*). The method of the new algorithm is developed such that each pursuing agent is assigned a target using the assignment strategy algorithm in the coupled stage, and it is enhanced to identify the attacking action based on the distance to the target in the decoupled stage. Another improvement includes position consideration of other agents and clearing any actions that lead to an occupied state of other agents.

The layout of this chapter is organised as follows: the cover computation and existing algorithm with its new enhanced approach is given in Section 7.2. Experimental results and analysis are presented along with discussions described in Section 7.3. Finally, relevant conclusions are drawn in Section 7.4.

7.2 Methods

This section first presents an existing method that allows multiple agents to coordinate and surround a single target using a covered area approach, which is the opposite of the "rush-in" method. Then, the section follows with a new approach that improves and enhances the existing approach by presenting multiple moving targets and having assignment strategies for the pursuers.

7.2.1 Existing Approches

In general, heuristics is the estimated distance between two points. The computed distance gives an indication of which action to take in order to move straight to the destinations. In an environment with multiple agents, the strategy can change and instead of directly approaching the target, it is possible for agents to disperse and subsequently trap the target. Thus, this method, the *cover heuristic*, is discussed in the first part of this section and followed by an algorithm that implements it.

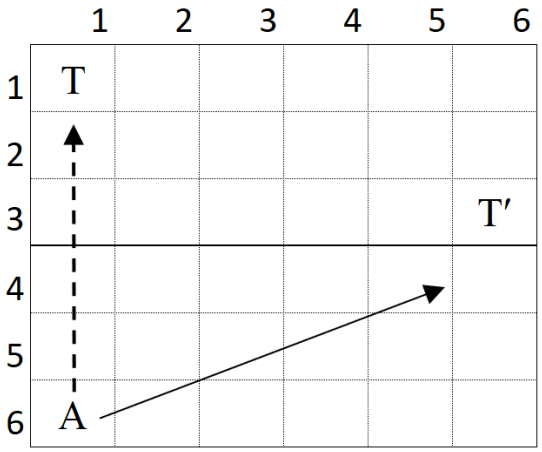


Figure 7.1: Agent A is positioned at the left bottom and moving target T is positioned on the left top. T' is the goal position for T. The top three rows are the cover set for T and the bottom three rows are the cover set for A. Assuming no obstacles, A move to T is the *distance heuristic* (dashed arrow) and A move to T' is the *cover heuristic* (straight arrow).

7.2.1.1 The Cover Heuristic

Imagine two players on the two-dimensional map with an agent A positioned on the left bottom and a target T positioned on the left top as illustrated in Figure 7.1. T has the moving ability and can run away from the approaching agent and the suggested target's goal is T'. Manhattan distance is used and A at the current node expands its nodes to the neighbouring nodes in a similar way to the breadth-first search. By taking turns, T expands its nodes from its current position. Each expanded node is the set of a *cover set* for A, and nodes expand until they reach the target node expansion, that is the *cover set* for T [60]. The expansion stops when both sets meet and there is no node to expand further. The position of the agent and the target defines the size of the *cover set*. For example, the *cover set* for the agent is larger if the target is cornered. The increase of the agents and their different positions is the joint effort of all *cover sets*.

In Figure 7.1, both players have the same number of nodes expanded in their *cover sets*. In a situation with no obstacles and the same speed for players, the *cover set* expansion is the top three rows for the target and the bottom three rows are for the agent. If the agent uses the *distance heuristic*, then the optimal

move is to follow the dashed arrow. The *cover set* can suggest a different route (straight arrow) and select an action that always gives a larger *cover set*. The notion of a *cover set* can be referred to as a *cover*. The computation of the *cover* is costly, grows exponentially with the number of the pursuers and has linear time complexity. Besides that, the *cover* can benefit from two features that can be effective. First, it lessens the mobility of the target by minimising the states that the target can travel to. Second, it increases the *cover set* for pursuers by spreading out and coordinating multiple pursuers, which leads to increasing *cover* and trapping the target.

7.2.1.2 Cover with Risk and Abstraction and Multi-Target

Heuristic search algorithms use *distance heuristics* to estimate the distance between an agent and the target. The goal of the search is to identify the shortest path between the states of the agent and the destination. The Cover with Risk and Abstraction (CRA) algorithm presented in this section contradicts the shortest distance capture [60]. Instead of heading directly towards the targets, the CRA algorithm surrounds and demobilises the target before the capture. It is a decentralised approach, and CRA receives three parameters (*agent*, *risk*, *abstraction*) for each pursuer and returns an action that maximises the cover value. The *risk* parameter $\rho \in [0, 1]$ identifies a choice between states that increase the cover ($risk = 0$) or select an action that moves straight towards the target to reduce the distance ($risk = 1$). The *abstraction* takes a predefined level where the PRA* algorithm [34] is used to run the heuristic search. An agent initially assesses all possible actions at the neighbouring states on each move, and it then sequentially computes its cover for each possible successor state. Then filters through and takes the action that results in the state with the highest cover value. Next, compares the predefined *risk* parameter and selects a move that results in either cover action or heuristic distance action, which is the PRA* algorithm that uses abstraction. Notice that the original CRA algorithm works only with one target, therefore, it has been enhanced to perform with multiple targets.

Algorithm 10 outlines the pseudo-code of the CRA algorithm for the pursuing

Algorithm 10 Cover with Risk and Abstraction and Multi-Target.

```
1: function CRA(a, ρ, ℓ)
2:     initialise state for an agent a                          ▷ default 0 for risk ρ, level ℓ
3:     initialise vectors allActions, maxActions, clearActions
4:     append available actions for a onto allActions
5:     for each allActions n do
6:         v ← getCoverageValue(n);
7:         maxValue ← v                                         ▷ the highest cover value
8:         append max. or equal value actions onto maxActions
9:     end for
10:    for each maxActions m do
11:        boolean include = true;
12:        for each agents p do
13:            if (wait action for p and p′ at the same position) ∨ (p's destination is the position of p′) then
14:                include = true;
15:                append m onto clearActions
16:            end if
17:        end for
18:    end for
19:    while target not caught do
20:        get the closest target t of all targets
21:    end while
22:    for each clearActions c do
23:        compute distance da from the action c to the t
24:        maxAction ← da;                                      ▷ da is the minimum distance
25:    end for
26:    hAction ← runHeuristicSearch;                            ▷ A* from a to t
27:    cover ← getCoverageValue(hAction);
28:    Δc = (maxValue − cover)/(maxValue + 1)
29:    if Δc > ρ then
30:        return maxAction;
31:    else
32:        return hAction;
33:    end if
34: end function
35: ---------------------------------------------------------------
36: function GETCOVERAGEVALUE(coverValue)
37:     calculateCover(state);
38:     return 2 × (coveredCells / (coveredCells + uncoveredCells)) - 1;
39: end function
```

agent. Depending on the position of the target and the cover values, the CRA algorithm aims to identify the action that maximises its cover which helps in catching the target. Therefore, after initialising states for the agent in line 2, three vectors are created in line 3. All available neighbouring actions are inserted into the *allActions* vector in line 4 and with the aid of the eliminating procedures, these vectors are functioning to lessen the set of succeeding states of optimal cover in lines between 5 and 25. The *allActions* vector gets all actions including the wait action, then only maximum value or equal to maximum value actions are left in the *maxActions* vector. The final vector *clearActions* identifies the action with the maximum cover value, the *maxAction*. Since the original algorithm is built to

work only with one target, and rather than providing one target's state, the lines from 19 to 21 compute the distances to each available, uncaught target and pick the one with the closest distance value. Line 26 gets a result for *hValue* using heuristic distance from the agent's position to the target. It is possible to use any heuristic search algorithm, in this case, PRA* is used. The move, which has an optimal heuristic action, computes a cover value in line 27. The cover value computed in lines 6 and 27 is obtained through the function *getCoverageValue()* where it returns the normalised value between 0 and 1. The *delta-cost* equation is shown in line 28, and the outcome indicates which action should be taken. A *risk* zero means that always cover action is taken that maximises the covered area and the *risk* equal to one mean PRA* move that involves taking a risk, moving forward, and attacking the target. The PRA* algorithm can be replaced by any heuristic search algorithm; however, it needs to use abstractions otherwise the computation is high. The equation in line 29 compares *delta-cost* to the predefined *risk* parameter and the outcome returns the action.

Since the CRA algorithm is a decoupled approach, it runs independently on each pursuer separately. The time complexity is linear in regard to the cover heuristics. First, line 2 is an operation with *one* time that is needed for an initialisation statement, next is the initialisation statements for three vectors which require *three* times in line 3. All available actions of the pursuer to append onto a vector generate operations up to *five* times. Then the iterative statements follow. Line 5 requires k times. Line 6 calls a function that is included in lines 36-39. This function expands the states to compute the coverage in line 37 with linear complexity requiring n times and line 38 needs *one* time. Consequently, line 6 generates up to kn times. Line 7 is an assignment statement that needs *one* time but line 8 has a complexity that is less than *five* times. Meanwhile, the statements in lines 10 to 18 have nested loops. Line 10 requires k operations for each *maxActions* and line 12 needs kn operations for the *agents*. Line 11 needs k time as it is an assignment statement inside the outer loop, however, lines 13, 14 and 15 are the statements in the inner loop including the condition requiring kn operations. The *maxActions* and *agents* generate $k \times n$ operations which is a need of up to kn time. This follows with a while loop in line 19 that needs k operations. Line 20 requires kn operations for targets. The total number of

operations for lines 19-21 is kn. Next is the final loop between lines 22-25. Line 22 needs k operations for *clearActions* and line 23 generates kn operations to compute the distance whereas line 24 is an assignment statement that requires k operations. Therefore, this loop requires kn time. The heuristic search needs k times in line 26. Line 27 calls this function a second time and requires up to kn times. Line 28 needs *one* time operation. Lastly, lines 29 to 33 are the conditional statements that require *one* time. Alongside the enhancement in finding the nearest target in the target assignment, it does not increase the total time in solving the problem. In addition, PRA* has $\mathcal{O}(n \log n)$ complexity [173]. Hence, the worst-case complexity of the CRA algorithm is $\mathcal{O}(n \log n + kn)$.

7.2.2 Proposed Approach

This subsection introduces a novel algorithm called Cover Dynamic Moving Target A* (CDMTA*), which involves multiple pursuing agents and multiple targets and finds the path using the *cover heuristic*, a surrounding strategy that outmanoeuvres the targets before being caught. The CDMTA* algorithm is the enhancement of the CRA algorithm, which is a multi-agent algorithm, that only engages one moving target in its actions. CRA has been modified to incorporate multiple moving targets in this section as a new approach. Since this is the instance of the PAMT problem, a strategy to assign targets is important. Therefore, another new approach, the assignment strategy algorithm is applied at the initial state and the number of pursuing agents p are assigned targets t if $p \geqslant t$. Assignment strategy is a surjective function. The algorithm helps agents to decide which targets to follow, one target assigned per pursuer and significantly reduces the runtime as demonstrated during the experiments in Section 2.5.1. It can be applied during the test runs where agents dynamically assess the new positions of all targets, and if necessary, targets are re-assigned. This method of navigation switches from static to dynamic mode and requires the use of a predefined parameter, *steps*. During the test runs, this dynamic mode halts and evaluates the position of all pursuers and targets. The use of the assignment strategy is the coupled approach where all agents are counted as one entity. After assigning the targets, each agent computes its path, a

decoupled approach, and takes actions toward the target by assessing the *risk* parameter that is mentioned in the previous section.

For search algorithms, chasing a single moving target is challenging, and it becomes more challenging when multiple targets exist. Knowing which target to pursue in order to quickly and efficiently catch all targets is another difficult task. The methods taken to address such problems are detailed in the pseudo-code of the proposed CDMTA* algorithm at Algorithm 11. Only the changes are highlighted as this is the enhancement to the CRA algorithm.

Although the algorithm incorporates multiple targets, the assignment strategy algorithm (coupled approach) assigns a target, *assignedTarget*, to increase its performance, and it is a required component that is specified after line 1. The *clearActions* vector uses *assignedTarget* to compute the distance in line 22 and similarly, it is used in lines 25 and 28 to get a heuristic distance *hAction* and *d*.

The next important improvement is the conditional statement in lines 13-15. The statement compares this pursuing agent's action to all pursuers, and it disregards if it is itself. According to the original CRA algorithm, the condition is met when both this pursuer and the other pursuer have a *wait* action at the same position OR this pursuer's destination is the other pursuer's position. The purpose of the cover is to spread out and surround the target, therefore, the statement should be opposite, and the conditional statement is corrected. Thus, only the actions that move in different directions are now inserted into the *clearActions* vector while looping through actions between lines 10 and 20.

The other enhancement is in line 27 where the *delta-cost* is normalised by utilising both values. The variables *cover* and *maxValue* have corresponding values of x and y. In the original formula of CRA, the difference $(x - y)$ is normalised using the $(x-y)/(x+1)$, however, the denominator only includes the contribution of x without y. In the revised formula, the contribution of both x and y are considered, resulting in $(x - y)/(x + y + 1)$. For instance, if $x = 0$, the previous formula's *delta-cost* would be equal to y, whereas the new formula's *delta-cost* would be equal to 1.

The final enhancement is the *risk* parameter that identifies the choice of movement. The authors of the CRA algorithm indicate the drawback of holding back and remaining still and possibly waiting for other pursuers to come where

Algorithm 11 Cover Dynamic Multiple Target A*.

```
1: function CoverDynamic(a, ρ, ℓ, s)
Input: assignedTarget ← assignTarget(target);
2:     initialise state for agent a                          ▷ default 0 for risk ρ, level ℓ, steps s
3:     initialise vectors allActions, maxActions, clearActions
4:     append available actions for a onto allActions
5:     for each allActions n do
6:         v ← getCoverageValue(n);
7:         maxValue ← v                                      ▷ the highest cover value
8:         append max. or equal value actions onto maxActions
9:     end for
10:    for each maxActions m do
11:        boolean include = true;
12:        for each agents p do
13:            a = wait action for p and p′ at the same position
14:            b = p's destination is the position of p′
15:            if ¬(a ∨ b) then
16:                include = true;
17:                append m onto clearActions
18:            end if
19:        end for
20:    end for
21:    for each clearActions c do
22:        compute distance da from the action c to the assignedTarget
23:        maxAction ← da;                                   ▷ da is the minimum distance
24:    end for
25:    hAction ← runHeuristicSearch;                         ▷ A* from a to assignedTarget
26:    cover ← getCoverageValue(hAction);
27:    Δc = (maxValue − cover)/(maxValue + cover + 1)
28:    compute distance d from a to assignedTarget
29:    if d ≤ s then
30:        ρ = 1;
31:    end if
32:    if Δc > ρ then
33:        return maxAction;
34:    else
35:        return hAction;
36:    end if
37: end function
38: ---------------------------------------------------------
39: function getCoverageValue(coverValue)
40:    calculateCover(state);
41:    return 2 × (coveredCells / (coveredCells + uncoveredCells)) - 1;
42: end function
```

the pursuer can quickly reach the target [60]. A new additional *steps* parameter is included to prevent local minima. Based on this parameter, the pursuer determines how close it is to the target and, if it is close, it can avoid assessing the *risk* parameter and move directly to capture the target. The algorithm takes *steps* parameter in line 1 and the initial value is set to zero unless defined otherwise in line 2. The conditional statement in lines 29 to 31 checks whether the distance d is less than or equal to the *steps* parameter and if it is, then the *risk* parameter is set to one, and the pursuer rushes to capture.

Similar to Algorithm 10, CDMTA* is a decoupled approach. It runs independently on each pursuer separately and the time complexity is linear in regard to the cover heuristics. First, the algorithm commences by assigning a target that has been provided by Algorithm 1 which has the time complexity of $\mathcal{O}(kln)$. Next, line 2 is an operation with *one* time that is needed for an initialisation statement, next is the initialisation statements for three vectors which require *three* times in line 3. All available actions of the pursuer to append onto a vector generate operations up to *five* times. Then the iterative statements follow. Line 5 requires k times. Line 6 calls a function that is included in lines 36-39. This function expands the states to compute the coverage in line 37 with linear complexity requiring n times and line 38 needs *one* time. Consequently, line 6 generates up to kn times. Line 7 is an assignment statement that needs *one* time but line 8 has a complexity that is less than *five* times. Meanwhile, the statements in lines 10 to 20 have nested loops. Line 10 requires k operations for each *maxActions* and line 12 needs kn operations for the *agents*. Line 11 needs k time as it is an assignment statement inside the outer loop, however, lines 13-17 are the statements in the inner loop including the condition requiring kn operations. The *maxActions* and *agents* generate $k \times n$ operations which is a need of up to kn time. This follows with the final loop between lines 21-24. Line 21 needs k operations for *clearActions* and line 22 generates kn operations to compute the distance whereas line 23 is an assignment statement that requires k operations. Therefore, this loop requires kn time. Next is the heuristic search that needs k times in line 25. Line 26 calls this function a second time and requires up to kn times. Line 27 needs *one* time operation. Line 28 requires k time operation for computing the distance. After that, the conditional statements in lines 29-31. The if statement needs *one* time in line 29 and so does the assignment statement in line 30. Finally, lines 32 to 37 are the conditional statements, too, that require *one* time. Furthermore, the algorithm does not need to loop through the targets as each target is assigned to the pursuer which reduces the number of time steps. In this regard, the algorithm conserves the complexity of Algorithm 1 and Algorithm 10 which is in overall the worst-case is $\mathcal{O}(kln)$.

7.3 Experimentation and Discussion

This section presents the experimental evaluation of the existing and proposed approaches in the PAMT environments. Performance is evaluated against an algorithm that uses the assignment strategy algorithm, STMTA*, and an algorithm that uses *cover heuristics*, CRA, and the different variations of the new proposed CDMTA*. First, the experimental problem setting is described and followed by a performance analysis of the pathfinding costs and conducts statistical tests. Additionally, the minimum and the maximum cost, the success of the results and the runtime measurements are also reported.

7.3.1 Experimental Problem Settings

Although there are several ways to solve the PAMT problems, this study evaluates an approach that uses a *cover heuristic*, covered area, method that maximises the area for pursuers to trap and outmanoeuvre the targets. Since there are no previous attempts to solve the multi-agent and moving multi-target problem using the covered area, the proposed method is compared to the CRA algorithm, which is enhanced to incorporate multiple targets. Additionally, the performance is also compared to the STMTA* algorithm that employs the assignment strategy algorithm. The assignment strategy can be optimised using a variety of methods and the Adaptive Weighted-cost criterion is used as overall it has been optimal as described in Chapter 6.

Two types of targets were tested against the pursuing agents. The first one is the SF algorithm [60] which is based on the A* algorithm and attempts to escape from the approaching pursuers by moving to one of several pre-computed randomly generated locations on the map. The algorithm moves away from the pursuers to one of these locations and evaluates its move using the parameters provided, if the selected location is safe to proceed then it continues to that location otherwise evaluates the situation and moves to another furthest away location. The second target algorithm is the MPTM algorithm (Chapter 5) which is built on a state-of-the-art TrailMax algorithm [78]. With the location of the pursuers available, MPTM can assess each pursuer's location and make smart moves to avoid them all and keep running away until it gets trapped and captured.

The experimental results have shown that MPTM is difficult to capture since it continuously evades pursuers, in comparison to SF with random moves, which most likely gets caught relatively quickly.

The CDMTA* algorithm is tested with three different modifications. The *delta-cost* equation is tested first with the previous formula and the new formula, line 27 in Algorithm 11. The second is the different *steps* parameter for dynamic risk analysis, lines between 28 and 31. The CRA algorithm performs better when the *risk* parameter is set to 0.1 and has the highest success rate. Therefore, the final variation of CDMTA* is tested with the same *risk* setting. The performance of CDMTA* and its variations are compared against enhanced CRA. Since the assignment strategy is used, it is also compared against the STMTA* algorithm. Altogether, including the variations, 14 algorithms are to be tested.

The general setup of experiments such as the map environment, player combinations, the movement direction of players, the cost of moves, the number of test runs for each configuration and the total number of individual tests as well as the computer settings are detailed in Section 2.5.3.

Numerous testbeds and different sets of scenarios are implemented to better understand and analyse the behaviour of the algorithms. There are five different pursuer-to-target combinations for the given environment, pursuers p and targets t. In the first set, there are three pursuers versus two targets (3 vs 2), then the pursuers increased to four (4 vs 2), and then the number of targets increased until $p = t$, (3 vs 3, 4 vs 3 and 4 vs 4). The pursuers and targets are positioned on preselected random locations on each map. Four alternative sets of starting states are provided. There are three starting states that position pursuers first in the same location on the corner of a map, second in the centre of a map closeby to each other and third to spread out by the walls of a map. The targets are always positioned as far away on the opposite side of the map.

7.3.2 Experimental Results and Performance Analysis

The evaluation of presented algorithms and their performance analysis is conducted with respect to the following key indicators: (i) the pathfinding cost, (ii) minimum and maximum costs, (iii) success rate and (iv) runtime. The

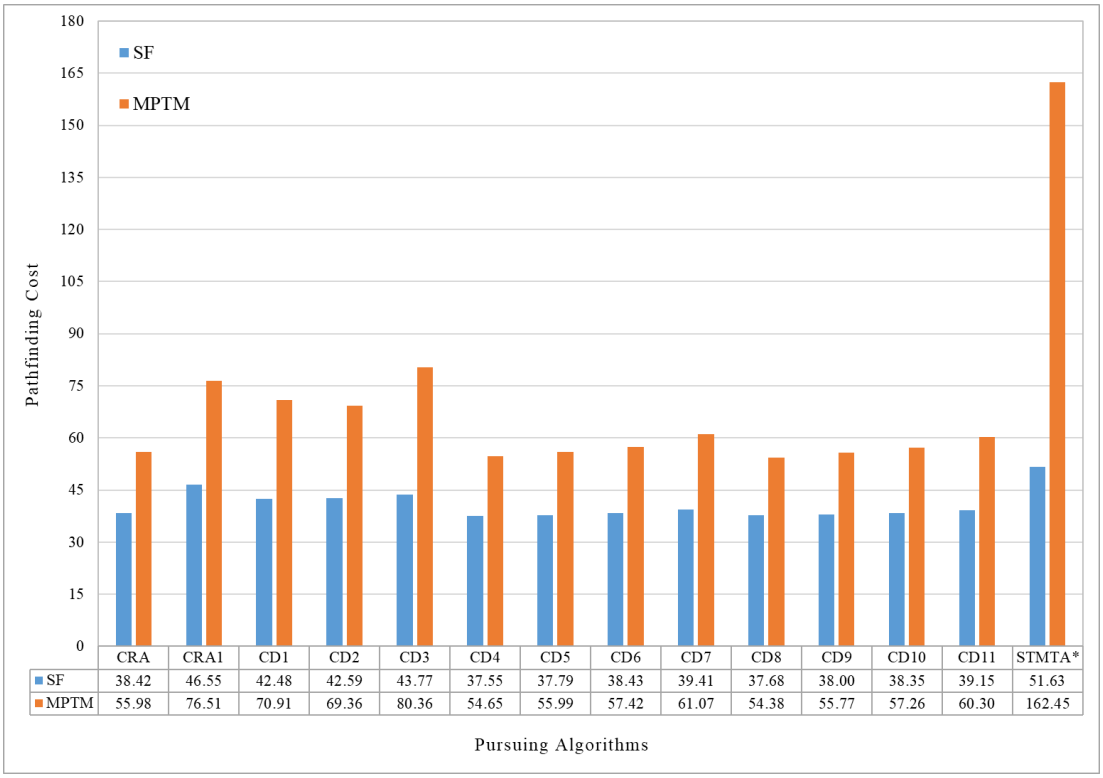
pathfinding cost is the total distance travelled by all pursuers from their starting location and capturing all targets for successful runs, or if unsuccessful in capturing then it is the maximum number of permitted steps (timeout). The results are statistically tested for significance. Similarly, the minimum cost and maximum costs are the number of steps within one set, corresponding to the lower and upper bounds. The timeout is excluded from the maximum cost and is substituted with the second maximum cost. The success rate is measured when every target is captured and the pursuer is placed in the same state, and one pursuer is sufficient to claim the occupancy. The runtime is measured in seconds.

7.3.2.1 Pathfinding Cost

To evaluate previously discussed algorithms, the pathfinding cost is measured in terms of the number of steps travelled from the initial position to the target capture. The CDMTA* algorithm takes different parameters for example *risk* or *steps*. Additionally, CDMTA* is tested with different formulas to normalise the *delta-cost* and with improved action clearance. Table 7.1 lists all of these

Table 7.1: Code names for the pursuing algorithms.

		Code-names	*risk* parameter	*delta-cost* formula	action clear	*steps* parameter
Pursuing Algorithms	Enhanced CRA	CRA	0	-	-	-
		CRA1	0.1	-	-	-
	CDMTA*	CD1	0.1	old	no	0-step
		CD2	0.1	old	yes	0-step
		CD3	0.1	old	yes	2-step
		CD4	0	old	yes	0-step
		CD5	0	old	yes	2-step
		CD6	0	old	yes	3-step
		CD7	0	old	yes	5-step
		CD8	0	new	yes	0-step
		CD9	0	new	yes	2-step
		CD10	0	new	yes	3-step
		CD11	0	new	yes	5-step
	STMTA*	STMTA*	-	-	-	-



	CRA	CRA1	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	STMTA*
SF	38.42	46.55	42.48	42.59	43.77	37.55	37.79	38.43	39.41	37.68	38.00	38.35	39.15	51.63
MPTM	55.98	76.51	70.91	69.36	80.36	54.65	55.99	57.42	61.07	54.38	55.77	57.26	60.30	162.45

Figure 7.2: Comparing the pathfinding costs for pursuing algorithms per target algorithm. The data table at the bottom is the mean for all configurations and settings.

algorithm modifications, and each CDMTA* algorithm is given a code name for simpler representation in the tables and graphs that follow. *Risk* has values between 0 and 1, where 0 is conservative in its forward attacking moves and chooses to take the surrounding approach, while 1 is the opposite and attempts to reach the goal as rapidly as possible without using any strategy. According to earlier research, the CRA algorithm performs better than others when the *risk* parameter is set to 0.1 [60]. Therefore, in this study, the enhanced CRA is tested with 0 and 0.1 *risk* settings. The *steps* parameter is the distance that is left to catch the target. In the absence of this parameter, the algorithms move to the last step to determine the kind of actions to execute. The *steps* parameter can avoid getting to the deadlock and instead attack the target without considering the cover action.

Figure 7.2 illustrates a bar chart of all pursuing algorithms and their

performance against SF and MPTM target algorithms. In all five sets of maps and five player combinations, CDMTA* with its variations is compared against CRA and STMTA*. In the first setting, CRA and CRA1 with *risk* parameters set to 0 and 0.1, respectively, are compared and unfortunately, CRA1 did not perform as expected in benchmarked maps and with moving multi-target scenarios. Second, CD1, CD2 and CD3 are compared. They use the existing formula for *delta-cost*, and their *risk* parameter is set to 0.1. CD2 and CD3 use improved action clearance and CD3 sets a 2-step for *steps* parameter. Among these three algorithms, CD2 outperforms the other two. Next is the comparison of CD4, CD5, CD6 and CD7. These methods differ in that they use the existing formula for *delta-cost* but with improved action clearance and the *steps* parameter is set to 0, 2, 3 and 5, respectively. The results display that CD4 has a better outcome. The final setting is between CD8, CD9, CD10 and CD11. It is similar to the previous setting and the only difference is that these algorithms apply the new revised formula for the *delta-cost* and CD8 achieves the best results in this setting.

The findings suggest that neither CRA nor CDMTA* showed outperforming outcomes when the *risk* parameter was set at 0.1. Despite using the assignment strategy, the STMTA* algorithm did not perform as well as CRA or CDMTA*. The results are not vastly different when STMTA* chases SF targets in contrast to MPTM, where STMTA* has the worst results with a significant difference. This might indicate that the optimal approach is not always to attack directly, instead, and as expected, the *cover heuristics* are promising with their results. The *steps* parameter displays some good performance in some individual player combinations on maps with narrow corridors when it is set to 2-step. It appears to perform better against the SF target algorithm. Similar behaviour is also observed occasionally in the CRA algorithm. Figure 7.2 shows that changing the *delta-cost* formula from the old to the new one did not show much effect on these experiments and the results demonstrate that CD4 and CD8 perform the best in all of these tests. There is a slight difference between these two, and overall, CD8 exhibits the lowest pathfinding cost.

Statistical tests are conducted to determine whether the pathfinding cost results are significantly different. The data obtained from the experiments are

not normally distributed, and the Friedman test [211, 212] is used for the analysis of all pursuing algorithms together but separately per target algorithms. Initially, Friedman uses ranking on the data set for each player combination setting on each map and ranks number 1 as the best-performing algorithm. All algorithms' ranks are obtained per map, and then the overall ranking value is summed for the final result as shown in Table 7.2.

In addition to rank values, the Friedman test obtains p-values by using these ranks and 0.5 is used as the level of significance. Table 7.3 presents the results for p-values. Although the overall pathfinding costs have small differences, there is very strong evidence suggesting that the results display statistically significant differences.

7.3.2.2 Minimum and Maximum Cost

Each test set is run 20 times and values for minimum cost and maximum cost are measured for the performance. The outcomes for the minimum and the maximum per target algorithm are shown in Table 7.4. Within the test sets, more occurrences of minimum outcomes and lower values for maximum reduce the pathfinding cost mean.

SF is a naive algorithm and due to its random movements, it is quicker to catch it when compared to smart MPTM. The results for SF demonstrate that the CDMTA* algorithms with 0 *risk* behave better than CRA or STMTA*, although the results are not inconsistent. Similar behaviour can be seen for MPTM, but the minimum cost for catching MPTM is harder by 45%. SF has been caught in all scenarios, thus there are no timeouts. However, on a few occasions, MPTM was able to avoid being captured by STMTA* and the CDMTA* algorithms when the *risk* parameter was set to 0.1 and the *steps* parameter was set to 3-step and 5-step. These are the situations where timeout exists, and it has been removed for the maximum results and indicated with an asterisk. Even with timeout results removed, the MPTM algorithm manages to escape the pursuers longer than SF by 90%.

In general, CRA1 has the worst performance in all maps. STMTA* does not have good performance either but it managed without using cover to display the

Table 7.2: Ranking of 14 algorithms based on pathfinding costs for two target algorithms. The bottom of the table displays the overall ranking. The best performance is ranked no. 1.

	CRA	CRA1	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	STMTA*
SF algorithm														
Sum of Ranking:	12785.5	21690	18777.5	18727	19089.5	11342.5	11509.5	12513.5	13858	11413	11981.5	12629	13857	19826.5
Rank	7	14	11	10	12	1	3	5	9	2	4	6	8	13
MPTM algorithm														
Sum of Ranking:	10744	21441.5	20301	19945	20810.5	10620	11852.5	12332	13887.5	10549	11723	12338.5	13509.5	19946
Rank	3	14	12	10	13	2	5	6	9	1	4	7	8	11
Average of Sums:	11764.8	21565.8	19539.3	19336.0	19950.0	10981.3	11681.0	12422.8	13872.8	10981.0	11852.3	12483.8	13683.3	19886.3
Rank	4	14	11	10	13	2	3	6	9	1	5	7	8	12

shortest routes on two occasions. In the first case, pursuers are grouped in one position while targets are dispersed on the AR0503SR map (Figure 7.4), while in the second case, pursuers and targets are aggregated in the same position but separated by a distance on the AR0512SR map. CD4 has the lowest result for the minimum but CD8 has the lowest result for the maximum, although there is a minor difference. There is a different graphical representation of the findings for minimum and maximum as illustrated in Figure 7.3. The details of minimum and maximum for SF from Table 7.4 are depicted in Figure 7.3(a) and similarly, the details of minimum and maximum for MTPM are from the same table in Figure 7.3(b). The bottom line represents the minimum, and the top line represents the maximum. The numbers between the lines inside the coloured area show the difference between the two. The mean for pathfinding cost may result in a smaller outcome when the top line is closer to the bottom line. Although

Table 7.3: The significance of pathfinding costs displays the p-values for SF (top) and MPTM (bottom) for each player combination per experimented map.

SF					
Maps	Player combinations				
	3 vs 2	3 vs 3	4 vs 2	4 vs 3	4 vs 4
AR0332SR	2.3E-62	3.2E-45	2.1E-62	3.7E-37	2.8E-44
AR0417SR	2.5E-51	4.8E-34	3.7E-33	1.5E-27	2.5E-19
AR0503SR	4.4E-26	6.8E-08	4.1E-36	4.5E-14	3.8E-14
AR0512SR	4.8E-60	6.5E-50	3.4E-118	4.3E-83	2.7E-77
AR0527SR	2.1E-31	1.5E-42	2.4E-78	1.7E-91	7.1E-103

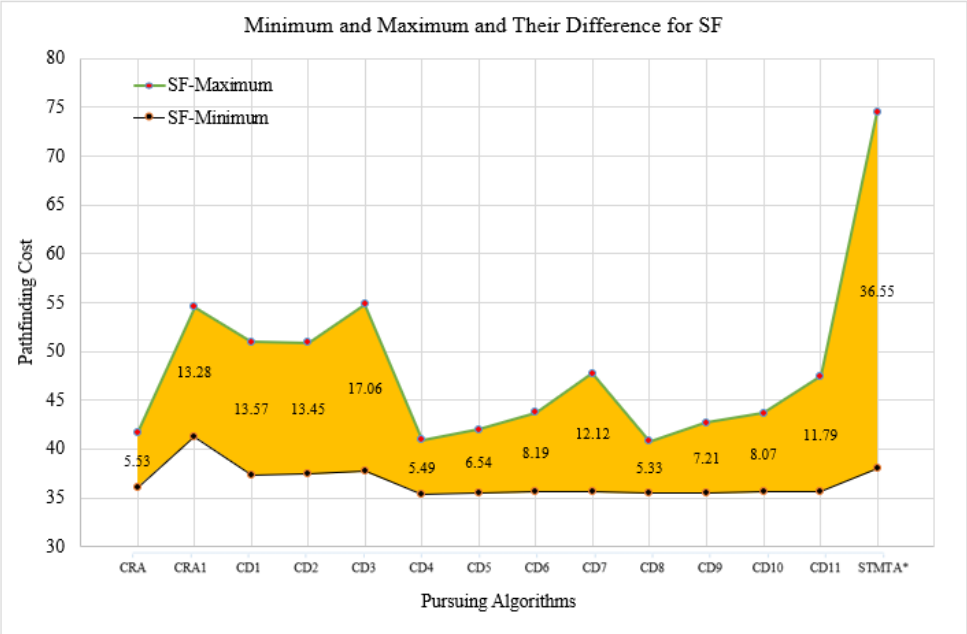
MPTM					
Maps	Player combinations				
	3 vs 2	3 vs 3	4 vs 2	4 vs 3	4 vs 4
AR0332SR	1.7E-68	4.7E-64	6.0E-62	7.4E-54	7.0E-38
AR0417SR	2.8E-96	6.2E-63	2.7E-129	1.8E-102	1.3E-104
AR0503SR	1.1E-06	1.5E-14	3.0E-23	2.0E-21	1.1E-55
AR0512SR	1.0E-27	5.6E-43	1.2E-24	7.3E-54	4.4E-77
AR0527SR	1.5E-88	8.0E-95	4.4E-89	4.6E-72	3.7E-77

the results differ for SF and MTPM, the shape of the coloured representation is similar. For instance, the graphs clearly display that algorithms with the *risk* parameters set to 0.1 do not perform as expected. Also, when the setting for the *steps* parameter has a larger number, it shows a decrease in performance.

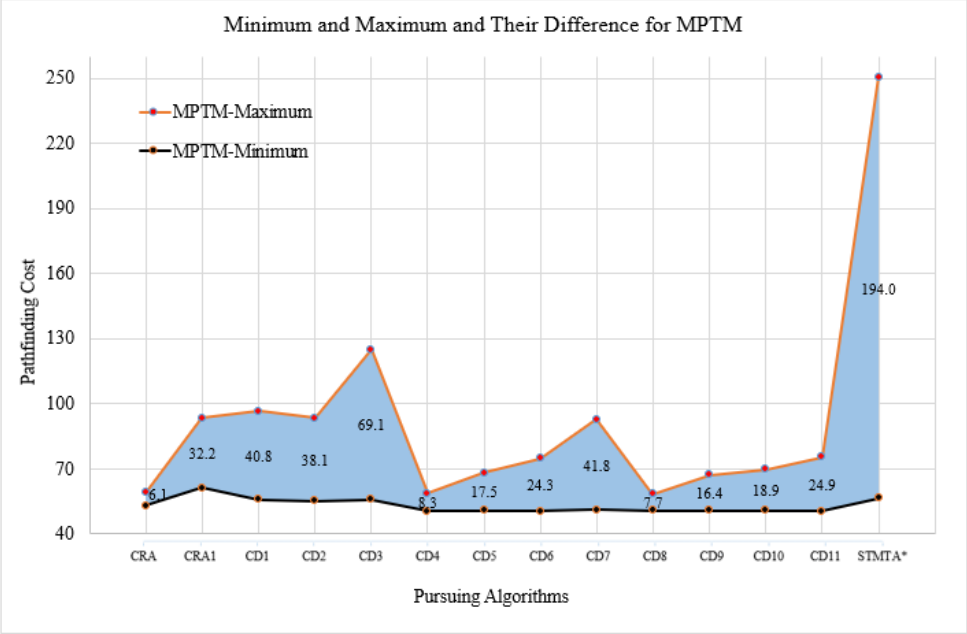
The results in Table 7.4 and Figure 7.3 suggest that the old formula for *delta-cost* works better for SF, whereas the new formula works better for a difficult target algorithm such as MPTM. CD8 has shown the best outcome when the average is taken between minimum and maximum. In the previous section, the pathfinding cost displayed overall the optimal result for CD8 and in this section the minimum and the maximum support the findings.

Table 7.4: Each algorithm's minimum cost and the maximum cost mean for all configurations. The asterisk indicates the removal of timeouts and substituted with the second maximum cost.

	SF		MPTM	
	Minimum	Maximum	Minimum	Maximum
CRA	36.07	41.60	53.09	59.17
CRA1	41.28	54.56	61.37	93.56*
CD1	37.37	50.94	55.86	96.64*
CD2	37.41	50.86	55.38	93.43*
CD3	37.73	54.79	56.02	125.1*
CD4	35.42	40.91	50.58	58.88
CD5	35.45	41.99	50.80	68.25
CD6	35.58	43.77	50.58	74.86*
CD7	35.62	47.74	51.07	92.82*
CD8	35.45	40.78	50.83	58.54
CD9	35.47	42.68	50.81	67.21
CD10	35.57	43.64	50.79	69.69*
CD11	35.65	47.44	50.59	75.48*
STMTA*	37.95	74.50	56.48	250.50*



(a) Minimum and maximum cost differences for SF



(b) Minimum and maximum cost differences for MPTM

Figure 7.3: The difference between minimum (bottom line) and maximum (top line) for each pursuing algorithm that is illustrated for (a) SF and (b) MPTM target algorithms.

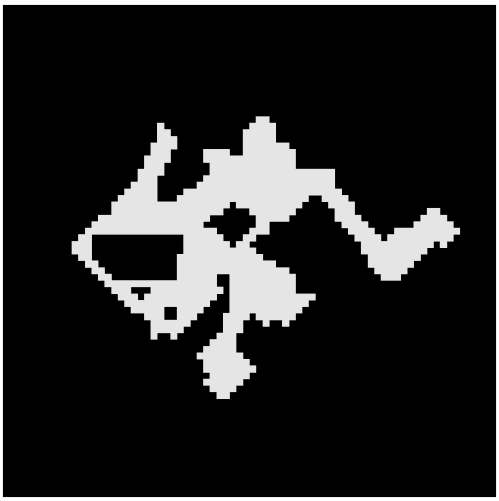


Figure 7.4: A sample AR0503SR map from Baldur's Gate video game.

7.3.2.3 Success Rate

The success rate is another performance indicator for multiple pursuers to capture all moving targets before reaching the maximum number of permitted steps. In the experiments, at least one pursuer is sufficient to occupy the target's position and claim success. Although the number of pursuers outnumbers or is equal to the targets, it is still possible for the targets to flee and avoid capture. In the previous Subsection 7.3.2.2, it was mentioned that there was not any timeout for the SF algorithm and the maximum was displayed without alteration. This indicates that in every scenario and in every test the SF algorithm was caught, and this has given a 100% success rate to all pursuers. On the other hand, the MPTM algorithm with its ability to avoid pursuers managed to escape in some experiments. The average success rate for each pursuing algorithm across all maps and player combinations on MPTM is illustrated in Figure 7.5.

Although this study is evaluating the performance of pursuing agents, it is worth mentioning that MPTM is a successful algorithm among the target algorithms as the previous studies in Chapter 2 suggested. CRA displayed 100% success in all experiments, but CRA1 has been successful only on maps with large open spaces and failed in some situations on maps with narrow corridors or island-style obstacles. Usually, CRA1 failed in situations where the number of pursuers and targets are equal and mostly happened once in the same

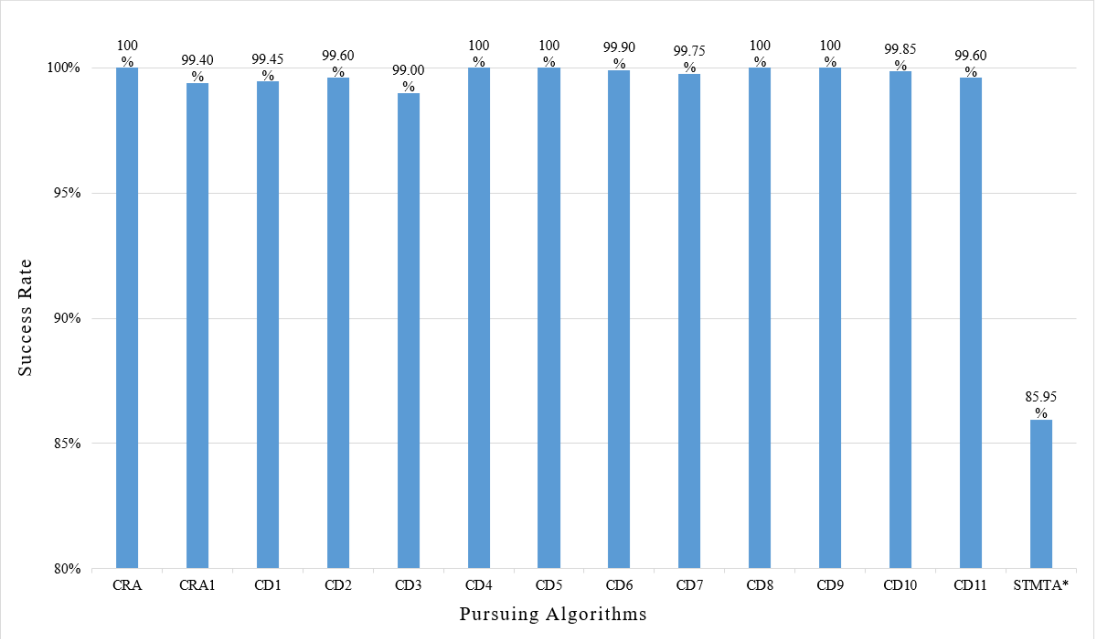


Figure 7.5: The success of the pursuing algorithm is only for MPTM.

starting state. The CDMTA* algorithm variations with the *risk* parameter set to 0.1 failed to achieve the 100% success rate. The CDMTA* algorithm variations with the *risk* parameter set to 0 perform better. When the *steps* parameter is set to 0-step or 2-step, it always catches the target and displays 100% success, but not when it is set to 3-step or 5-step. The failure to catch the target only happened a few times, most of the time once per player combination on two maps with narrow passages and island-style obstacles (AR0503SR and AR0527SR).

Each algorithm is tested 2,000 times on all maps and Figure 7.5 displays the mean for all 5 maps. The variations of CDMTA* sometimes failed to catch the targets mostly under 10 occasions and only once on 20 occasions for CD3. Unfortunately, the STMTA* algorithm demonstrated the worst performance. It failed on 281 occasions, and it makes an 85.95% success rate, which is the lowest performance among all. The behaviour of all algorithms per map produced results that were consistent, which means that the dimensions of the map, the forms of the static obstacles, the number of empty states and player combinations all provided similar outcomes. The *cover heuristic* algorithms such as enhanced CRA

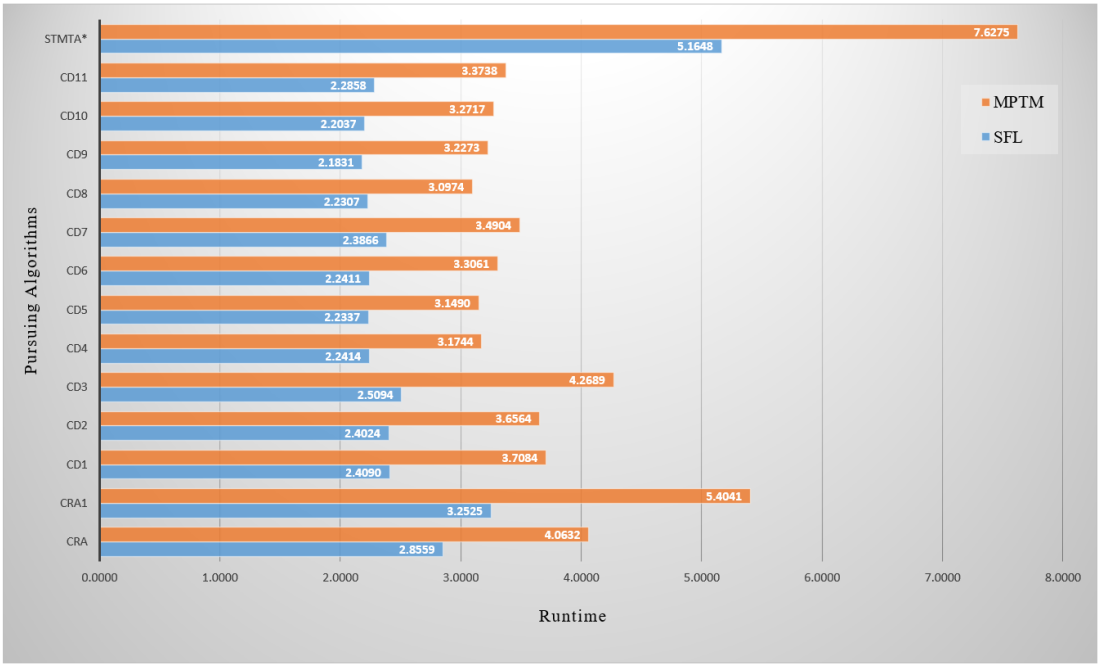


Figure 7.6: Runtime differences in seconds are displayed for all pursuing algorithms per target.

and CDMTA* with 0 or 2 *steps* parameter setting displayed excellent results, all 100% successful.

7.3.2.4 Runtime

Runtime is the measurement to evaluate the time for each algorithm during the same experiments that measured the pathfinding cost, minimum, maximum and success rate. Similar to the previous sections, the time is recorded for all pursuers per target algorithms and the mean is illustrated in Figure 7.6. The bar chart provides each algorithm's results, which are displayed in seconds. Table 2.3 contains information about the maps. Larger-sized maps and maps with more empty states increase the runtime, for instance, on average the AR0332SR map has the highest runtime and the AR0527SR has the lowest.

The graph in Figure 7.6 presents the mean of runtime for all player combinations and maps. When each pursuing algorithm is considered separately, it becomes clear that CRA, CRA1 or STMTA* have consistently

been slow on both target algorithms, unlike the CDMTA* algorithm and its variations. The results also show that the algorithms displayed poor-performing outcomes when the *risk* parameter was set to 0.1. The graph clearly shows that catching the MPTM algorithm requires more time than the SF algorithm, and of course, the timeouts also have an impact on the outcomes. When algorithms were evaluated on SF, CD9 performed well, and when comparable experiments were conducted on MPTM, CD8 displayed better results. Overall, CD8 produces the best performance.

7.4 Conclusion

Recently, there has been an increase in attempts to find solutions for PAMT problems. In this study, this problem is addressed not only to static multiple targets but to moving targets by using *cover heuristics*. The existing CRA algorithm has been enhanced to adapt to multiple targets and based on this algorithm, a modified, improved, and new CDMTA* algorithm is introduced. CDMTA* uses the assignment strategy algorithm to identify which targets to follow and if applicable re-assign as well as using the *cover heuristics*. It is enhanced with its dynamic *risk* parameter to address the problem with deadlocks, and two additional improvements one for action clearance and the second for the *delta-cost* formula for normalisation. The empirical analysis is conducted using CRA, STMTA* and CDMTA* with its variations.

Whilst the enhanced CRA algorithm has satisfactory outcomes, it is preferred that the new CDMTA* algorithm performs better. It was indicated that CRA behaves better when the *risk* parameter was set to 0.1, unfortunately in these benchmarked environments, the results were the opposite. For this reason, CRA is the benchmark for the new algorithms. In terms of pathfinding cost, the empirical evaluation displays that both algorithms are fast and have close results with a cost mean of 47.2 for CRA and 46.0 for CD8. Similar outcomes are seen for minimum and maximum mean, CRA with 47.5 and CD8 with 46.4, the lower the result, the better it is. The success rate is certainly 100% for both. These are promising results, but the CDMTA* algorithm and its variations are quicker than the CRA algorithm. The runtime is 3.4596 seconds for CRA and 2.6641 seconds

for CD8. Hence, it can be concluded that CRA and CD8, both are successful algorithms in catching targets and they always deliver 100% results. They have similar outputs but CD8 is slightly faster in terms of costs, as results displayed for the pathfinding cost, minimum, and maximum. However, in terms of runtime, CD8 is much quicker than CRA by 30%. Thus, CD8 is the CDMTA* algorithm with improved action clearance, the new formula for *delta-cost* and the assignment strategy algorithm (Adaptive Weighted-cost) can display a similar cost to CRA but it catches the moving targets quicker. It is possible to conclude that an algorithm can perform faster and produce optimal results when the assignment strategy is used with a combination of *cover heuristics*, that is the surrounding or outmanoeuvring strategy.

There are limitations in the number of pursuing agents used in the experiments. The PAMT problem is NP-hard and with more pursuers, it grows exponentially. As a future study, one way to solve the problem could be to subdivide the number of agents and then consider this subdivision as one team with assigning strategy per team. Another solution to the problem is to reduce the search space which also improves the computation costs. It can be solved by partitioning a map into four search spaces and pursuers within each portioned space are computed.

Chapter 8

Conclusion and Future Work

8.1 Thesis Summary

The work undertaken in this thesis has presented a range of novel contributions to search algorithms that are applicable in many real-world applications of multi-agent systems. The pursuing agents coordinate their efforts and employ assignment strategies to assign targets, which can improve performance and find a path for their actions that leads to the target with an optimal solution. The proposed approach enables two stages, coupled and decoupled, where pursuing agents were able to identify and assign targets in the first stage, and once the targets were assigned to each pursuer then they independently chased the moving targets in the second stage.

The thesis specifically focused on multi-agent pathfinding algorithms that chase multiple moving targets in a dynamic environment. The research presented in this thesis achieved that by investigating and implementing multi-agent algorithms in scenarios with moving targets, where the information about the position of the targets changes when they move to another state on the map. Thus, the change of positions requires computing a new path for both pursuing agents and targets at each time step. The approach for the new path used two different distance metrics: distance heuristics (the shortest path to rush-in) and cover heuristics (maximised covered area to surround). Although this research was conducted to provide an optimal solution for pursuing agents,

an additional new method was developed for a target that intelligently evades the pursuing agents.

In summary, throughout this research, this thesis contributes:

- An enhanced framework for MAS where multiple pursuers and multiple targets can be evaluated.
- A development of two novel pursuing agent algorithms that are capable of chasing moving targets.
- An investigation of assignment strategy algorithms to find an optimal criterion and implement six new assignment strategy algorithms.
- Extending the state-of-the-art target algorithm with proposes with the smart escaping method.
- A comprehensive experimental evaluation with various pursuer-to-target ratios on benchmarked gaming maps to measure the pathfinding cost, success rate and runtime.

The remaining sections in this chapter summarise the concluding remarks drawn from the thesis and outline a possible direction of some future research.

8.2 Concluding Remarks

The research presented in this thesis has demonstrated the plausibility of developing an enhanced framework for multi-agent systems that enables multiple agents and multiple targets in the same scenario. The thesis initially provided a simple solution to multi-agent pathfinding problems using an assignment strategy to assign targets and find a path to the moving targets by using repetitive A* searches. Then, efforts were made to explore novel ways of assigning targets and introduced six new assignment strategy algorithms that compute an optimal combination. Alongside, the thesis developed a novel algorithm for pursuing agents that can surround and outmanoeuvre multiple moving targets. Moreover, the contribution comprises one novel target

algorithm that can make smart moves to avoid capture. Besides all these algorithms, comprehensive experimental evaluations with various pursuer-to-target ratios on benchmarked gaming maps measured the pathfinding cost, success rate and runtime. The concluding remarks regarding the various aspects of the work in this thesis are presented in the remaining section.

8.2.1 Coordinating Multiple Agents with Assignment Strategy to Pursue Multiple Moving Targets

In Chapter 3, the new STMTA* algorithm was proposed to find the solution for multiple pursuers in a dynamic environment while chasing multiple moving targets. The presented algorithm is divided into two approaches, coupled and decoupled. The coupled approach coordinates all pursuers to find the combination using the assignment strategy algorithm which runs all possible pursuer-to-target combinations at the initial position using the given criteria and the optimal combination was selected that assigned one target to each pursuer. In the decoupled approach the pursuers independently computed their path towards the moving target at each time step.

The proposed approach in this thesis was investigated more thoroughly in this chapter on multi-agent and multi-target scenarios using benchmark environments. Detailed experiments were carried out using ten purposely made and commercial gaming benchmark testbeds with six different player combinations. During these experiments, the number of steps and success rate were measured, and for statistical analysis, the Wilcoxon rank-sum test was applied. The study in this chapter established that the proposed algorithm captured targets quicker and produced a higher rate of success, approximately by 16%, in scenarios with multiple moving targets. This algorithm, therefore, provided a useful approach in dealing with scenarios where multiple moving targets were present

8.2.2 Multi-agent Path Planning Approach Using Assignment Strategy Variations in Pursuit of Moving Targets

The research presented in Chapter 4 investigated and identified more new alternative methods for assignment strategies in multi-agent scenarios in order to increase efficiency. The proposed methods such as Twin-cost, Cover-cost and Weighted-cost criteria have been experimented with, and performance analysis measured the number of steps travelled and the success of completed test runs on grid-based gaming maps. These findings highlighted the potential usefulness of these methods and evaluated the strengths and weaknesses during the experiments. It was suggested that the use of the proposed criteria made the algorithms more efficient than those currently used including the Mixed-cost criterion. The results obtained from experiments suggested that the Weighted-cost criteria depending on parameters have performed the best.

8.2.3 A Strategy-based Algorithm for Moving Targets in an Environment with Multiple Agents

The work presented in Chapter 5 of this thesis was to provide a solution for multi-agent multi-target pathfinding problems and develop a target algorithm that would consider multiple pursuers and make a smart escape. Numerous interesting studies have been conducted on search algorithms, and among them were solutions to the frameworks with multiple pursuing agents. However, only a few studies have been carried out on target algorithms, especially in multi-target environments.

This research in this chapter showed that TrailMax is a successful algorithm for control of targets if developed further for dealing with multiple pursuers. The amendments were proposed to the TrailMax algorithm to make it work as a strategy for multi-agent multi-target search problems in dynamic environments.

The resulting MPTM algorithm has been shown to outperform other target algorithms for the same scenario, and that could make pursuit and evasion scenarios in computer games more challenging, meaningful, and interesting. The

results clearly showed that the MPTM algorithm performed far better, with at least doubling capture cost and escaping success by 13% on the gaming maps used for benchmarking.

8.2.4 Adaptive Weighted-Cost Assignment Strategy for Efficient Multi-Agent Path Planning

The study presented in Chapter 6 proposed new approaches for coupled planning in developing assignment strategy algorithms for multiple pursuing agents in the dynamic environment. The proposed methods such as Adaptive Weighted-cost, Joint Weighted-cost and Joint Twin-cost algorithms provided combinations that help to increase the performance by reaching the targets most cost-effectively. These newly introduced algorithms were analysed and compared against the existing overall best approach, the Weighted-cost algorithm which was introduced in Chapter 4.

The proposed approaches in this thesis were experimentally evaluated in terms of the pathfinding cost, the minimum cost per test set, the success of the completed test runs and the computation time for assigning targets. The results highlighted that using dynamic weight parameters for the Adaptive Weighed-cost algorithm with the adjusted method displayed the best overall performance. Furthermore, it succeeded with a rate of 99.9% and was proven to be statistically significant. To conclude, the new proposed approach was more efficient and gave better performance over other assignment strategy algorithms.

8.2.5 Increasing Covered Area to Capture Moving Targets in a Dynamic Environment

The study in Chapter 7 of this thesis employed *cover heuristics* to address the problem of moving targets in addition to static multiple targets. The existing CRA algorithm was modelled only to one target, and it was enhanced to adapt to multiple targets. Hence, a modified, improved, and new CDMTA* algorithm was introduced. CDMTA* used the assignment strategy algorithm, which was introduced in Chapter 6, to identify which targets to follow and if applicable

re-assign as well as using the *cover heuristics*. It was enhanced with its dynamic *risk* parameter to address the problem with deadlocks, and two additional improvements one for action clearance and the second for the *delta-cost* formula for normalisation. The empirical experiments were conducted using CRA, STMTA* and CDMTA* with its variations. CRA with default parameter settings was the benchmark for the new algorithms. In terms of pathfinding cost, the empirical evaluation displayed that both algorithms were fast and had close results with a cost mean. Similar outcomes were seen for minimum and maximum mean values. The success rate was certainly 100% for both. Besides, these results were promising, the CDMTA* algorithm and its variations were quicker than the CRA algorithm. It was possible to conclude that an algorithm could perform faster and produce optimal results when the assignment strategy was used with a combination of *cover heuristics*, that was the surrounding or outmanoeuvring strategy.

8.3 Limitations

This section states some of the current limitations of the presented algorithms for pursuing agents, assignment strategies and the target algorithm. Current pathfinding solutions in this thesis solve the problem while searching the whole

Table 8.1: The total number of combinations is required for the assignment strategy algorithm per agent count.

Number of Pursuers	Total Combinations
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800

path before even making any move. The reason for this is that, if the path from the starting position to the goal position is not found, the optimal first step cannot be guaranteed. For a better optimal solution, the assignment strategy algorithms improve the performance, however, with multiple pursuers, it is an exhaustive computation process. Table 8.1 displays the number of pursuers and the total number of combinations to assign targets to the pursuers. It can be seen that the growth is exponential with the number of pursuers. The assignment strategy algorithm is a coupled approach and finding an optimal combination for 5 pursuers is a difficult task and with limited computer resources it becomes impossible for 10 pursuers that require millions of combinations.

The complexity of the assignment strategy algorithm is dominated by the total number of pursuers. In the experiments, the algorithm can efficiently explore pathfinding solutions that involve up to 5 pursuers. This limitation is not critical in practice, for instance, it can be well suited to the Perfect Heist 2 [213] cop and robber video game where the setting of two to four players on each team works the best. Moreover, it is not known a solution or alternative approach to scale the problem in coupled state for multiple pursuing agents. However, this would seem that the coupled approach does not work necessary in some multi-agent problem instances such as robotics, where a large number of robots solve the problem by decoupling into a set of sub-problems [13] or merging their plan to the current coordinated plan sets [214].

Similarly, the target algorithm, MPTM, considers multiple pursuers and evaluates their position for its escape to a further position on the map. The suggested smart moves in Chapter 5 require improving the computation process as it is exhaustive and intensive with larger player combinations.

Each assignment strategy algorithm assigns targets at the current position of pursuers before the test run starts and this leads to chasing the initial assigned target while another target might be nearby which could potentially change the outcome. However, Chapter 6 introduces a more dynamic procedure to periodically stop and evaluate the new positions of yet non-caught targets and if needed re-assign targets. Even though this might be a good idea, unfortunately, the experiments demonstrated that this repetitive assessment increases the computation cost on the maps with larger open spaces where the distance to the

targets is far away. This is time-consuming and resource-engaging for the whole team of pursuers.

Section 2.5.3 mentions a constraint that allows pursuing agents to be in the same position as other agents. During the experiments, sometimes this can continue longer until the pursuers spread out or capture the target. This limits correctly utilising the resources and delays the success of the results. Pursuers should be required to split and surround targets if they happen to be in the same position for specified time steps.

There is also a limitation of experiments when the pursuers are outnumbered by targets. For instance, when there are not enough resources in search and rescue situations or in warehouse management operations with insufficient resources to allocate tasks to the agents and then complete them.

Although the experiments were carried out on benchmarked gaming maps, large-sized gaming maps or city-shaped maps [215] derived from actual capital cities of the world can be considered with respect to pursuer-to-target combinations and computing resources. It is possible to scale to the larger maps and it is achievable with a decoupled approach which subdivides pathfinding problems into smaller search problems [197]. However, the experiments were carried out on various-sized Baldur's Gate gaming maps in [114] where the results demonstrate that the total pre-processing time for larger maps is about 5 times slower than other maps.

It is not possible to use pursuer (STMTA* or CDMTA*) or target (MPTM) algorithms in large-scale real-time applications, due to their computation cost and also they cannot perform an action unless the whole path towards the target is known. As a result of this unavoidable cost, the optimal solutions and scalability of problems are restricted to relatively non-large problems in practice.

8.4 Directions for Future Works and Recommendations

Based on the work presented in this thesis, this section identifies several potential directions for future research and recommendations:

8.4.1 Assignment Strategy

- A further direction for studies needs to explore methods to reduce the search space for algorithms and improve computational costs. One of the possible ways to reduce the computation cost is to compute all possible combinations for the agents offline and provide the best combination in advance.

- Another interesting area is to dynamically evaluate the assignment strategy without the predefined number of steps parameter. The algorithm measures the distance and assesses the combination, if a different combination is found then the targets are re-assigned.

- Further studies should be undertaken to confirm the success of new assignment strategies that are introduced in Chapter 4 and Chapter 6 with thorough experiments on the problem introduced in this thesis. Moreover, the possible experiments could be on MAPF problems where targets are static.

8.4.2 Multi-Agent Algorithms

- A potential direction to extend the work of this thesis is to find another solution to the problem which reduces the search space and also improves the computation costs. It can be solved by partitioning a map into four search spaces and pursuers within each portioned space are computed. A similar study conducted an offline computation of the map for pathfinding with compressed path databases [114] or reduced the size of the search space [34, 216].

- There are limitations in the number of pursuing agents used in the experiments. The problem with multiple pursuing agents is NP-hard and with more pursuers, it grows exponentially. As a future study, one way to solve the problem could be to subdivide the number of agents and then consider this subdivision as one team with an assigned strategy per team.

- It will be interesting to extend the work to develop a paradigm with various responsibilities, where different roles and strategies are assigned to

the pursuers, and/or add more mechanisms for coordination between them.

- Although this thesis presented comprehensive experiments with various pursuing algorithms and target algorithms, the experimental evaluation conducted between pursuers and targets was the chase, on grid-based benchmarked gaming maps. However, further exploration of the behaviour and evaluation of the performance of novel multi-agent algorithms on directed maps, such as road maps, could be a study of future work.

- During the empirical evaluation, the constraints can be relaxed for search algorithms, such that pursuing agents or targets are restricted and not allowed to occupy the same state; the environment can have moving obstacles; the size of agents could vary, for instance, in military video games a soldier can occupy one space where trucks or tanks will need a few; or in disaster search and rescue operations, the travelling speed of players could be various, and the number of agents can be less than many survivors.

8.4.3 Target Algorithms

- The issue of comparatively high computational costs could be explored in further research, for example, by exploring the use of heuristics that cut off parts of the search space or removing symmetric path segments from grid maps [217, 218, 219].

- Although the study in Chapter 5 focused on a single target's evasion from multiple pursuing agents, further investigation to extend the MPTM algorithm to collaborate with other targets would be very interesting research.

Appendix A

Here is the list of all algorithms presented in the thesis displayed in the tables below. The targets are in Table A1 and the pursuing algorithms are in Table A2.

Table A1: List of target algorithms mentioned in the thesis.

No	Algorithm Name	Year
1	Greedy	1950s
2	Minimax	2006
3	Dynamic Abstract Minimax (DAM)	2006
4	Simple Flee (SF)	2008
5	TrailMax	2009
6	Multiple Pursuers TrailMax (MPTM)	2022

Table A2: List of pursuing algorithms mentioned in thesis and ordered by year.

No	Algorithm Name	Year	Agents	Type		Target or Goal Destination	
			Single Multiple	Incremental	Real-Time	Fixed Dynamic	Single Multiple
1	A*	1968	single		yes	fixed	single
2	Learning Real-Time A*(LRTA*)	1990	single		yes	dynamic	single
3	Real-Time A* (RTA*)	1990	single		yes	dynamic	single
4	Moving Target Search (MTS)	1991	single		yes	dynamic	single
5	Moving Target Search with Commitment and Deliberation	1992	single		yes	dynamic	single
6	Dynamic A* (D*)	1994	single	yes			single
7	Focused Dynamic A*	1995		yes			single
8	D* Lite	2002	single	yes		fixed	single
9	Lifelong Planning A* (LPA*)	2002		yes			single
10	Anytime A*	2002	single	yes			single
11	Multiple Agents Moving Target (MAMT)	2003	multiple		yes	dynamic	
12	Anytime Repairing A* (ARA*)	2003	single	yes			single
13	Adaptive A*	2005	single	yes		fixed	single
14	Anytime Dynamic A*	2005					
15	Hierarchical Cooperative A* (HCA*)	2005	multiple		yes		
16	Partial-Refinement A* (PRA*)	2005	single			dynamic	single
17	Window Hierarchical Cooperative A* (WHCA*)	2005	multiple		yes	dynamic	single
18	Local Repair A* (LRA*)	2005	single				single
19	Real-Time Adaptive A*	2006	single		yes	dynamic	single
20	MT-Adaptive A*	2007	single	yes			single
21	Anytime Weighted A*	2007	single	yes			single
22	Real-Time Moving Target Evaluation Search (MTES)	2007	single		yes		single
23	Efficient Path Planning (eMIP)	2007	multiple				
24	Cover with Risk and Abstraction (CRA)	2008	multiple		yes	dynamic	
25	Flow Annotation Replanning (FAR)	2008	multiple		yes		
26	Generalized Adaptive A* (GAA*)	2008	single	yes		fixed, dynamic	single
27	Abstraction MTS (A-MTS)	2009	multiple		yes	fixed	single
28	Fringe-Retrieving A*	2009	single	yes			single
29	Fuzzy MTS (F-MTS)	2009	multiple		yes	fixed	single
30	Path Adaptive A* (Path-AA*)	2009	single	yes			single

Table A2 continued from previous page

No	Algorithm Name	Year	Agents	Type		Target or Goal Destination	
			Single Multiple	Incremental	Real-Time	Fixed Dynamic	Single Multiple
31	Generalized Fringe-Retrieving A* (G-FRA*)	2010	single	yes			single
32	Moving Target D* Lite	2010	single	yes		dynamic	single
33	Field D*	2010	single	yes			single
34	Multi-Target Adaptive A*	2010	multiple	yes			multiple
35	Independence Detection (ID)	2010	multiple			fixed	multiple
36	Operator Decomposition (OD)	2010	multiple			fixed	multiple
37	Conflict Based Search (CBS)	2011	multiple			fixed	multiple
38	Tree Adaptive A* (Tree-AA*)	2011	single	yes			single
39	Incremental Anytime Repairing A* (I-ARA*)	2012	single	yes			single
40	Time-Bounded Adaptive A* (TBAA*)	2012	single	yes			single
41	Meta-Agent CBS (MA-CBS)	2012	multiple			fixed	multiple
42	Moving Target Search with Compressed Paths (MtsCopa)	2013	single	yes		dynamic	
43	Agent Decomposition Planner (ADP)	2013	multiple				single
44	Multipath Adaptive A* (MPAA*)	2014	multiple	yes			
45	Multipath Generalized Adaptive A* (MPGAA*)	2015	single	yes		dynamic	
46	Moving Target Search with Subgoal Graphs (MTSub)	2015	single	yes		dynamic	
47	Distributed Multi-agent Path Planning (DMAPP)	2015	multiple				
48	Meet in the Middle (MM)	2016	multiple			fixed	single
49	Conflict-Based Min-Cost-Flow (CBM)	2016	multiple			fixed	multiple
50	Distributed Multi-agent Path Planning (DiMPP)	2017	multiple		yes	fixed	
51	Token Passing (TP)	2017	multiple			fixed	multiple
52	Token Passing with Task Swap (TPTS)	2017	multiple			fixed	multiple
53	Multi-label A* (MLA*)	2019	multiple				single
54	Task Assignment and Prioritized (TA-Prioritized)	2019	multiple				
55	Task Assignment and Hybrid (TA-Hybrid)	2019	multiple				
56	Multi-Directional Meet in the Middle (MM*)	2020	multiple			fixed	single
57	Multi-Objective Path-Based D* Lite (MOPBD*)	2022	multiple	yes			multiple
58	Strategy Multiple Target A* (STMTA*)	2022	multiple		yes	dynamic	multiple
59	Cover Dynamic Moving Target A* (CDMTA*)	2022	multiple		yes	dynamic	multiple

References

[1] Rachel Kirby, Reid Simmons, and Jodi Forlizzi, "Variable sized grid cells for rapid replanning in dynamic environments", in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. IEEE, 2009, pp. 4913–4918.

[2] Dave Ferguson and Anthony Stentz, "The Delayed D* algorithm for efficient path replanning", in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA*, April 2005, pp. 2045–2050.

[3] Damien Pellier, Humbert Fiorino, and Marc Métivier, "Planning when goals change: A moving target search approach", in *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, Yves Demazeau, Franco Zambonelli, Juan M. Corchado, and Javier Bajo, Eds., vol. 8473 of *Lecture Notes in Computer Science*, pp. 231–243. Springer, Salamanca, Spain, 2014.

[4] Jingjin Yu and Steven M LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics", *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[5] Roni Stern, "Multi-agent path finding – an overview", in *Artificial Intelligence. Lecture Notes in Computer Science*, Gennady S. Osipov, Aleksandr I. Panov, and Konstantin S Yakovlev, Eds., vol. 11866, pp. 96–115. Springer International Publishing, Cham, 2019.

[6] Keisuke Okumura, Yasumasa Tamura, and Xavier Défago, "Time-independent planning for multiple moving agents", in *Proceedings of the*

35th AAAI Conference on Artificial Intelligence. 2021, vol. 35, p. 11299 – 11307, AAAI Press.

[7] Hang Ma, "Graph-based multi-robot path finding and planning", *Current Robotics Reports*, vol. 3, pp. 77–84, 2022.

[8] Simon M. Lucas, "Computational intelligence and games: Challenges and opportunities", *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008.

[9] Daniel Johnson and Janet Wiles, "Computer games with intelligence", in *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, 2001, vol. 3, pp. 1355–1358.

[10] Georgios N. Yannakakis and Julian Togelius, "A panorama of artificial and computational intelligence in games", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, pp. 317–335, 2015.

[11] Stuart Russell and Peter Norvig, *Artificial intelligence: A modern approach (Global edition)*, Pearson Education Limited, 4 edition, 2021.

[12] Stephen Kloder and Seth Hutchinson, "Path planning for permutation-invariant multirobot formations", *IEEE Transactions on Robotics*, vol. 22, pp. 650–665, 2006.

[13] Jur Van Den Berg, Jack Snoeyink, Ming Lin, and Dinesh Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans", in *Proceedings of Robotics: Science and Systems*, Jeff Trinkle, Yoky Matsuoka, and Jose A. Castellanos, Eds., vol. 5, p. 137 – 144. MIT Press Journals, Seattle, USA, 2009.

[14] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig, "New techniques for pairwise symmetry breaking in multi-agent path finding", in *Proceedings of the 13th International Symposium on Combinatorial Search, SoCS*, 2020, vol. 30, pp. 193–201.

[15] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, T K Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon,

"Overview: Generalizations of multi-agent path finding to real-world scenarios", in *Proceedings of the 2nd International Workshop on Multi-Agent Path Finding co-located with 25th International Joint Conference on Artificial Intelligence, IJCAI*, 2016, pp. 1–4.

[16] Yiming Liu, Mengxia Chen, and Hejiao Huang, "Multi-agent pathfinding based on improved cooperative A* in Kiva system", in *2019 5th International conference on control, automation and robotics, ICCAR*, 2019, pp. 633–638.

[17] Arturo Rankin, Mark Maimone, Jeffrey Biesiadecki, Nikunj Patel, Dan Levine, and Olivier Toupet, "Mars curiosity rover mobility trends during the first 7 years", *Journal of Field Robotics*, vol. 38, pp. 759 – 800, 2021.

[18] Neil Abcouwer, Shreyansh Daftry, Tyler Del Sesto, Olivier Toupet, Masahiro Ono, Siddarth Venkatraman, Ravi Lanka, Jialin Song, and Yisong Yue, "Machine learning based path planning for improved rover navigation", in *IEEE Aerospace Conference Proceedings*, 2021, vol. 2021-March.

[19] Liviu Panait and Sean Luke, "Cooperative multi-agent learning: The state of the art", *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[20] Hazem El-Alfy and Amr Kabardy, "A new approach for the two-player pursuit-evasion game", in *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI*, 2011, pp. 396–397.

[21] Fedor V Fomin, Petr A Golovach, and Daniel Lokshtanov, "Cops and robber game without recharging", *Theory of Computing Systems*, vol. 50, no. 4, pp. 611–620, 2012.

[22] Toru Ishida and Richard E. Korf, "Moving target search", in *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI*, 1991, vol. 1, pp. 204–210.

[23] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses", *AI Magazine*, vol. 29, no. 1, pp. 9, 2008.

[24] Michal Čáp, Peter Novák, Jiří Vokřínek, and Michal Pěchouček, "Multi-agent RRT*: Sampling-based cooperative pathfinding", in *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, Maria Gini, Onn Shehory, Takayuki Ito, and Catholijn Jonker, Eds., Saint Paul, Minnesota, USA, 2013, vol. 2, p. 1263–1264, International Foundation for Autonomous Agents and Multiagent Systems, (IFAAMAS).

[25] Yugang Liu and Goldie Nejat, "Robotic urban search and rescue: A survey from the control perspective", *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 72, no. 2, pp. 147–165, 2013.

[26] Andreas Kolling, Alexander Kleiner, Michael Lewis, and K. Sycara, "Computing and executing strategies for moving target search", in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation, ICRA*, 2011, pp. 4246–4253.

[27] Toru Ishida, "Moving target search with intelligence", in *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI*, 1992, vol. 92, pp. 525–532.

[28] Sven Koenig and Maxim Likhachev, "D* Lite", in *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI*, 2002, vol. 15, pp. 476–483.

[29] Çağatay Ündeğer and Faruk Polat, "RTTES: Real-time search in dynamic environments", *Applied Intelligence*, vol. 27, no. 2, pp. 113–129, 2007.

[30] Sven Koenig and Maxim Likhachev, "Adaptive A*", in *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, New York, NY, USA, 2005, p. 1311–1312, Association for Computing Machinery.

[31] Ko Hsin Cindy Wang and Adi Botea, "Fast and memory-efficient multi-agent pathfinding", in *ICAPS 2008 - Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 2008, p. 380 – 387.

[32] David Silver, "Cooperative pathfinding", in *Proceedings of the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2005, vol. 1, pp. 117–122.

[33] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant, "Meta-agent conflict-based search for optimal multi-agent path finding", in *Proceedings of the 5th International Symposium on Combinatorial Search, SoCS*, 2012, p. 97 – 104.

[34] Nathan Sturtevant and Michael Buro, "Partial pathfinding using map abstraction and refinement", in *Proceedings of the 20th National Conference on Artificial Intelligence, AAAI*, Pittsburgh, Pennsylvania, 2005, vol. 3, pp. 1392–1397, AAAI Press.

[35] Mark Goldenberg, Alexander Kovarsky, Xiaomeng Wu, and Jonathan Schaeffer, "Multiple agents moving target search", in *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI*, 2003, p. 1536 – 1538.

[36] Peter K.K. Loh and Edmond C. Prakash, "Novel moving target search algorithms for computer gaming", *Computers in Entertainment (CIE)*, vol. 7, no. 2, pp. 1–16, 2009.

[37] Lucia Pallottino, Vincenzo G. Scordio, Antonio Bicchi, and Emilio Frazzoli, "Decentralized cooperative policy for conflict resolution in multi-vehicle systems", *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1170–1183, 2007.

[38] Michael Wooldridge, *An introduction to multiagent systems*, John wiley & sons, Hoboken, N.J. : Chichester, 2 edition, 2009.

[39] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge, "A roadmap of agent research and development", *Autonomous Agents and Multi-Agent Systems*, vol. 1, pp. 7 – 38, 1998.

[40] Balaji Parasumanna Gokulan and Dipti Srinivasan, "An introduction to multi-agent systems", in *Innovations in multi-agent systems and applications-1*, Dipti Srinivasan and Lakhmi C. Jain, Eds., pp. 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[41] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Pearson Education Limited, Harlow, United Kingdom, 3 edition, 2016.

[42] Jorge Rocha, Inês Boavida-Portugal, and Eduardo Gomes, "Introductory chapter: Multi-agent systems", in *Multi-agent Systems*, Jorge Rocha, Ed., chapter 1. IntechOpen, Rijeka, 2017.

[43] Vicente Julian and Vicente Botti, "Multi-agent systems", *Applied Sciences*, vol. 9, no. 7, pp. 1402, 2019.

[44] Gerhard Weiss, *Multiagent Systems*, The MIT Press, 2 edition, 2013.

[45] Ross Graham, Hugh Mccabe, and Stephen Sheridan, "Pathfinding in computer games", *The ITB Journal*, vol. 4, pp. 57–81, 2003.

[46] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan Sturtevant, Robert C. Holte, and Jonathan Schaeffer, "Enhanced partial expansion A*", *Journal of Artificial Intelligence Research*, vol. 50, pp. 141–187, 2014.

[47] Sven Koenig, "A comparison of fast search methods for real-time situated agents", in *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS*. IEEE Computer Society, 2004, vol. 3, pp. 864–871.

[48] Ryan Luna and Kostas E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees", in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, 2011, p. 294 – 300.

[49] Pavel Surynek, "Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems", *Annals of Mathematics and Artificial Intelligence*, vol. 81, no. 3-4, pp. 329–375, 2017.

[50] Hang Ma and Sven Koenig, "AI buzzwords explained: Multi-agent path finding (MAPF)", *AI Matters*, vol. 3, pp. 15–19, 2017.

[51] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner, "The increasing cost tree search for optimal multi-agent pathfinding", in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, 2011, p. 662 – 667.

[52] Aysu Bogatarkan, Volkan Patoglu, and Esra Erdem, "A declarative method for dynamic multi-agent path finding", in *Proceedings of the 5th Global Conference on Artificial Intelligence, (GCAI)*, Diego Calvanese and Luca Iocchi, Eds., vol. 65 of *EPiC Series in Computing*, pp. 54–67. EasyChair, Bozen/Bolzano, Italy, 2019.

[53] Fan Xie, Adi Botea, and Akihiro Kishimoto, "A scalable approach to chasing multiple moving targets with multiple agents", in *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI*, 2017, vol. 0, p. 4470 – 4476.

[54] Malcolm R.K. Ryan, "Exploiting subgraph structure in multi-robot path planning", *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, 2008.

[55] Malcolm Ryan, "Graph decomposition for efficient multi-robot path planning", in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, pp. 2003–2008.

[56] Anders Jonsson and Michael Rovatsos, "Scaling up multiagent planning: A best-response approach", in *ICAPS 2011 - Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011, p. 114 – 121.

[57] Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Štolba, "Cooperative multi-agent planning: A survey", *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[58] Daniel Borrajo, "Multi-agent planning by plan reuse", in *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2013, vol. 2, p. 1141 – 1142.

[59] Dor Atzmon, Jiaoyang Li, Ariel Felner, Eliran Nachmani, Shahaf Shperberg, Nathan Sturtevant, and Sven Koenig, "Multi-directional heuristic search", in *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI*, 2020, pp. 4062–4068.

[60] Alejandro Isaza, Jieshan Lu, Vadim Bulitko, and Russell Greiner, "A cover-based approach to multi-agent moving target pursuit", in *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*, 2008, p. 54 – 59.

[61] Carsten Moldenhauer and Nathan R. Sturtevant, "Optimal solutions for moving target search", in *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2009, vol. 2, pp. 1249–1250.

[62] Satyendra Singh Chouhan and Rajdeep Niyogi, "DiMPP: a complete distributed algorithm for multi-agent path planning", *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 29, no. 6, pp. 1129–1148, 2017.

[63] Trevor Scott Standley and Richard E. Korf, "Complete algorithms for cooperative pathfinding problems", in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, 2011, p. 668 – 673.

[64] Malcolm Ryan, "Multi-robot path-planning with subgraphs", in *Proceedings of the 19th Australasian Conference on Robotics and Automation, ACRA*, 2006, pp. 1–8.

[65] Craig Tovey, Michail G. Lagoudakis, Sonal Jain, and Sven Koenig, "The generation of bidding rules for auction-based robot coordination", in *Multi-Robot Systems. From Swarms to Intelligent Automata - Proceedings from the 2005 International Workshop on Multi-Robot Systems*, 2005, vol. 3, p. 3 – 14.

[66] Jonathan Vermette, "A survey of path-finding algorithms employing automatic hierarchical abstraction", *Journal of the Association for Computing Machinery*, vol. 377, pp. 383, 2011.

[67] Meir Goldenberg, Ariel Felner, Alon Palombo, Nathan Sturtevant, and Jonathan Schaeffer, "The compressed differential heuristic", *AI Communications*, vol. 30, pp. 393 – 418, 2017.

[68] Ian Millington and John Funge, *Artificial intelligence for games*, CRC Press, 2 edition, 2009.

[69] Matthew David Crosby, *Multiagent classical planning*, PhD thesis, The University of Edinburgh, Edinburgh, UK, 2014.

[70] Blizzard Entertainment, "World of warcraft", Microsoft Windows, 2004.

[71] Westwood Studios, "Command and conquer", Microsoft Windows, 1995.

[72] Nathan R. Sturtevant, Devon Sigurdson, Bjorn Taylor, and Tim Gibson, "Abstraction and refinement in games with dynamic weighted terrain", in *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 2020, vol. 34, pp. 13697–13699.

[73] Kaushlendra Sharma and Rajesh Doriya, "Path planning for robots: an elucidating draft", *International Journal of Intelligent Robotics and Applications*, vol. 4, pp. 294 – 307, 2020.

[74] Omar Souissi, Rabie Benatitallah, David Duvivier, Abedlhakim Artiba, Nicolas Belanger, and Pierre Feyzeau, "Path planning: A 2013 survey", in *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management, IEEE - IESM 2013*, 2013, pp. 1–8.

[75] Enric Galceran and Marc Carreras, "A survey on coverage path planning for robotics", *Robotics and Autonomous Systems*, vol. 61, pp. 1258–1276, 2013.

[76] Wikipedia contributors, "Euclidean distance — Wikipedia, the free encyclopedia", 2023, [Online; accessed 24-July-2023].

[77] Daniel Rolf Wichmann, "Automated route finding on digital terrains", Master's thesis, University of Auckland, Auckland, New Zealand, 2004.

[78] Carsten Moldenhauer and Nathan R. Sturtevant, "Evaluating strategies for running from the cops", in *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI*, 2009, p. 584 – 589.

[79] Çağatay Ündeğer and Faruk Polat, "Multi-agent real-time pursuit", *Autonomous Agents and Multi-Agent Systems*, vol. 21, pp. 69–107, 2010.

[80] Xiaoxun Sun, William Yeoh, and Sven Koenig, "Moving target D* Lite", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2010, vol. 1, pp. 67–74.

[81] Thierry Siméon, Stéphane Leroy, and Jean-Paul Laumond, "Path coordination for multiple mobile robots: A resolution-complete algorithm", *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 42–49, 2002.

[82] Richard E. Korf, "Artificial intelligence search algorithms", in *Algorithms and Theory of Computation Handbook*, Mikhail J. Atallah, Ed., Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press, 1999.

[83] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[84] Wikipedia contributors, "A* search algorithm — Wikipedia, the free encyclopedia", 2023, [Online; accessed 7-July-2023].

[85] Çağatay Ündeğer, *Single and multi agent real-time path search in dynamic and partially observable environments*, PhD thesis, Middle East Technical University, Çankaya, Ankara, Turkey, 2007.

[86] Trevor Standley, "Finding optimal solutions to cooperative pathfinding problems", in *Proceedings of the 24th National Conference on Artificial Intelligence, AAAI*, 2010, vol. 1, p. 173 – 178.

[87] Glenn Wagner and Howie Choset, "Subdimensional expansion for multirobot path planning", *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.

[88] Dave Ferguson, Maxim Likhachev, and Anthony Stentz, "A guide to heuristic-based path planning", in *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling, ICAPS*, June 2005, pp. 9–18.

[89] Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy, "Incremental heuristic search in artificial intelligence", *Artificial Intelligence Magazine*, vol. 25, no. 2, pp. 99 – 112, 2004.

[90] Sven Koenig and Xiaoxun Sun, "Comparing real-time and incremental heuristic search for real-time situated agents", *Autonomous Agents and Multi-Agent Systems*, vol. 18, pp. 313 – 341, 2009.

[91] Ravikiran A S, "A* algorithm concepts and implementation", 2021, [Online; accessed 7-July-2023].

[92] Thaddeus Abiy, Hannah Pang, Beakal Tiliksew, Karleigh Moore, and Jimin Khim, "A* search", 2016, [Online; accessed 7-July-2023].

[93] Richard E. Korf, "Real-time heuristic search", *Artificial intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.

[94] Xiaoxun Sun, Sven Koenig, and William Yeoh, "Generalized Adaptive A*", in *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2008, vol. 1, p. 462 – 469.

[95] Sven Koenig, Maxim Likhachev, and Xiaoxun Sun, "Speeding up moving-target search", in *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2007, pp. 1–8.

[96] Sandip Aine and Maxim Likhachev, "Truncated incremental search", *Artificial Intelligence*, vol. 234, pp. 49 – 77, 2016.

[97] Carlos Hernández, Jorge Baier, Tansel Uras, and Sven Koenig, "Position paper: Incremental search algorithms considered poorly understood", in

Proceedings of the 5th International Symposium on Combinatorial Search, SoCS, 2012, vol. 3, p. 159 – 161.

[98] Sven Koenig, Maxim Likhachev, and David Furcy, "Lifelong planning A*", *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[99] Sven Koenig and Maxim Likhachev, "Incremental A*", in *Proceedings of the Advances in neural information processing systems 14*, Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, Eds., vol. 2, pp. 1539–1546. MIT Press, Vancouver, British Columbia, Canada, 2001.

[100] Anthony Stentz, "Optimal and efficient path planning for partially-known environments", in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, ICRA*, San Diego, CA, USA, 1994, vol. 4, pp. 3310–3317, IEEE Computer Society.

[101] Anthony Stentz, "The Focussed D* algorithm for real-time replanning", in *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI*, August 1995, vol. 2, pp. 1652–1659.

[102] Sven Koenig and Maxim Likhachev, "Fast replanning for navigation in unknown terrain", *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[103] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset, "Multi-objective path-based D* Lite", *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3318 – 3325, 2022.

[104] Xiaoxun Sun, William Yeoh, and Sven Koenig, "Efficient incremental search for moving target search", in *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI*, 2009, p. 615 – 620.

[105] Xiaoxun Sun, William Yeoh, and Sven Koenig, "Generalized Fringe-Retrieving A*: faster moving target search on state lattices", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2010, vol. 2, pp. 1081–1088.

[106] Luis Henrique Oliveira Rios and Luiz Chaimowicz, "A survey and classification of A* based best-first heuristic search algorithms", in *Brazilian Symposium on Artificial Intelligence*. Springer, 2010, vol. 6404 LNAI, pp. 253–262.

[107] Dave Ferguson and Anthony Stentz, "Field D*: An interpolation-based path planner and replanner", in *Robotics Research. Springer Tracts in Advanced Robotics*, Sebastian Thrun, Rodney Brooks, and Hugh Durrant-Whyte, Eds., Berlin, Heidelberg, 2007, vol. 28, pp. 239–253, Springer Berlin Heidelberg.

[108] Rong Zhou and Eric A. Hansen, "Multiple sequence alignment using Anytime A*", in *Proceedings of the 18th National Conference on Artificial Intelligence, AAAI*, 2002, pp. 975–977.

[109] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun, "Anytime search in dynamic graphs", *Artificial Intelligence*, vol. 172, pp. 1613 – 1643, 2008.

[110] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality", in *Advances in Neural Information Processing Systems 16*, Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, Eds., vol. 16, pp. 767–774. MIT Press, Vancouver and Whistler, British Columbia, Canada, 2003.

[111] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun, "Anytime Dynamic A*: An anytime, replanning algorithm", in *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 2005, vol. 5, pp. 262–271.

[112] Eric A. Hansen and Rong Zhou, "Anytime heuristic search", *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[113] Xiaoxun Sun, Tansel Uras, Sven Koenig, and William Yeoh, "Incremental ARA*: An incremental anytime search algorithm for moving-target search", in *ICAPS 2012 - Proceedings of the 22nd International Conference*

on Automated Planning and Scheduling, Lee McCluskey, Brian Charles Williams, José Reinaldo Silva, and Blai Bonet, Eds., Atibaia, São Paulo, Brazil, 2012, p. 243 – 251, AAAI.

[114] Adi Botea, "Ultra-fast optimal pathfinding without runtime search", in *Proceedings of the 7th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2011, p. 122 – 127.

[115] Adi Botea, "Fast, optimal pathfinding with compressed path databases", in *Proceedings of the 5th International Symposium on Combinatorial Search, SoCS*, 2012, vol. 3, p. 204 – 205.

[116] Adi Botea, Jorge A. Baier, Daniel Harabor, and Carlos Hernández, "Moving target search with compressed path databases", in *ICAPS 2013 - Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2013, p. 288 – 292.

[117] Jorge A. Baier, Adi Botea, Daniel Harabor, and Carlos Hernández, "Fast algorithm for catching a prey quickly in known and partially known game maps", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, pp. 193–199, 2015.

[118] Doron Nussbaum and Alper Yörükçü, "Moving target search with subgoal graphs", in *ICAPS 2015 - Proceedings of the 25th International Conference on Automated Planning and Scheduling*, 2015, vol. 2015-January, pp. 179–187.

[119] Sven Koenig and Maxim Likhachev, "Real-time Adaptive A*", in *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2006, pp. 281–288.

[120] Kengo Matsuta, Hayato Kobayashi, and Ayumi Shinohara, "Multi-target Adaptive A*", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2010, vol. 1, pp. 1065–1072.

[121] Carlos Hernández, Xiaoxun Sun, Sven Koenig, and Pedro Meseguer, "Tree Adaptive A*", in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2011, pp. 123–130.

[122] Carlos Hernández, Pedro Meseguer, Xiaoxun Sun, and Sven Koenig, "Path-adaptive A* for incremental heuristic search in unknown terrain", in *ICAPS 2009 - Proceedings of the 19th International Conference on Automated Planning and Scheduling*, 2009, p. 358 – 361.

[123] Carlos Hernández, Jorge A Baier, Tansel Uras, and Sven Koenig, "Time-bounded Adaptive A*", in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2012, pp. 997–1006.

[124] Carlos Hernández, Roberto Asín, and Jorge A Baier, "Reusing previously found A* paths for fast goal-directed navigation in dynamic terrain", in *Proceedings of the 29th National Conference on Artificial Intelligence, AAAI*, 2015, vol. 2, p. 1158 – 1164.

[125] Carlos Hernández, Jorge A. Baier, and Roberto Asín, "Making A* run faster than D* Lite for path-planning in partially known terrain", in *ICAPS 2014 - Proceedings of the 24th International Conference on Automated Planning and Scheduling*, 2014, vol. 24, pp. 504–508.

[126] Toru Ishida, "Real-time search for autonomous agents and multiagent systems", *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 2, pp. 139–167, 1998.

[127] Makoto Yokoo and Yashiko Kitamura, "Multiagent real-time-A* with selection: Introducing competition in cooperative search", in *Proceedings of the 2nd International Conference on Multiagent Systems, ICMAS-96*, 1996, pp. 409–416.

[128] Vadim Bulitko, Nathan Sturtevant, Jieshan Lu, and Timothy Yau, "Graph abstraction in real-time heuristic search", *Journal of Artificial Intelligence Research*, vol. 30, pp. 51–100, 2007.

[129] Peter K.K. Loh and Edmond C. Prakash, "Performance simulations of moving target search algorithms", *International Journal of Computer Games Technology*, vol. 2009, 2009.

[130] Carlos Hernández and Jorge A. Baier, "Avoiding and escaping depressions in real-time heuristic search", *Journal of Artificial Intelligence Research*, vol. 43, pp. 523–570, 2012.

[131] Toru Ishida and Richard E. Korf, "Moving-target search: A real-time search for changing goals", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 609–619, 1995.

[132] Çağatay Ündeğer and Faruk Polat, "Moving target search in grid worlds", in *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2007, pp. 1–3.

[133] Çağatay Ündeğer and Faruk Polat, "Real-time moving target search", in *Agent Computing and Multi-Agent Systems. Pacific Rim International Conference on Multi-Agents (PRIMA 2007)*, Aditya Ghose, Guido Governatori, and Ramakoti Sadananda, Eds., Berlin, Heidelberg, 2009, vol. 5044, pp. 110–121, Springer Berlin Heidelberg.

[134] Nerea Luis, Susana Fernández, and Daniel Borrajo, "Plan merging by reuse for multi-agent planning", *Applied Intelligence*, vol. 50, pp. 365–396, 2020.

[135] Aniello Murano, Giuseppe Perelli, and Sasha Rubin, "Multi-agent path planning in known dynamic environments", in *18th International conference on principles and practice of multi-agent systems, PRIMA*, 2015, vol. 9387, pp. 218–231.

[136] Carlos Hernández, Tansel Uras, Sven Koenig, Jorge A. Baier, Xiaoxun Sun, and Pedro Meseguer, "Reusing cost-minimal paths for goal-directed navigation in partially known terrains", *Autonomous Agents and Multi-Agent Systems*, vol. 29, pp. 850–895, 2015.

[137] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger, "Verification of agent navigation in partially-known environments", *Artificial Intelligence*, vol. 308, pp. 103724, 2022.

[138] Ariel Felner, Roni Stern, Asaph Ben-Yair, Sarit Kraus, and Nathan Netanyahu, "PHA*: Finding the shortest path with A* in an unknown physical environment", *Journal of Artificial Intelligence Research*, vol. 21, pp. 631–670, 2004.

[139] Christopher Dragert, Jörg Kienzle, and Clark Verbrugge, "Statechart-based AI in practice", in *Proceedings of the 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2012, p. 136 – 141.

[140] Bryan Stout, "Smart moves: Intelligent pathfinding", *Game developer magazine*, vol. 10, pp. 28–35, 1996.

[141] Nathan R. Sturtevant, Devon Sigurdson, Bjorn Taylor, and Tim Gibson, "Pathfinding and abstraction with dynamic terrain costs", in *Proceedings of the 15th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2019, vol. 15, pp. 80–86.

[142] Ronen I. Brafman and Carmel Domshlak, "From one to many: Planning for loosely coupled multi-agent systems", in *ICAPS 2008 - Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 2008, vol. 8, pp. 28–35.

[143] Raz Nissim, Ronen I. Brafman, and Carmel Domshlak, "A general, fully distributed multi-agent planning algorithm", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2010, vol. 2, p. 1323 – 1330.

[144] Jonas Kvarnström, "Planning for loosely coupled agents using partial order forward-chaining", in *ICAPS 2011 - Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011, p. 138 – 145.

[145] Matt Crosby and Michael Rovatsos, "Heuristic multiagent planning with self-interested agents", in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2011, vol. 2, pp. 1213–1214.

[146] Matthew Crosby, Michael Rovatsos, and Ronald P.A. Petrick, "Automated agent decomposition for classical planning", in *ICAPS 2013 - Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2013, vol. 23, pp. 46–54.

[147] Hang Ma, Jiaoyang Li, T. K.Satish Kumar, and Sven Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks", in *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2017, vol. 2, pp. 837–845.

[148] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma, "Multi-goal multi-agent pickup and delivery", in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2022, vol. 2022-October, pp. 9964–9971.

[149] Florian Grenouilleau, Willem-Jan van Hoeve, and John N Hooker, "A multi-label A* algorithm for multi-agent pathfinding", in *ICAPS 2019 - Proceedings of the 29th International Conference on Automated Planning and Scheduling*, 2019, vol. 29, pp. 181–185.

[150] Robert C. Holte, Ariel Felner, Guni Sharon, and Nathan R. Sturtevant, "Bidirectional search that is guaranteed to meet in the middle", in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, vol. 30, p. 3411 – 3417.

[151] Robert C. Holte, Ariel Felner, Guni Sharon, Nathan R. Sturtevant, and Jingwei Chen, "MM: A bidirectional search algorithm that is guaranteed to meet in the middle", *Artificial Intelligence*, vol. 252, pp. 232 – 266, 2017.

[152] Amarjeet Singh, William Kaiser, Maxim Batalin, Andreas Krause, and Carlos Guestrin, "Efficient planning of informative paths for multiple robots", in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, p. 2204 – 2211.

[153] Satyendra Singh Chouhan and Rajdeep Niyogi, "DMAPP: A distributed multi-agent path planning algorithm", in *Proceedings of the 28th Australasian Joint Conference on Artificial Intelligence*, Bernhard

Pfahringer and Jochen Renz, Eds., vol. 9457 of *Lecture Notes in Computer Science*, pp. 123–135. Springer, Canberra, Australia, 2015.

[154] Oren Salzman and Roni Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems blue sky ideas track", in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2020, vol. 2020-May, p. 1711 – 1715.

[155] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig, "Task and path planning for multi-agent pickup and delivery", in *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2019, vol. 2, p. 1152 – 1160.

[156] Pavel Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering", in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021, vol. 35, pp. 12409–12417.

[157] Sumanth Varambally, Jiaoyang Li, and Sven Koenig, "Which MAPF model works best for automated warehousing?", in *Proceedings of the 15th International Symposium on Combinatorial Search, SOCS*, 2022, vol. 15, pp. 190–198.

[158] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig, "Lifelong multi-agent path finding in large-scale warehouses", in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021, vol. 35, pp. 11272–11281.

[159] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant, "Conflict-based search for optimal multi-agent path finding", in *Proceedings of the 26th National Conference on Artificial Intelligence, AAAI*, Ontario, Canada, 2012, vol. 1, p. 563 – 569, AAAI Press.

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

[161] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding", *Artificial Intelligence*, vol. 219, pp. 40–66, 2 2015.

[162] Ko Hsin Cindy Wang and Adi Botea, "Mapp: A scalable multi-agent path planning algorithm with tractability and completeness guarantees", *Journal of Artificial Intelligence Research*, vol. 42, pp. 55–90, 2011.

[163] Hang Ma and Sven Koenig, "Optimal target assignment and path finding for teams of agents", in *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2016, p. 1144 – 1152.

[164] Jingjin Yu and Steven M. Lavalle, "Multi-agent path planning and network flow", in *Algorithmic foundations of robotics X, Springer Tracts in Advanced Robotics*, Emilio Frazzoli, Tomas Lozano-Perez, Nicholas Roy, and Daniela Rus, Eds., Berlin, Heidelberg, 2013, vol. 86, pp. 157–173, Springer Berlin Heidelberg.

[165] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig, "Improved heuristics for multi-agent path finding with conflict-based search", in *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI*, 2019, vol. 2019-August, pp. 442–449.

[166] Xinyi Zhong, Jiaoyang Li, Sven Koenig, and Hang Ma, "Optimal and bounded-suboptimal multi-goal task assignment and path finding", in *Proceedings of the 2022 EEEI International Conference on Robotics and Automation, ICRA*, 2022, pp. 10731–10737.

[167] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig, "Multi-agent path finding for large agents", in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019, vol. 33, pp. 7627–7634.

[168] Wolfgang Honig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian, "Persistent and robust execution of mapf schedules in warehouses", *IEEE Robotics and Automation Letters*, vol. 4, pp. 1125–1131, 2019.

[169] Nir Greshler, Ofir Gordon, Oren Salzman, and Nahum Shimkin, "Cooperative multi-agent path finding: Beyond path planning and collision avoidance", in *2021 International Symposium on Multi-Robot and Multi-Agent Systems, MRS 2021*, 2021, pp. 20–28.

[170] Devon Sigurdson, Vadim Bulitko, William Yeoh, Carlos Hernández, and Sven Koenig, "Multi-agent pathfinding with real-time heuristic search", in *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2018, vol. 2018-August, pp. 1–8.

[171] Vadim Bulitko and Nathan Sturtevant, "State abstraction for real-time moving target pursuit: A pilot study", in *AAAI Workshop: Learning For Search*, 2006, vol. WS-06-11, pp. 72–79.

[172] Edmund K. Burke and Graham Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer, 2 edition, 2014.

[173] Alejandro Isaza, "A heuristic-based approach to multi-agent moving-target search", Master's thesis, University of Alberta, Edmonton, Alberta, Canada, 2008.

[174] William B. Kinnersley, "Cops and robbers is exptime-complete", *Journal of Combinatorial Theory, Series B*, vol. 111, pp. 201–220, 2015.

[175] Alessandro Berarducci and Benedetto Intrigila, "On the cop number of a graph", *Advances in Applied Mathematics*, vol. 14, pp. 389–403, 1993.

[176] Alejandro Torreño, Eva Onaindia, and Óscar Sapena, "FMAP: Distributed cooperative multi-agent planning", *Applied Intelligence*, vol. 41, no. 2, pp. 606–626, 2014.

[177] Akihiro Kishimoto, Alex Fukunaga, and Adi Botea, "Scalable, parallel best-first search for optimal sequential planning", in *ICAPS 2009 - Proceedings of the 19th International Conference on Automated Planning and Scheduling*, 2009, vol. 19, pp. 201–208.

[178] Nathan R. Sturtevant, "Benchmarks for grid-based pathfinding", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[179] Hang Ma, Craig Tovey, Guni Sharon, T K Kumar, and Sven Koenig, "Multi-agent path finding with payload transfers and the package-exchange robot-routing problem", in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, vol. 30, p. 3166 – 3173.

[180] Matteo Bellusci, Nicola Basilico, and Francesco Amigoni, "Multi-agent path finding in configurable environments", in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2020, vol. 2020-May, pp. 159–167.

[181] Jack Edmonds and Richard M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the ACM (JACM)*, vol. 19, pp. 248–264, 1972.

[182] John E. Hopcroft and Richard M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM Journal on Computing*, vol. 2, pp. 225–231, 1973.

[183] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Barták, "Multi-agent pathfinding: Definitions, variants, and benchmarks", in *Proceedings of the 12th International Symposium on Combinatorial Search, SoCS*, 2019, p. 151 – 158.

[184] Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau, "Pathfinding in games", in *Artificial and Computational Intelligence in Games*, Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, Eds., vol. 6, pp. 21–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[185] Amit Patel, "Heuristics", 1999, [Online; accessed 14-July-2023].

[186] Nathan Sturtevant and Michael Buro, "Improving collaborative pathfinding using map abstraction", in *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*, 2006, pp. 80–85.

[187] Christopher Wilt and Adi Botea, "Spatially distributed multiagent path planning", in *ICAPS 2014 - Proceedings of the 24th International Conference on Automated Planning and Scheduling*, 2014, vol. 24, pp. 332–340.

[188] Jirí Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták, "Online multi-agent pathfinding", in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019, vol. 33, pp. 7732–7739.

[189] Xiaoming Zheng and Sven Koenig, "K-swaps: Cooperative negotiation for solving task-allocation problems", in *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI*, 2009, p. 373 – 378.

[190] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig, "Anytime multi-agent path finding via large neighborhood search", in *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI*, 2021, pp. 4127–4135.

[191] Fatih Semiz and Faruk Polat, "Incremental multi-agent path finding", *Future Generation Computer Systems*, vol. 116, pp. 220–233, 2021.

[192] Yinbin Shi, Biao Hu, and Ran Huang, "Task allocation and path planning of many robots with motion uncertainty in a warehouse environment", in *2021 IEEE International Conference on Real-Time Computing and Robotics, RCAR*, 2021, p. 776 – 781.

[193] Ben Strasser, Daniel Harabor, and Adi Botea, "Fast first-move queries through run-length encoding", in *Proceedings of the International Symposium on Combinatorial Search*, 2014, vol. 5, pp. 157–165.

[194] Carsten Maple, Edmond Prakash, Wei Huang, and Adnan N. Qureshi, "Taxonomy of optimisation techniques and applications", *International Journal of Computer Applications in Technology*, vol. 49, no. 3-4, pp. 251–262, 2014.

[195] Ko-Hsin Cindy Wang, "Massively multi-agent pathfinding made tractable, efficient, and with completeness guarantees", in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2011, vol. 2, pp. 1343–1344.

[196] Alborz Geramifard, Pirooz Chubak, and Vadim Bulitko, "Biased cost pathfinding", in *Proceedings of the 2nd AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2006, vol. 2, pp. 112–114.

[197] Ko Hsin Cindy Wang and Adi Botea, "Tractable multi-agent path planning on grid maps", in *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI*, 2009, vol. 9, p. 1870 – 1875.

[198] Richard M. Karp, "Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane", *Mathematics of operations research*, vol. 2, pp. 209–224, 1977.

[199] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni, "Intractability of time-optimal multirobot path planning on 2D grid graphs with holes", *IEEE Robotics and Automation Letters*, vol. 2, pp. 1941–1947, 2017.

[200] Mathijs De Weerdt, Adriaan Ter Mors, and Cees Witteveen, "Multi-agent planning: An introduction to planning and coordination", Tech. Rep., In: Handouts of the European Agent Summer, 2005.

[201] Ard C M Al and Mark Hoogendoorn, "Moving target search using theory of mind", in *Proceedings - 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT*. IEEE, 2011, vol. 2, pp. 66–71.

[202] Tng C. H. John, Edmond C. Prakash, and Narendra S. Chaudhari, "Strategic team AI path plans: Probabilistic pathfinding", *International Journal of Computer Games Technology*, vol. 2008, pp. 1–6, 2008.

[203] Wikipedia contributors, "Mann–whitney u test — wikipedia, the free encyclopedia", 2022, [Online; accessed 23-November-2022].

[204] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Afnizanfaizal Abdullah, "A new weighted pathfinding algorithms to reduce the search time on grid maps", *Expert Systems with Applications*, vol. 71, pp. 319–331, 4 2017.

[205] Wikipedia contributors, "Breadth-first search — wikipedia, the free encyclopedia", 2022, [Online; accessed 23-November-2022].

[206] Wikipedia contributors, "Dijkstra's algorithm — wikipedia, the free encyclopedia", 2022, [Online; accessed 23-November-2022].

[207] Carsten Moldenhauer, "Game tree search algorithms for the game of cops and robber", Master's thesis, University of Alberta, Edmonton, Alberta, Canada, 2010.

[208] Rafa M. Kasim, "Interaction effect", in *Encyclopedia of Survey Research Methods*, Paul J. Lavrakas, Ed., pp. 340–342. Sage Publications, Thousand Oaks, 2008, [Online; accessed 23-November-2022].

[209] Michal Čáp, Peter Novák, Jiří Vokřínek, and Michal Pěchouček, "Multi-agent RRT*: Sampling-based cooperative pathfinding", *arXiv preprint arXiv:1302.2828*, 2013.

[210] Stanley Melax, "New approaches to moving target search.", in *Proceedings of the AAAI Fall Symposium, Games: Planning and Learning*, 1993, pp. 30–38.

[211] Wikipedia contributors, "Friedman test — wikipedia, the free encyclopedia", 2022, [Online; accessed 23-November-2022].

[212] Janez Demšar, "Statistical comparisons of classifiers over multiple data sets", *The Journal of Machine learning research*, vol. 7, pp. 1–30, 2006.

[213] yeswecamp, "Perfect heist 2", Microsoft Windows, 2021.

[214] Rachid Alami, Frédéric Robert, Félix Ingrad, and Sho'ji Suzuki, "Multi-robot cooperation through incremental plan-merging", in *Proceedings of 1995 IEEE International Conference on Robotics and Automation, ICRA*, 1995, vol. 3, p. 2573 – 2579.

[215] Eli Boyarski, Shao-Hung Chan, Dor Atzmon, Ariel Felner, and Sven Koenig, "On merging agents in multi-agent pathfinding algorithms", in *Proceedings of the 15th International Symposium on Combinatorial Search, SoCS*, 2022, vol. 15, pp. 11–19.

[216] Adi Botea, Martin Müller, and Jonathan Schaeffer, "Near optimal hierarchical path-finding", *Journal of game development*, vol. 1, pp. 1–30, 2004.

[217] Daniel Harabor and Adi Botea, "Breaking path symmetries on 4-connected grid maps", in *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE*, 2010, p. 33 – 38.

[218] Daniel Harabor and Alban Grastien, "Online graph pruning for pathfinding on grid maps", in *Proceedings of the 25th National Conference on Artificial Intelligence, AAAI*, 2011, vol. 2, p. 1114 – 1119.

[219] Yue Hu, Daniel Harabor, Long Qin, and Quanjun Yin, "Regarding goal bounding and jump point search", *Journal of Artificial Intelligence Research*, vol. 70, pp. 631 – 681, 2021.