THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Advances in Multi-Agent Path Finding

Scalable lifelong planning and optimal continuous-time solutions.

Alvin Combrink

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Advances in Multi-Agent Path Finding

Scalable lifelong planning and optimal continuous-time solutions.

Alvin Combrink

Acknowledgements, dedications, and similar personal statements in this thesis, reflect the author's own views.

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Cover:
Safe Interval Path Planning (SIPP) [1] underlies many continuous-time multi-agent path finding algorithms. The cover figure illustrates an environment with four agents, three of which have a planned trajectory. SIPP is applied to the center, unplanned, agent to obtain the earliest time for it to arrive and remain indefinitely at every other vertex in the environment, with each vertex's color representing that time. Read more about SIPP on page 53.

Printed by Chalmers Digital Printing
Gothenburg, Sweden, April 2025

Advances in Multi-Agent Path Finding

Scalable lifelong planning and optimal continuous-time solutions.

Alvin Combrink
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Multi-Agent Path Finding (MAPF) addresses the collision-free coordination of multiple agents moving through a shared environment, finding applications in warehouses, airports, and video games, to name a few. Classical MAPF assumes discrete time and agents with pre-assigned goals, yet real-world scenarios demand continuous-time operations with dynamically arriving tasks. This thesis explores the techniques employed by scalable discrete-time MAPF and Lifelong MAPF (LMAPF) solvers; the challenges that arise from extending LMAPF to continuous time (LMAPF_R); and how MAPF in continuous-time (MAPF_R) can be solved for optimal solutions.

We find that many scalable MAPF methods rely on prioritized planning, windowed planning, and dimensional simplifications. These insights culminate in a method that solves LMAPF for hundreds of agents with competitive throughput. Furthermore, addressing LMAPF_R presents challenges related to asynchronous movements, a dense search space, and volumetric agents complicating real-time collision detection and resolution. Our proposed algorithm is, to our knowledge, the first to address LMAPF_R, capable of planning up to a thousand agents in real time. Finally, *Continuous-time Conflict-Based Search* (CCBS) was thought to be the sole MAPF_R method for optimal solutions. Recent work suggests otherwise. We explain why CCBS *could*, and provide experimental evidence that CCBS *does*, return sub-optimal solutions. Consequently, we present a new MAPF_R solver — to our knowledge, the first to provably return optimal solutions in finite time.

This work advances both the practical applicability and theoretical foundations of MAPF, bridging the gap between discrete-time abstractions and continuous-time realities.

Keywords: Multi-agent Path Finding, Continuous-time, Lifelong MAPF, Optimal Planning, Conflict-Based Search

To my dad.

List of Publications

This thesis is based on the following publications:

[A] Francesco Popolizio, Martina Vinetti, **Alvin Combrink**, Sabino Francesco Roselli, Maria Pia Fanti, Martin Fabian, "Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots". *20th International Conference on Automation Science and Engineering (CASE)*, IEEE, Bari, Italy, Aug. 2024.

[B] **Alvin Combrink**, Sabino Francesco Roselli, Martin Fabian, "Prioritized Planning for Continuous-time Lifelong Multi-agent Pathfinding". *11^{th} International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, Split, Croatia, Jul. 2025.

[C] **Alvin Combrink**, Sabino Francesco Roselli, Martin Fabian, "Optimal Multi-agent Path Finding in Continuous Time". Under review in *Artificial Intelligence*, Elsevier, 2025.

Other publications by the author, not included in this thesis, are:

[D] **Alvin Combrink**, David Johnson, Petr Moldan, Martin Fabian, "Discrete-Event Based Patient Flow Simulation of an Emergency Surgery Department". *IEEE 10^{th} International Conference on Control, Decision and Information Technologies (CoDIT)*, Valletta, Malta, Jul. 2024.

[E] **Alvin Combrink**, Stephie Do, Kristofer Bengtsson, Sabino Francesco Roselli, Martin Fabian, "A Comparative Study of SMT and MILP for the Nurse Rostering Problem". *IEEE 11^{th} International Conference on Control, Decision and Information Technologies (CoDIT)*, Split, Croatia, Jul. 2025.

Contents

Acknowledgments

This journey is only partly done, yet there are already so many people to thank. Starting with my supervisors *Martin*, *Sabino*, and *Kristofer*, thank you for the many valuable lessons you have taught me, and for always challenging my reasoning, biases, writing, and communication. *Bengt*, *Bertil*, and *Petter*, thank you for the years of teaching in your courses and trusting me with the freedom to explore different ways to do so. Thank you *Christine* for keeping the wheels turning, without you, the office would descend into chaos.

Thank you to all of my friends, I get to live in a colorful world because of you: *Hampus*, for our time deep in the sea and high in the sky; *Tobias*, for the many years of friendship; *Rikard*, *Julius*, *Ludvig*, and *Anton*, for the good times and fun discussions; my office mates *Adam* and *Viktor*, without whom my days would be spent in boring silence; *Rita*, *Gabriel*, *Alejandra*, *Christian*, *Nishant*, *Nanami*, *Yara*, *André*, *Ying*, *Ola*, *Mattia*, *Laura*, *Boss*, *Mimmi*, *Kilian*, *Sherry*, *Lorenzo*, *Ben*, *Zhitao*, *Erik Börve*, *Ze*, *Endre*, *Rémi*, and *Moein*, may we create many more memories together; and all of my friends in and outside the halls of Chalmers: *Constantin*, *Lasse*, *Kristian*, *Martina*, *Francesco*, *Erik Brorsson*, *Chiara*, *Amal*, *Albert*, *Godwin*, *Huang*, *Sten-Elling*, *Yao*, *Maxi*, *Xiaolei*, *Sondre*, *Carl-Johan*, *Johannes*, *Wenhao*, *Faris*, *Filip*, *Isac*, *Ahmet*, *Daniel*, and *Attila*. If you are reading this but don't see your name, you likely belong here. Please forgive my poor memory.

My love and gratitude goes to my family: my mom *Selly* and step-dad *Ken*, my sister *Adabelle* and her beautiful family with *Erik*, *Milo*, *Enzo* (and the cats *Jinx* and *Ezio*), my sister from another mister *Karolin* — and *Stephie*, who has been my partner and best friend at every step of the way. Finally, to my dad, *Schutte*, thank you for sharing with me your curiosity in the workings of the universe and encouraging me to explore all that is fascinating. You are dearly missed.

Acronyms

MAPF:	Multi-agent Path Finding
LMAPF:	Lifelong Multi-Agent Path Finding
MAPF_R:	Multi-agent Path Finding in Continuous Time
LMAPF_R:	Lifelong Multi-Agent Path Finding in Continous Time
SIPP:	Safe Interval Path Planning
CBS:	Conflict-Based Search
CCBS:	Continuous-time Conflict-Based Search
SOC:	Sum-Of-Costs
CPLP:	Continous-time Prioritized Lifelong Planner
AMR:	Autonomous Mobile Robot
MAPD:	Multi-Agent Pickup and Delivery

Part I

Overview

CHAPTER 1

Introduction

Everyday, people weave through streets, bustling offices, and busy supermarkets, moving goods, meeting friends, or simply making their way home. Rarely do we notice the complexity of the dances that we unknowingly and intuitively perform with strangers without saying a word; somehow, we unconsciously plan and adapt our path through a busy crowd using our sense of agency and innate ability to avoid collisions with others — we are all collectively solving the problem of *Multi-Agent Path Finding* (MAPF).

This innate ability to solve MAPF problems does not come as naturally to computers, which live in the world of decided rationality. That is not to say that computers are at a disadvantage; they do not suffer from bias and unpredictability, unless we humans instill such tendencies upon them through our algorithms. Computers follow the instructions that we as algorithm designers lay out for them, and if we are careful enough, they might just solve problems better and faster than what we ever could.

When mentioning the problem of MAPF, a reasonable first thought may be of factories and warehouses with scores of logistical robots performing tasks and carrying goods. This is indeed an increasingly common sight in today's industry [1–4]. However, MAPF finds application in diverse fields,

such as video games [5, 6], traffic management systems [7, 8], airport surface management [9, 10], search and rescue missions [11], collaborative robots in office spaces [12, 13], and many more. These domains are united by having multiple agents moving in a shared space where collisions must be avoided.

The two common goals among all MAPF applications are, with various degrees of importance, to find *good* solutions *fast*. In fact, this is the general goal within all forms of optimization. We certainly do not want to find bad solutions, especially not if it takes a long time, and even worse if we never obtain a solution at all. Many of the current algorithms to finding MAPF solutions balance these two goals in various ways. Algorithms for real-time use tend to search for good enough solutions in as short a time as possible, or use all available time and return the thus-far best found solution. Then there exists another class of algorithms that search for the *best* solution and hope to succeed within a reasonable time. What constitutes a solution and what makes it good or bad depends on the type of MAPF problem that we are talking about. In most MAPF problems, however, a solution specifies how each agent should move to eventually arrive at its goal without collision, and a good solution gets the agents there sooner rather than later.

MAPF problems come in many flavors. In a *one-shot* [14] MAPF problem, a fixed number of agents each begin at some start location and are tasked with moving to some goal location. This is referred to as "one-shot" because all information that is required to solve the problem is given at the start, and a solution can be computed in a single attempt. *Classical MAPF* [15], which "MAPF" often refers to, is a one-shot problem with discretized time. Time being discrete means that at every timestep, each agent performs an action to either move somewhere or wait where it is. The aim is then to decide how each agent should act to eventually reach its goal while avoiding collisions with other agents. The *Continuous-time MAPF* problem (MAPF_R, "R" indicating that real values in $\mathbb{R}$ are used) [16] is similar to classical MAPF, however, with the one small difference that agents can perform actions at *any* time instead of at discrete timesteps. As we will see in later chapters, this small change comes with several fundamental challenges.

The alternative to one-shot problems — such as classical MAPF and MAPF_R introduced above — are *lifelong* problems, which evolve with new information over time such that solving the problem once is not enough. The *Lifelong MAPF* (LMAPF) problem [17] shares similarities with the classical MAPF

problem in that time is discrete and each agent starts at a unique location. However, in LMAPF, agents are not initially assigned a goal location. Instead, an endless stream of unassigned goals are revealed over time. The planner then assigns revealed goals to agents and plans their movements to those goals. This lifelong formulation better suits most industrial settings where transport orders are placed over time, such as in a warehouse where purchase orders require products to be fetched from shelves and moved to outbound trucks. Finally, the combined problem of time being continuous and tasks endlessly entering the system is referred to as the *Continuous-time Lifelong MAPF* (LMAPF_R) problem, defined in Paper B.

Almost all MAPF literature assumes that agents live and move on *graphs*, such as the graph shown on the cover of this thesis. The breadth of structures that a humble graph can represent is astonishing: roads and railways [18], social networks [19], linguistics [20], financial systems [21], knowledge [22], to mention just a few. Google's original success relied on PageRank [23], a method that represents the internet as a graph to rank websites. Many high-profile scientific breakthroughs in recent years — such as AlphaFold [24] for accurate protein structure prediction and GraphCast [25] for weather prediction — have stemmed from the realization that machine learning can be applied to graphs [26]. Even transformers [27], the underlying architecture behind large language models, can be seen as fundamentally graph-based [28]. In MAPF, graphs represent the environment and come in various levels of complexity, depending on the problem variant.

MAPF problems are generalizations of the *shortest path* problem, which involves finding the shortest path between a start and a goal in a graph. This problem was solved by Edsger W. Dijkstra in 1956 with what is now commonly known as *Dijkstra's algorithm* [29], which was later extended to the also well-known A* algorithm [30]. One could apply Dijkstra's algorithm or A* to each agent individually to find a path from their start to goal, and hope that no collisions occur. However, the chance of that working is exceptionally small for most practical problems. Thus, smarter methods are needed.

The field of MAPF can be compartmentalized in several ways, with each MAPF variant warranting distinct solution approaches. We have above introduced classical MAPF, LMAPF, MAPF_R, and LMAPF_R. However, many other variants exist that we briefly introduce throughout the subsequent chapters. As we will see, MAPF belongs to the class of problems that are thought

to be more difficult to solve than most other problems. This means that, as the problem size grows, optimality and practical usefulness quickly form a dichotomy; both cannot be obtained simultaneously. Eventually, finding the optimal solution will take more time than what is available. Thus, this field can also be divided into methods for optimal solutions that scale poorly, and fast methods that return sub-optimal solutions. Our main focus in this work is on practical approaches to finding sub-optimal solutions to the LMAPF and LMAPF_R problems, and theoretical results for guaranteeing to find optimal solutions to the MAPF_R problem within finite time.

This thesis invites you, the reader, on a journey through the fascinating world of Multi-Agent Path Finding — a field where mathematics and algorithmic design meet the everyday choreography of coordinated movement. Together, we shall stand on the shoulders of giants and enjoy the view of decades of foundational research, examine how the work presented here nudges us closer to practical solutions and theoretical guarantees, and peer into the future to imagine what new paths we might find.

1.1 Research Questions

The main topics of this thesis are centered around three research questions which are formulated based on identified gaps in the existing MAPF literature. Table 1.1 provides an overview of the coupling between each research question, the chapter and paper where it is addressed, and the corresponding MAPF variant.

The first research question explores how existing LMAPF algorithms successfully handle large fleets of agents under real-time constraints, despite the underlying MAPF problem's complexity scaling exponentially with the number of agents [31]. Specifically, we aim to understand what techniques these methods use to reduce problem complexity while finding acceptable solutions. The first research question that we aim to address is:

> **RQ1**: *How can Lifelong MAPF algorithms achieve real-time performance for large-scale industrial fleet management?*

To answer this question, we explore the existing MAPF and LMAPF literature in Chapter 3 to identify common strategies used for achieving scalability.

Table 1.1: The coupling between each research question, the chapter and paper where it is addressed, and the corresponding MAPF variant.

	One-shot MAPF	Lifelong MAPF
Discrete Time		**RQ1** Chapter 3, Paper A
Continuous Time	**RQ3** Chapter 5, Paper C	**RQ2** Chapter 4, Paper B

Then, in Paper A, these strategies are applied in a LMAPF solver to verify their effectiveness.

The next topic explores the extension of LMAPF to the continuous-time domain. Recently, there has been an increasing amount of attention placed on continuous-time MAPF and its related aspects, as discrete time brings with it several practically limiting assumptions. However, to the best of our knowledge, no methods in the existing literature address the LMAPF_{R} problem. Therefore, we seek to identify what key problems must be solved when extending discrete-time approaches to continuous-time lifelong scenarios, and explore how techniques from established solution methods can be effectively utilized or modified for this domain. A critical consideration in this extension is that lifelong algorithms typically operate under strict real-time constraints, meaning that any approach to the LMAPF_{R} problem must maintain computational tractability while handling the additional complexity of continuous-time dynamics. Our second research question therefore addresses this challenge:

RQ2: *What is required to extend discrete-time Lifelong MAPF to continuous time while maintaining computational tractability?*

To find an answer to **RQ2**, we form insights in Chapter 4 about the challenges with addressing the LMAPF_{R} problem through comparison with discrete-time LMAPF. These challenges are identified in the context of existing LMAPF literature, containing potential solution strategies. Paper B then introduces an algorithm that addresses these challenges.

The third and final research question that we aim to address relates to

the MAPF_R problem. Based on our literature review, there exists only one method for solving the MAPF_R problem for optimal solutions: *Continuous-time Conflict-Based Search* (CCBS) [16]. However, recent insights [32] have raised important questions regarding CCBS, suggesting potential concerns about finite-time convergence guarantees in its underlying theory and optimality guarantees in its publicly available implementation. These observations highlight a critical gap in the current state of MAPF_R research: the need for methods with clear theoretical foundations that can provably deliver optimal solutions within finite time. This leads us to our third and final question:

> **RQ3**: *How can the continuous-time MAPF problem be solved in finite time with guaranteed solution optimality?*

To answer this question, we take a theoretical approach to identifying the underlying mechanisms behind the existing MAPF_R solution method's problems. Learning from these insights, Paper C proposes a new method with the desired guarantees.

1.2 Contributions

Toward answering research question **RQ1** — *How can Lifelong MAPF algorithms achieve real-time performance for large-scale industrial fleet management?* — Chapter 3 provides a systematic analysis of existing MAPF and LMAPF literature. Based on this analysis, we identify three common strategies for enabling real-time performance in large-scale applications: prioritized planning, windowed planning, and dimensional simplifications. Paper A proposes the *Fleet Manager* that applies two of these strategies — prioritized planning and dimensional simplifications — to achieve computation times in the order of milliseconds for hundreds of agents while outperforming a widely adopted existing method in terms of solution quality.

To answer research question **RQ2** — *What is required to extend discrete-time Lifelong MAPF to continuous time while maintaining computational tractability?* — Chapter 4 identifies the key technical challenges that arise when transitioning from discrete-time to continuous-time Lifelong MAPF. These are:

- *Asynchronized agent movements*, impeding the use of common discrete-

time strategies that rely on natural synchronization points when all agents can execute their next action.

- *Dense configuration spaces* as a result of time being continuous, requiring more sophisticated strategies for navigating the assignment space.

- *Resolving collisions between volumetric agents.* Time being continuous requires agents to occupy space (unlike in discrete-time where agents are typically represented as sizeless points). Consequently, the range of ways for collisions to occur is greatly expanded, requiring solution algorithms to handle collision resolution in a more generalized way.

- *Real-time collision detection between volumetric agents.* Since agents occupy space, the task of detecting collisions becomes much more computationally demanding. Therefore, methods must address this if the algorithm is to be used in real-time.

Paper B addresses these challenges with the *Continuous-Time Prioritized Lifelong Planner* (CPLP). *Safe Interval Path Planning* [33] is used to navigate the dense assignment space, asynchronous windowed planning addresses the asynchrony between agent movements, CCBS handles collision resolution, and preprocessing strategies allow for real-time collision detection. Through these strategies, CPLP addresses the identified challenges, demonstrating that computational tractability can be maintained for up to a thousand volumetric agents on large graphs with non-unit edge traversal times.

Finally, the contributions in Paper C address research question **RQ3** — *How can the continuous-time MAPF problem be solved in finite time with guaranteed solution optimality?* CCBS is to the best of our knowledge the only method aimed toward finding optimal solutions to the MAPF_R problem. Paper C introduces an analytical framework that provides sufficient criteria for a CCBS-styled algorithm to guarantee the return of an optimal solution in finite time. We find that CCBS does not satisfy these criteria, which merely suggests that CCBS *could* return sub-optimal solutions. However, we provide experimental evidence to show that CCBS *does* return sub-optimal solutions. Paper C then introduces a new *branching rule* for CCBS and proves that it satisfies the framework's criteria. Thus, we provide to our knowledge the first algorithm that provably guarantees to find an optimal solution to a MAPF_R problem in finite time.

1.3 Methodology

We use complementary methodological approaches tailored to the distinct nature of each research question, reflecting the progression from applied algorithm engineering to fundamental theoretical analysis.

For answering research questions **RQ1** and **RQ2**, we use a largely empirical-constructive approach. Our methodology follows three general phases, applied iteratively in practice. First, existing literature is analyzed to identify effective techniques with empirically proven efficiency at addressing the challenges posed by the research question — real-time fleet management for **RQ1** and continuous-time extension challenges for **RQ2**. Second, we apply these insights through iterative algorithm development. Finally, our methods are experimentally evaluated on large benchmark sets, systematically varying problem parameters to assess both effectiveness and scalability.

For **RQ3**, a fundamentally different theoretical-analytical approach is used, centered on mathematical analysis and formal proof construction. Beginning with the identified shortcomings in existing Continuous-time MAPF methods, we develop a mathematical framework to explain these limitations and formulate conditions for algorithm soundness and solution completeness. Through analytical exploration, we construct proofs that establish sufficient conditions for these properties, then synthesize novel algorithms that satisfy these conditions.

This form of logical exploration is — fittingly — comparable to path finding. From a starting point, heuristically and logically guided steps (assumptions and implications) can be taken, some leading to dead-ends, others in unproductive directions, and a select few toward our goal. One of the many explored sequences of logical steps eventually forms a proof, with several proofs establishing the necessary conditions. The theoretical proofs we present are merely one valid logical path through the space of possible demonstrations. Like optimal paths in MAPF, there likely exist alternative proof strategies that may be more direct or intuitive. Our methodology prioritized establishing rigorous results within available time constraints, while acknowledging that the rich landscape of mathematical proof offers multiple valid routes to the same theoretical destination.

1.4 Thesis Outline

This thesis is divided into the two main parts: Part I: Overview provides the necessary background to understanding the field of MAPF and the contribution of this work; Part II: Papers contains the papers on which this thesis builds.

Part I consists of seven chapters. Chapter 1 (this chapter) introduces the thesis as a whole, the research questions that it aims to address, the contributions it makes to the field of MAPF, and the methodological approach taken to do so. Chapter 2 then introduces general concepts surrounding algorithms and how they are measured. Chapters 3, 4, and 5 respectively address MAPF and LMAPF to answer **RQ1**, LMAPF_{R} to answer **RQ2**, and MAPF_{R} to answer **RQ3**. Finally, Chapter 6 provides a short summary of each paper included in Part II, and Chapter 7 draws conclusions, reflects on this work's place in the larger MAPF community, and formulates directions for future work.

CHAPTER 2

What are Problems, Solutions, and Algorithms?

Problem solving is fundamentally about finding solutions to problems; algorithms are the tools that we use to do so. In this chapter, we introduce the fields of problem solving and complexity analysis and thereby lay the ground work for subsequent chapters.

Fundamentals

A computational *problem* defines an input/output relation, where the input is an *instance* and the output is a *solution* [34]. This relationship is described by *constraints* which are created from the problem and instance, and apply to the solution. If a *candidate solution* satisfies the constraints, then it is *feasible* and therefore a solution. If it does not satisfy the constraints, then it is *infeasible* and not a solution. For example, the *sorting problem* is about organizing a set of numbers in, say, ascending order. The sorting problem and the instance $i = \{2, 5, 1, 7, 8, 3\}$ together define the constraints: a solution must be a sequence containing the numbers in i in ascending order. The candidate solution $\langle 1, 2, 3, 5, 7, 8 \rangle$ satisfies these constraints and is therefore a solution, while $\langle 8, 1, 7, 10 \rangle$ does not since it does not contain all numbers in

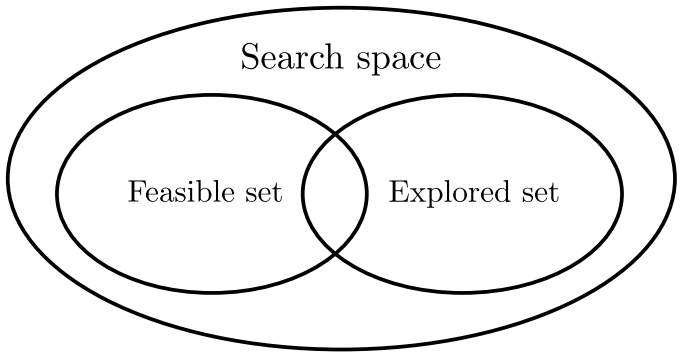


Figure 2.1: The search space, feasible set, and explored set. The explored set must overlap with the feasible set for an algorithm to find a solution.

i and is not in ascending order. We refer to the set of all possible candidate solutions as the *search space*, of which the *feasible set* comprises all solutions.

An *algorithm* is a well-defined computational procedure for finding a solution to any problem instance [34]. To locate solutions, algorithms explore what we refer to as the *explored set* — the subset of the search space containing all candidate solutions that the algorithm actually considers during execution. Figure 2.1 visualizes the relationship between the search space, the feasible set, and the explored set. Algorithms that explore the entire search space without a particular strategy are called *brute-force* algorithms. Unfortunately, many interesting problems suffer from the *curse of dimensionality* [35], making the search space far too large for a brute force algorithm to find a solution fast enough. On the other hand, the explored set of a brute force algorithm usually contains all candidate solutions, including those in the feasible set, consequently guaranteeing that a solution will eventually be found. The goal of more sophisticated algorithms is, however, to reduce the number of infeasible candidate solutions in the explored set and thereby minimize unnecessary exploration. To find *any* solution, it suffices to ensure that at least some portion of the feasible set overlaps with the explored set; to find *all* solutions, the feasible set must be a subset of the explored set.

The field of problem solving consists of, among others, *decision problems* and *optimization problems*. Decision problems can be answered with a simple "yes" or "no", such as whether a number is prime or a path is Hamiltonian.

The sorting problem from above is a type of decision problem where we ask "does there exists a sequence of numbers satisfying all constraints". By providing such a sequence, the answer is "yes". Optimization problems, on the other hand, are about finding solutions that satisfy all constraints and have sufficient quality. Thus, a notion of solution quality is needed, typically defined by an *objective function* that takes a candidate solution as input and returns some quantitative measure of quality. The search space of an optimization problem is sometimes defined as the domain of the objective function [36]. Many optimization problems are about finding the *optimal solution*, that is the solution optimizing (often minimizing) the objective function.

Two desirable properties of an algorithm are [37]:

- **Soundness**: when a sound algorithm for finding a solution to a problem returns a candidate solution, we can trust that the candidate solution is indeed a solution.
- **Completeness**: a complete algorithm for finding a solution to a problem is certain to find a solution if one exists, and to report the non-existence of a solution if none exists.

Note that an algorithm for finding a solution to a problem cannot be complete unless it is also sound; if an algorithm sometimes returns a non-solution, then it does not always find a solution (if one exists) or report the non-existence of a solution (when none exists). Incomplete and unsound algorithms may still be useful, especially if they can find solutions to instances commonly found in practice. Nonetheless, soundness and completeness guarantees that an algorithm will work as intended for any instance. The formal definitions of soundness and completeness are typically only formulated for decision problems. However, in the MAPF community, which deals with optimization, these terms generally have the same meaning: a sound algorithm returns only solutions and a complete algorithm finds a solution if one exists.

The terminology within the field of computer science can be seen as inconsistent and somewhat confusing. For some, an optimal algorithm returns only optimal solutions. For others, an optimal algorithm finds a solution with minimal resource usage (runtime and memory). Established computer science textbooks, such as [34] and [37], do not provide clear and decisive definitions on these terms. However, [34] describes an *optimal binary search tree* as "*... a binary search tree whose expected search cost is smallest.*", a property that [37]

refers to as optimal "*time complexity*", suggesting that both textbooks adopt the latter definition. In this thesis, we adopt the following definitions which we find to be somewhat consistent within the broader computer science context:

- **Exact**: an exact optimization algorithm returns only optimal solutions.
- **Approximate** or *Heuristic*: an approximate (or heuristic) optimization algorithm does not guarantee to return an optimal solution.
- **Optimal**: an optimal algorithm has the lowest possible expected runtime or memory usage.
- **Sub-optimal**: a sub-optimal algorithm does not have the lowest possible expected runtime or memory usage.

Although the MAPF literature may not always be consistent with these terms (e.g., [16] uses "optimal" to mean exact, [14] uses "suboptimal" to mean approximate), the meaning is often clear from context.

Complexity

Algorithms require computational resources in time and memory to execute [34]. These resources are finite and therefore limit what types of algorithms can be run in practice. Thus, the field of complexity analysis is devoted to quantifying the amount of resources required to solve problems [37].

Memory is relatively inexpensive and easily scalable in modern computing environments. Therefore, we mainly care about an algorithm's runtime. Typically, the time it takes to solve a problem grows with the problem's size [34]. This is intuitive; a large crossword puzzle takes longer to solve than a small one, assuming everything else is the same. Complexity theory, however, is less concerned with the absolute runtime and more concerned with the asymptotic behavior of runtime as the problem size grows. This approach abstracts away hardware-specific factors and implementation details, focusing instead on how the fundamental computational requirements scale with input size. Considering the N^2 and 2^N curves in Figure 2.2 provides motivation for this; although 2^N is smaller than N^2 for N between 2 and 4, 2^N grows far beyond N^2 as $N \to \infty$.

An algorithm's worst-case complexity is denoted using the $\mathcal{O}$-notation. Given an upper-bound on an algorithm's worst-case asymptotic runtime as a

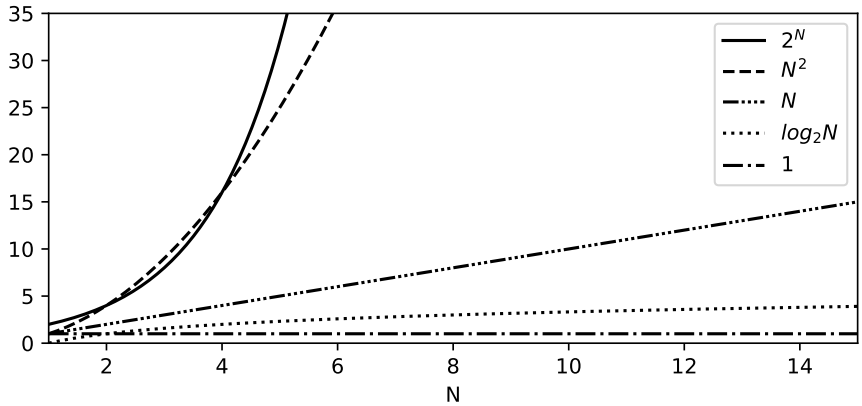


Figure 2.2: A visualization of the grow-rates of the complexity classes *constant* (1), *logarithmic* ($\log_2 N$), *linear* (N), *polynomial* (N^2), and *exponential* (2^N).

function of the problem size N, the $\mathcal{O}$-notation denotes the function's highest-order term [34]. For example, an algorithm with the worst-case runtime upper-bounded by a polynomial of order k with respect to N has complexity $\mathcal{O}(N^k)$, regardless of any lower-order terms. We then say that this algorithm *runs in polynomial time*, or that it is a *polynomial-time algorithm*. Instead, if the highest-order term is an exponential k^N, then the algorithm has complexity $\mathcal{O}(k^N)$. Complexity is often categorized as *constant* $\mathcal{O}(1)$, *logarithmic* $\mathcal{O}(\log_k N)$, *linear* $\mathcal{O}(N)$, *polynomial* $\mathcal{O}(N^k)$, or *exponential* $\mathcal{O}(k^N)$. Figure 2.2 visualizes the grow-rate of these complexity classes, showing their asymptotic behavior as N grows.

A problem's complexity denotes the amount of resources required to solve it. It is relatively easy to find an upper-bound on a problem's complexity; find an algorithm for solving the problem, then the problem's complexity is at most the complexity of the algorithm. On the other hand, it is much harder to find a problem's lower-bound complexity as that would require showing that no solution algorithm with a lower complexity than the lower-bound can exist. We say that a decision problem is

- **polynomial-time solvable** if it can be solved in polynomial time, and

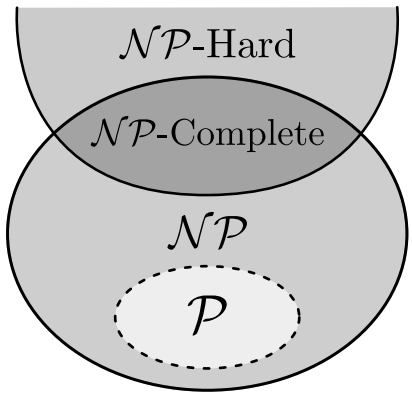


Figure 2.3: The relation between the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard.

- **polynomial-time verifiable** if a candidate solution can be verified to indeed be a solution in polynomial time.

All polynomial-time decision problems belong to the complexity class $\mathcal{P}$. Problems in $\mathcal{P}$ are sometimes called "easy", which should not be interpreted literally since even problems with astronomically large polynomial complexity order are included $\mathcal{P}$ [37].

The complexity class $\mathcal{NP}$ — nondeterministic polynomial — contains all problems for which an algorithm can guess a solution that can then be verified in polynomial time [37]. That is, $\mathcal{NP}$ contains all polynomial-time verifiable problems. The idea behind this is that problems in $\mathcal{NP}$ can be solved in polynomial time with access to an arbitrary number of computers, since each computer can guess its respective solution and then verify feasibility in polynomial time [37]. Every polynomial-time solvable problem is certainly polynomial-time verifiable, hence $\mathcal{P} \subseteq \mathcal{NP}$, as shown in Figure 2.3.

Reductions are algorithms that reformulate one problem into another [34]. This technique enables solving the original problem by applying an algorithm designed for a target problem. If a polynomial-time reduction exists from an $\mathcal{NP}$ problem to a $\mathcal{P}$ problem, then the $\mathcal{NP}$ problem becomes polynomial-time solvable: the reduction takes polynomial time, and solving the resulting $\mathcal{P}$ instance also takes polynomial time. Since $\mathcal{P}$ is a subset of $\mathcal{NP}$, this would prove that the original problem is in $\mathcal{P}$ — and if such reductions exist for

all $\mathcal{NP}$ problems, it would prove that $\mathcal{P} = \mathcal{NP}$. The question of whether $\mathcal{P} = \mathcal{NP}$ remains as one of the greatest unanswered questions of our time, with many computer scientists believing that $\mathcal{P} \neq \mathcal{NP}$, although without proof [37].

$\mathcal{NP}$-complete is a special class of $\mathcal{NP}$ problems: there exists a polynomial-time reduction from every problem in $\mathcal{NP}$ to every problem in $\mathcal{NP}$-complete [34]. This implies that every $\mathcal{NP}$-complete problem is at least as hard to solve as every $\mathcal{NP}$ problem. Therefore, finding a polynomial-time algorithm to solve any $\mathcal{NP}$-complete problem implies that every $\mathcal{NP}$ problem is polynomial-time solvable, making $\mathcal{NP}$-complete the prime problem set to target if aiming to determine if $\mathcal{P} = \mathcal{NP}$.

The $\mathcal{NP}$-hard complexity class can be considered to contain, as the name suggests, the hardest problems of all. Similar to $\mathcal{NP}$-complete problems, there exists a polynomial-time reduction from every problem in $\mathcal{NP}$ to every problem in $\mathcal{NP}$-hard [34]. In other words, $\mathcal{NP}$-hard problems are at least as hard as $\mathcal{NP}$ problems. However, $\mathcal{NP}$-hard problems are not necessarily polynomial-time verifiable, nor are they necessarily decision problems. Many optimization problems are neither polynomial-time solvable nor polynomial-time verifiable, and therefore belong to $\mathcal{NP}$-hard.

CHAPTER 3

Classical and Lifelong Multi-Agent Path Finding

This chapter examines the classical MAPF problem — the foundational challenge behind much of the research in MAPF — and the Lifelong MAPF variant. Each of these problems are defined formally, followed by a short summary of their respective variations found in the literature. A subsequent survey highlights several prominent approaches to solving both MAPF and LMAPF, identifying common techniques that enhance scalability across different methods. This analysis finalizes by answering research question **RQ1**.

3.1 Classical Multi-Agent Path Finding

The classical MAPF problem [15] is defined by a tuple

$$\langle \mathcal{G}, N, s, g \rangle$$

where $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ is a connected and directed graph with vertices $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$; N agents are located on $\mathcal{G}$, starting at a unique vertex by $s : \{1, .., N\} \rightarrow \mathcal{V}$ and assigned a unique goal vertex (commonly referred to as a *target*) by $g : \{1, .., N\} \rightarrow \mathcal{V}$. Although $\mathcal{G}$ is often undirected, alternative formulations do appear in the literature with $\mathcal{G}$ being *directed* (e.g. [38–40])

since no component inherently relies on undirectedness. Time is discrete and each agent executes an *action* at every timestep. There are two types of actions: *move* actions and *wait* actions. A move action $m = \langle v, v' \rangle \in \mathcal{V} \times \mathcal{V}$ (with $\langle v, v' \rangle \in \mathcal{E}$) means that an agent moves from v at one timestep to arrive at v' in the next timestep. For such a move action, let $from^v(m) = v$ and $to^v(m) = v'$. A wait action $w = v \in \mathcal{V}$ instead means that an agent waits in v until the next timestep. Let $from^v(w) = to^v(w) = v$.

Every agent $i \in \{1, .., N\}$ executes a sequence of actions $\pi_i = \langle a_1, a_2, \ldots, a_n \rangle$ called a *plan*, with the length $|\pi_i| = n$ being individual to each agent and varying across solutions. When deciding the agents' plans, *conflicts* must be avoided. A *vertex conflict* occurs when two agents occupy the same vertex at the same time, and a *swapping conflict* occurs when two agents traverse the same edge at the same time in opposite directions [15]. Most MAPF methods consider only these two conflict types — as do we in the following. For a plan π_i to be *valid*,

1. the first action must start at i's starting vertex — $from^v(a_1) = s(i)$,
2. each subsequent action must start where the previous ended — $from^v(a_k) = to^v(a_{k-1})$ for all $k = 2, .., n$, and
3. the last action must end at i's goal vertex — $to^v(a_n) = g(i)$.

A *joint plan* $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ contains one plan for each agent. A solution is a joint plan containing only valid plans and no conflicts. That is, for all pairs $\langle \pi_i, \pi_j \rangle$ where $\pi_i, \pi_j \in \Pi$, agents i and j are never located at the same vertex or traverse the same edge in opposite directions at the same timestep.

A solution's quality is determined by its evaluation on an objective function. The two most common objective functions σ in MAPF are the *makespan*,

$$\sigma(\Pi) = \max_{\pi \in \Pi} |\pi|,$$

and *sum-of-costs* (SOC), or *flowtime*,

$$\sigma(\Pi) = \sum_{\pi \in \Pi} |\pi|.$$

Letting $\mathbb{S}$ be the feasible set for a MAPF problem, an optimal solution Π^* minimizes σ over $\mathbb{S}$, $\Pi^* = \arg\min_{\Pi \in \mathbb{S}} \sigma(\Pi)$. Note that multiple optimal solutions to a MAPF problem may exist, we seek any of them.

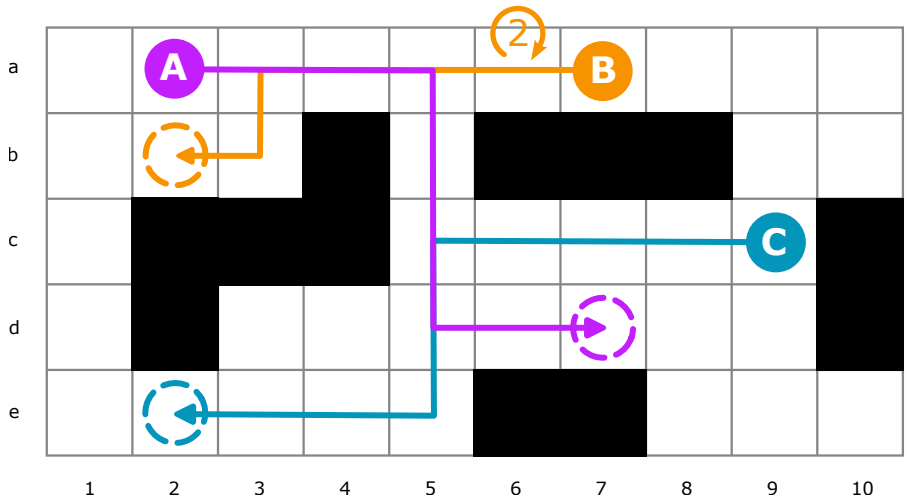


Figure 3.1: A MAPF problem with three agents, each with a respective start cell (circles) and goal cell (rings). Black cells are untraversable obstacles. A solution taking each agent to their goal is shown, with agent B waiting for two timesteps at $a7$ to avoid colliding with agent A.

Figure 3.1 illustrates a MAPF problem with three agents on a grid map. Agents can wait at empty cells and move to adjacent cells, with black cells representing untraversable obstacles. The underlying graph $\mathcal{G}$ has a vertex for each empty cell with edges connecting adjacent empty cells. A solution is also shown, where agents A and C perform only move actions until reaching their goals, while agent B executes two wait actions at $a7$ to avoid colliding with A. It takes 8 timesteps for agent A to arrive at its goal, 8 for B, and 9 for C, resulting in the makespan $\max(8, 8, 9) = 9$ and SOC $8 + 8 + 9 = 25$.

MAPF for optimal makespan, SOC, and total travel distance belongs to the class of $\mathcal{NP}$-hard problems [31]. Consequently, it is highly unlikely that a polynomial-time algorithm exists for solving MAPF optimally (else $\mathcal{P} = \mathcal{NP}$). This computational complexity imposes practical upper bounds on the problem sizes that optimal MAPF algorithms can handle. The MAPF problem is polynomial-time verifiable by checking every action pair for a conflict; with N agents, each with at most n actions, we check $(nN)^2$ action pairs — complexity $\mathcal{O}(N^2)$. However, to our knowledge, there does not exist a polynomial-time

algorithm for verifying that a given solution is optimal, making it difficult to confirm solution optimality.

MAPF Variants

The following introduces a few variations to the classical MAPF formulation. However, we defer descriptions of the LMAPF problem to the subsequent section, and the variants corresponding to continuous-time formulations to subsequent chapters. The following MAPF variants, and many more, are well documented in e.g. [15, 41–43].

In *Anonymous MAPF*, or *Unlabeled MAPF*, agents are not assumed to be assigned a goal vertex beforehand. Instead, the problem involves both assigning a goal vertex to each agent from a set of goals, and then planning their paths to those goals [15, 41]. This problem shares similarities with LMAPF, introduced in Section 3.2, below. A generalization of this is referred to as *Colored MAPF*, where agents are grouped into teams, each team being assigned a set of goals [15].

Including *deadlines* is useful in applications where agents must complete their tasks before a maximum time [15, 44]. For instance: manufacturing parts must be delivered before assembly or else production halts; airplanes must be ready for take-off at their allotted time or else delays occur. Without deadlines, if at least one solution exists for the MAPF problem then there exists infinitely many (see Section 5.1). The inclusion of deadlines constrains the feasible set to at most a finite set, since solutions cannot be delayed arbitrarily. Although restricting the feasible set may present challenges, it also presents opportunities for algorithms to reduce the explored set, potentially leading to faster convergence to solutions.

The common *stay-at-target* assumption means that agents remain at their goals until all other agents have arrived at their respective goals, as opposed to the less common *disappear-at-target* which assumes that agents are removed from the graph upon arrival. This means that, with the stay-at-target assumption, a plan π contributes to the objective function with a value of $|\pi|$, even if the agent momentarily visits its goal vertex at an earlier time than $|\pi|$ before leaving to later return. Although the disappear-at-target is less common, it is still considered in some work [45].

In many real-world applications, agents represent untethered autonomous mobile robots (AMRs) which rely on a finite on-board power source, such as

a battery or fuel reserve. This consideration offers several aspects to incorporate into the planning of agent movements. Many methods aim to reduce the power consumption by, for instance, considering the *sum-of-fuel* objective function [15, 46] or optimizing the low-level control of agent's kinematics [47]. Refueling operations commonly remove agents from transportation operations for some time, leading to a reduction in the AMR-fleet's transportation capacity which indirectly impacts makespan and SOC. Work such as [48] consider the refueling time directly when scheduling agents.

Several other types of conflicts are documented in [15]:

- *edge conflicts* — two agents traversing the same edge in the same direction at the same timestep,
- *following conflicts* — an agent is planned to occupy a vertex that was occupied by another agent in the previous timestep, and
- *cycle conflicts* — a set of agents switch positions in a rotating cycle pattern in the same timestep.

Edge conflicts are avoided by disallowing vertex conflicts, and neither following conflicts nor cycle conflicts necessarily mean that agents collide (in the real-world sense) with each other. These are the reasons, we believe, as to why our literature review has not found any studies considering these three constraint types.

In MAPF, *robustness* typically refers to a solution's ability to handle timing delays while still avoiding collisions. In the discrete-time MAPF formulation, a *k-robust* solution allows for any subset of agents to be delayed by up to k timesteps without collisions occurring [15, 49]. *MAPF with Uncertainty* is a related problem where delay-uncertainty is incorporated into plans to obtain a solution with a certain collision-free confidence level [41, 50].

Most MAPF formulations assume that traversing an edge takes exactly one discrete timestep. This assumption arises from the discrete-time, lock-step synchronization model where all agents execute actions simultaneously at each timestep. If an edge required more than one timestep to traverse, that agent would be unable to execute an action at the next timestep, violating the fundamental assumption that agents can act at every discrete time interval. Several approaches exist for handling non-unit-length edges in discrete-time MAPF. One method, demonstrated in Paper A, treats edge lengths as integer

multiples of a base unit-length. During multi-step edge traversals, the agent's action is effectively decided (to continue traversing the edge) until it completes the transition. An alternative approach to handling non-unit-length edges is to assume, in practice, that agents remain idle at their respective destination vertex until all agents have completed their actions. This strategy, however, introduces artificial waiting periods that directly degrade solution quality for time-based objective functions (e.g. makespan and SOC), as agents accumulate unnecessary delay times rather than making progress toward their goals.

Multi-Goal MAPF [51] is a variant where each agent is assigned multiple goals, instead of just one, and must visit each of the goals at least once in any order. This variant introduces the additional complexity of determining the order in which to visit the goals, commonly known as the *traveling salesperson problem* [52].

Transient MAPF [53] is a one-shot MAPF problem that provides more flexibility compared to classical MAPF when used to solve Lifelong MAPF. By repeatedly solving one-shot problems where one instance starts where the previous instance ended, Lifelong MAPF problems can be solved. However, the drawback of using classical MAPF with the common stay-at-target assumption for this is that agents are assumed to wait at their goals until all other agents have arrived at their respective goals [15]. That is, in MAPF it is not enough for an agent to have arrived at its goal; if an agent must move away from its goal to let another agent pass, then it must thereafter return to its goal. Transient MAPF assumes that an agent has reached its goal at the first time it arrived there. Thus, in a solution to Transient MAPF, an agent must have visited its goal but may not necessarily end its path there. Since agents do not necessarily need to end their plans at their goals, the set of feasible solutions is expanded, providing more flexibility and leading to better Lifelong MAPF solutions in almost all tested cases [53].

3.2 Lifelong Multi-Agent Path Finding

In classical MAPF, we assume that all agents' goals are known beforehand. However, in the real world where uncertainty lives, it is not always reasonable to assume that we have access to all information. For instance, stations in a factory may need restocking as parts are consumed. We cannot assume to know when every screw will be used and every production mistake will occur.

However, these all contribute to when restocking is required and therefore when an agent may need to transport goods to that station. Lifelong MAPF (LMAPF) [17], sometimes referred to as *Online MAPF*, handles this uncertainty and better reflects real-world never-ending operations by not assuming to know beforehand when and where agents must be. Instead, previously unknown *tasks* are released to the system at various times, each task specifying a particular goal vertex. The problem involves assigning agents to tasks, and then planning the agents' paths to those tasks. In this problem, we assume that there is no end to the incoming stream of tasks [17]. Therefore, the aim is to continuously complete tasks as they enter the system, and thereby avoid amassing a backlog.

Formally, a LMAPF problem [17] can be defined by a tuple

$$\langle \mathcal{G}, N, s, \mathcal{T} \rangle$$

where $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ is a connected and directed graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$; each of the N agents start at a unique vertex by $s : \{1, .., N\} \to \mathcal{V}$; and $\mathcal{T}$ is a possibly unbounded multiset over $\mathcal{V} \times \mathbb{N}$ containing tasks. Time is discrete. Knowledge of a task $\tau = \langle v, t \rangle \in \mathcal{T}$ is released to the system at timestep t, and is completed once an agent is located at v at some time $\geqslant t$.

Just as in classical MAPF, agents execute move and wait actions at every timestep, with multiple actions forming a plan $\pi_i = \langle a_1, a_2, \ldots, a_n \rangle$. Contrary to classical MAPF, however, a *valid* plan π_i in LMAPF requires only starting in the agent's starting vertex, $\mathit{from}^v(a_1) = s(i)$, and every subsequent action starting where the previous ended, $\forall k = 2, .., n : \mathit{from}^v(a_k) = \mathit{to}^v(a_{k-1})$. Since tasks enter the system over time, agent plans are periodically updated to attend to new tasks. The prefix of a plan containing all actions before the system's current time cannot be changed, only the suffix containing actions that have not yet begun executing. The same two conflict types as in classical MAPF, vertex and swapping conflicts, are also commonly considered in LMAPF; no two agents can occupy the same vertex at the same time-step, and no two agents can traverse the same edge in opposite directions. Thus, a joint plan $\Pi = \{\pi_1, ..., \pi_N\}$ containing one plan for each agent is a solution if every plan is valid and no two plans together cause a conflict.

Figure 3.2 illustrates an LMAPF problem with three agents attending to an incoming stream of tasks. At the current time $t = 8$, the tasks $\tau_1, \ldots, \tau_7$ have been released to the system and assigned to agents. Agent A was assigned task $\tau_1 = \langle b1, 0 \rangle$ at $t = 0$ and completed it at $t = 2$. Thereafter, it was assigned

$\tau_5 = \langle e3, 6\rangle$ and is planned to complete it at $t = 6 + 5 = 11$. Likewise for the other agents; B completed τ_3 at time $t = 5$, and is since $t = 7$ attending to τ_7, C completed τ_2 at $t = 3$ and τ_4 at $t = 7$, and is since $t = 8$ attending to τ_6. Future tasks, τ_8–τ_{10}, have not yet been released to the system and are therefore unknown to the planner.

Defining optimality in LMAPF presents fundamental challenges that distinguish it from classical MAPF. Unlike classical MAPF, which has a well-defined terminal state where all agents have reached their goal, LMAPF operates continuously with a potentially unbounded task set $\mathcal{T}$, making traditional notions of global optimality ill-defined. That tasks are continuously released complicates measuring solution quality, as it is affected by future information unavailable at the time when decisions are to be made. This introduces an inherent randomness that can dominate differences in performance across methods. For instance, an agent positioned near a newly released task will be able to complete the task earlier than if it were located farther away, all else being equal. Consequently, comparing solutions based on metrics such as makespan or SOC can be misleading. Furthermore, the makespan and SOC of a solution grows with time, as more tasks enter the system, requiring some arbitrary end-time to be set in order to obtain a remotely fair comparison value. Therefore, many LMAPF works focus on *throughput* — the average number of completed tasks per timestep. When averaged over an extended period of time, the metric should be comparably stable and therefore provide an indication of algorithmic performance independent of the time the system is running.

LMAPF Variants

Relative to the classical MAPF problem, our literature review found fewer variants of the LMAPF problem. We present these variants here, except for the *Continuous-Time LMAPF* which is treated in detail in Chapter 4.

The *Multi-agent Pickup-and-Delivery* (MAPD) problem [17] can be regarded as a generalization of the LMAPF problem. In MAPD, tasks contain both a pickup and delivery position; the assigned agent must first move to the pickup position (supposedly to pick up goods) and then to the delivery position (to drop those goods off). MAPD represents a more realistic formulation for many practical applications. Taxi drivers must first reach passengers before transporting them to their destinations, and similarly, most

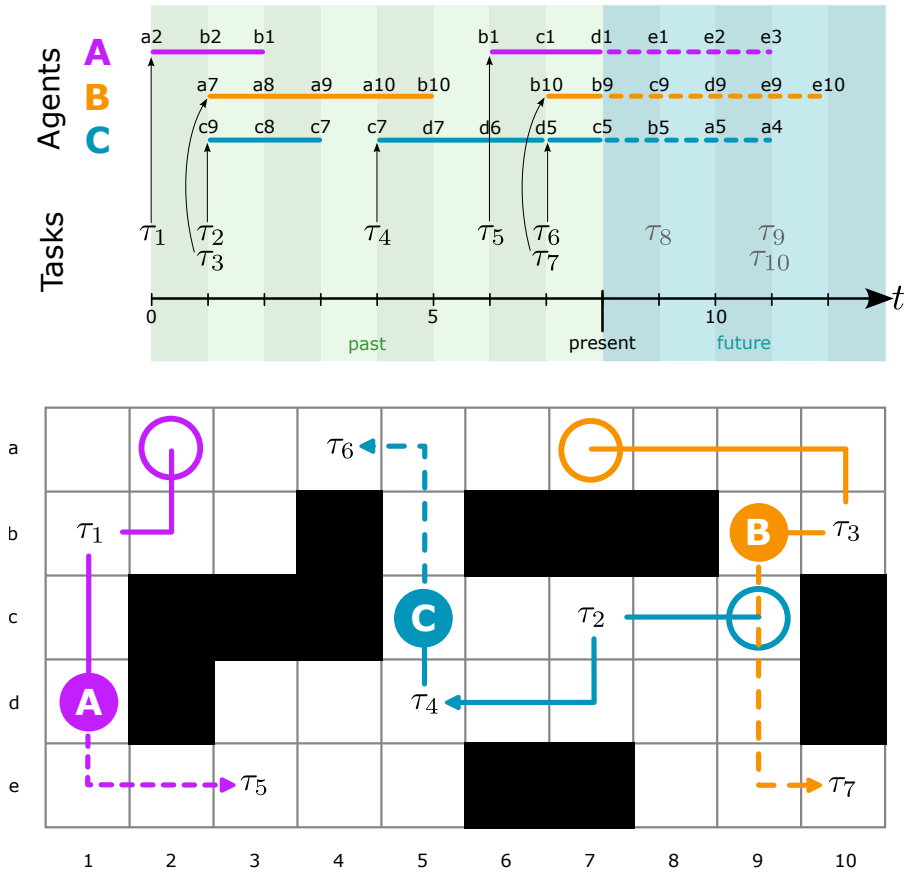


Figure 3.2: An LMAPF problem with three agents (A, B, C) and with the system currently at timestep $t = 8$. **Top:** A timeline of agent movements and task releases, with arrows showing which agents are assigned which tasks. **Bottom:** An illustration of the agents in the environment as they attend to tasks, showing at time $t = 8$ their starting positions (unfilled circles), current positions (filled circles), completed movements (full lines), and planned movements (dashed lines).

industrial AMR deployments involve collecting items from pickup locations before delivery. LMAPF solution approaches can be adapted to MAPD problems through constrained task assignment. As demonstrated in [14], this is achieved by treating pickup and delivery locations as paired tasks, where an agent assigned to a pickup task must subsequently be assigned the corresponding delivery task. The constrained assignment approach demonstrates that LMAPF and MAPD are computationally related, with MAPD effectively decomposable into constrained LMAPF instances.

Multiple LMAPF variants relate to the number of goals assigned to agents and the order in which they must be visited. *Multi-Goal Multi-Agent Pickup and Delivery* (MG-MAPD) [54] generalizes MAPD by allowing each task to contain a sequence of vertices (instead of just one pickup and one delivery vertex). An assigned agent must visit the vertices in the specified order. A similar variant where agents can visit the goal vertices in any order is described in [46]. These formulations are particularly relevant for industrial applications where agents must perform multi-stage operations, such as collecting components from multiple warehouse locations before assembly or visiting several delivery points for a single pickup.

MAPD with External Agents [55] addresses scenarios where the team of controlled agents must operate in an environment populated by moving external agents that cannot be directly controlled. Such entities — including human workers, autonomous vehicles from other teams, or human-controlled vehicles — affect path planning and collision avoidance but follow their own objectives and cannot be coordinated with the team agents. This variant reflects realistic warehouses where autonomous vehicles share the operational space with other moving entities.

In *Capacitated MAPD* [56], agents can carry items from multiple tasks simultaneously rather than being limited to one item at a time. In this variant, agents may pickup multiple items from different locations before delivering them, thereby enabling more efficient task execution compared to single-item transport. This extension increases the problem complexity since the decision to collect additional items, and the order in which to do so, must be decided.

Cache-enhanced LMAPF [57] models warehouse environments with designated cache grids, allowing agents to temporarily store items during task execution. This spatial extension introduces optimization considerations beyond traditional pickup-delivery constraints, enabling agents to strategically posi-

tion items for future tasks. Naturally, this introduces additional complexity to the problem.

3.3 Common MAPF and LMAPF Approaches

MAPF and LMAPF are computationally challenging problems, with scalability often achieved through various strategic simplifications. This section surveys prominent approaches and identifies three main strategies for scalably solving MAPF and LMAPF:

- prioritized planning,
- windowed planning, and
- dimensional simplification.

We additionally introduce exact solution methods [58–61], as well as a few methods [62–66] in the relatively young field of machine-learning-based MAPF.

Exact Methods

An exact algorithm is defined in Chapter 2 to only return optimal solutions. Several exact algorithms for finding MAPF solutions have been introduced in the literature. While typically unable to handle large agent counts or complex scenarios within practical time limits, these methods are nonetheless foundational for understanding the problem. A common theme among the exact methods presented here, as we will see in this section, is the adoption of a "lazy" approach where agent movements are planned individually, and conflicts are only addressed as they occur.

*M** [58] extends A* with *sub-dimensional expansion* to plan optimal paths efficiently. The algorithm begins by computing individually optimal paths for each agent (called a *robot* in [58]). If collisions are detected, the method then adaptively expands into a higher-dimensional space where paths are considered jointly, to resolve collisions. This approach allows M* to safely ignore regions of the search space representing paths for agents not involved in collisions, where a shortest-path algorithm suffices. This significantly reduces the average-case computational cost compared to the pure A* search in the full space of jointly-considered agent paths [58].

Increasing Cost Tree Search (ICTS) [59] uses a two-level search approach: the high-level searches the *Increasing Cost Tree* where each node represents a cost vector (one cost per agent), starting from individually optimal costs; the low-level validates whether collision-free paths exist at those costs. If a collision-free path for an agent with the cost specified by the node's cost vector does not exist, child nodes are generated by incrementing the cost for one agent, continuing until a feasible collision-free solution is found.

Conflict Based Search (CBS) [60] is a widely adopted exact MAPF method, with many extensions targeting various properties and MAPF variants [16, 67–71]. CBS is inspired by ICTS, using a similar two-level search approach: the high-level searches the *Constraint Tree* (CT), where each node contains a set of constraints and a candidate solution satisfying those constraints; the low-level finds the optimal path for individual agents given their constraints. Starting at the root with an empty constraint set, CBS validates the candidate solution by checking for conflicts. When a conflict is detected between two agents at a specific location and time, two CT child nodes are generated — each adding a constraint for one of the conflicting agents to avoid that specific collision — continuing until a conflict-free solution is found. If no solution exists, CBS does not guarantee termination.

Lazy Constraints Addition Search for MAPF (LaCAM*) [61] represents a recent evolution in exact MAPF methods, combining optimality guarantees with practical scalability. Unlike the previous exact methods, LaCAM* is an *anytime* algorithm, meaning that it quickly finds sub-optimal solutions and gradually refines them until a satisfactory solution is found or the available computation time has been exhausted. LaCAM* performs a high-level search in a tree of configurations, a configuration specifying the location of every agent, with each node containing a constraint tree which initially contains only a root node (representing no constraints). High-level nodes are expanded multiple times, each expansion leading to the creation of a single child node representing a possible configuration in the next time step. An expansion of a node N entails:

- A node η in N's constraint tree is selected according to breadth-first search. The constraint tree is structured such that each node at depth 1 constrains agent 1's next action, each node at depth 2 constrains both agents 1 and 2's next action, etc.
- *Priority Inheritance with Backtracking* (PIBT) [72] (see next section)

is used to generate a configuration for the next timestep satisfying the constraints at η. This configuration is used to create a child node of N with an empty constraint tree.

- The constraint tree node η is marked as searched, upon which its children (one for each action of the next agent to be constrained) are spawned.

This method lazily searches in the direction of where PIBT goes, thereby achieving remarkable scalability. LaCAM* is able to sub-optimally solve 99% of benchmark instances from [15] with up to 10 000 agents within 30 seconds while guaranteeing eventual convergence to optima.

While exact methods like M* and CBS remain foundational, the anytime approach of LaCAM* demonstrates that optimality guarantees need not preclude practical scalability. However, for the largest and most complex instances, sub-optimal methods such as those explored in the following sections remain necessary.

Prioritized Planning

Prioritized planning is a widely adopted strategy for achieving scalability, where agents are planned sequentially, one at a time [73, 74]. The priority of agents denotes the order in which they are planned, however, this prioritization need not be explicit. For instance, randomly ordering agents and then planning them sequentially implicitly enforces a prioritization. By planning sequentially rather than jointly, the problem reduces to multiple instances of single-agent path planning. This reduces the computational complexity, though at the cost of completeness and optimality guarantees.

Cooperative Pathfinding [75] introduces the foundational prioritized planning approach to MAPF. The algorithm plans paths for agents sequentially in an arbitrary order, with each agent using A* to search for a collision-free route to its goal while treating previously planned agents' paths as dynamic obstacles. Three variants that build upon each other are proposed, progressively enhancing performance. *Cooperative A** (CA*) uses a reservation table, recording occupied space-time positions for previously planned agents, excluding these from subsequent agents' A* searches. CA* suffers from order-sensitivity — agents that are planned early constrain later plans, potentially causing failures when agents that should wait are planned too early. *Hierarchical Cooperative A** enhances CA* with *Reverse Resumable A** (based on *Hi-*

*erarchical A** [76]), which computes heuristics via backward search from each goal in an agent-free environment, reusing the search state across evaluations for better guidance without redundant computation. *Windowed Hierarchical Cooperative A** (WHCA*) limits planning to w steps ahead, committing only the actions within the immediate window and continuously re-planning during execution. This reduces memory, mitigates order-sensitivity, and enables adaptation to dynamic conditions at the cost of solution quality. While Cooperative Pathfinding establishes the prioritized planning paradigm, subsequent work [17, 61, 72, 77, 78] extends these ideas to more complex problem variants and with improved scalability.

Token Passing (TP) [17] extends prioritized planning to MAPD. Agents coordinate via a shared token containing all planned paths and available tasks. When idle, agents sequentially request the token, select unassigned tasks with minimum heuristic cost, and plan collision-free space-time paths avoiding reserved positions. TP guarantees completeness for certain types of MAPD instances and efficiently handles hundreds of agents. An enhanced variant of TP, *Token Passing with Task Swaps*, allows agents to swap previously assigned tasks that have not begun execution, improving solution quality through limited inter-agent negotiation at the cost of additional computation.

While TP extends prioritized planning to lifelong scenarios, *Priority Based Search* (PBS) [77] addresses a fundamental limitation: order sensitivity. PBS systematically explores the space of different priority orderings. Rather than committing to an arbitrary agent order like in CA* and TP, PBS uses a depth-first search over a tree of partial priority orderings, branching to explore both possible orderings between two agents when conflicts arise between them. This approach shares similarities with exact methods like M* and CBS, which also branch when conflicts arise. However, PBS operates differently: it establishes agent-level priority relationships (agent i has priority over agent j), whereas CBS and M* impose location-time specific constraints (forbidding the use of specific vertices at specific times). In other words, PBS can be summarized as applying CA* for different orders of agent priority, making it more efficient but also leading to worse solutions. Despite these simplifications, PBS achieves near-optimal solution quality while maintaining efficiency, solving instances with 600 agents in under a minute and succeeding where fixed-priority methods fail [77].

Priority Inheritance with Backtracking (PIBT) [72] addresses MAPF by

iteratively planning one timestep at a time. Each agent receives a unique priority that persists until reaching its goal. Agents plan their next locations in priority order, avoiding positions occupied by higher-priority agents. When a low-priority agent blocks a higher-priority agent's path, it temporarily inherits that priority when moving out of the way, allowing it to block otherwise higher-priority agents. When an agent cannot move without causing conflicts, backtracking signals to prior agents to reconsider their decisions. On biconnected graphs, PIBT guarantees that the highest-priority agent will move along its preferred edge, therefore guaranteeing that all agents will eventually reach their goal within a finite time.

LaCAM* (introduced above in Exact Methods) demonstrates the effectiveness of prioritized planning mechanisms within search-based approaches. For instance, on the grid maps, every agent at each timestep executes one of 5 actions (move up, down, left, right, or wait). With N agents, there are up to 5^N possible configurations in the next timestep. Instead of searching the entire 5^N possible configurations in the next timestep, LaCAM* uses PIBT to generate one high-quality configuration at a time and thereby initially narrows the search, allowing it to broaden as time permits. This integration of prioritized planning into a systematic search framework exemplifies how prioritization strategies can scale to thousands of agents while retaining solution quality.

Having introduced methods spanning pure sequential planning (CA*, TP), adaptive priority exploration (PBS), and dynamic one-step iterative methods using prioritized planning (PIBT, LaCAM*), we now examine hybrid approaches that combine prioritized planning with iterative refinement using coupled planning.

MAPF-LNS [78] applies *Large Neighborhood Search* [79] to MAPF as an anytime framework. Starting from an initial solution, it iteratively selects subsets of agents, called *neighborhoods*, discards their paths, and re-plans them to reduce total cost. The algorithm can use either prioritized planning or coupled search (CBS, *Explicit Estimation CBS* [80]) for re-planning — empirically, prioritized planning performs best by enabling more re-planning iterations within time limits, as opposed to fewer, coupled, re-planning iterations. The algorithm adaptively selects which agents to re-plan using multiple strategies, adjusting its approach based on which selection successfully improves the solution. *MAPF-LNS2* [81] extends this framework to start from

infeasible collision-containing solutions and repairs them until collision-free, rather than improving an already conflict-free solution. By minimizing collisions first, then switching to cost optimization, MAPF-LNS2 dramatically improves scalability — achieving higher success rates than all other methods that they compare with while maintaining a $< 1\%$ sub-optimality in 99.5% of benchmark instances with known optimal solutions.

Planning and Improving while Executing (PIE) [82] introduces a hybrid framework, combining prioritized and coupled planning through concurrent planning and execution. PIE decouples planning from execution — agents begin executing immediately while the planner continues improving uncommitted paths. It uses LaCAM* for fast prioritized initial solutions and MAPF-LNS for coupled refinement of agent neighborhoods, committing only k actions at a time. This substantially reduces goal achievement time for both one-shot and lifelong MAPF. PIE is extended in [83] for robust execution policies and delay-aware re-planning, improving throughput up to threefold as execution delays occur.

TSWAP [84] addresses Anonymous MAPF, where any agent can be assigned to any goal. TSWAP uses prioritized one-step planning similar to PIBT, starting from an arbitrary target assignment and planning agents sequentially. At every time step, each agent either (1) stays where it is if located at its goal, (2) moves toward its target if the next node is available, (3) swaps goals with an adjacent agent that is occupying the next vertex toward its goal, (4) resolves a deadlock by rotating goal assignments, or (5) waits. This flexibility in target assignment combined with prioritized planning enables TSWAP to efficiently handle Anonymous MAPF, solving instances with 2 000 agents near-optimally within seconds while guaranteeing eventual goal achievement.

Prioritized planning has evolved significantly from simple sequential approaches to sophisticated hybrid frameworks. Early methods like Cooperative A* established the paradigm but suffered from order-sensitivity. Subsequent work addressed this limitation through systematic priority exploration (PBS), dynamic priority mechanisms (PIBT), and integration with search-based frameworks (LaCAM*). Modern approaches increasingly combine prioritized initialization with coupled refinements (MAPF-LNS, PIE), leveraging the computational efficiency of sequential planning while recovering solution quality through iterative improvement. The spectrum ranges from purely sequential methods achieving polynomial complexity to hybrid frameworks that

approach near-optimal solutions. While prioritized planning sacrifices completeness guarantees, its computational tractability makes it the dominant approach for practical large-scale applications, with recent methods scaling to thousands of agents.

Windowed Planning

The concept of planning within a limited time-frame, know as *windowed planning* or *windowed search*, has been used in many fields for a long time. The moving horizon approach used frequently in *Model Predictive Control* can be traced back to the 60's [85], with similar ideas from Bellman on finite-horizon problems [35] laying the foundation of large parts of Dynamic Programming. Planning within a horizon is an appealing way to reduce computational complexity, as the problem can then be broken down into sequential steps, starting at the current time and planning each window sequentially until a full solution is found. The drawback of this is often that guarantees of optimality are forfeited, similar to when using prioritized planning, as the full problem is never considered in its entirety. Several methods from the previous section are revisted here.

As the name suggests, *Windowed Hierarchical Cooperative A** (WHCA*) applies windowed planning. Figure 3.3 illustrates WHCA*'s approach: agent movements are planned within a w-timestep window into the future, ensuring that collisions are avoided within this window. Thereafter, $h < w$ timesteps (specifically, $h = w/2$ in WHCA*, while [14] generalizes h) of actions are committed, upon which the movements for the next w-timesteps are computed. Results in [75] show that windowed planning offers faster computation times at the cost of potential incompleteness, with smaller window sizes being particularly susceptible to deadlocks.

PIE takes a fundamentally different approach to windowed planning compared to WHCA*. Instead of planning for only a fixed window of w timesteps ahead, PIE generates an initial feasible plan for each agent to reach their goal and iteratively refines these plans during execution. As refinement occurs, only the next k actions of the currently best solution are committed to and executed by agents. In the meantime, the planner continues refining the remainder of the plan beyond these committed actions. Thus, the committed interval k defines which actions of the currently best plan are executed, while that plan is continuously refined from the end of the committed portion

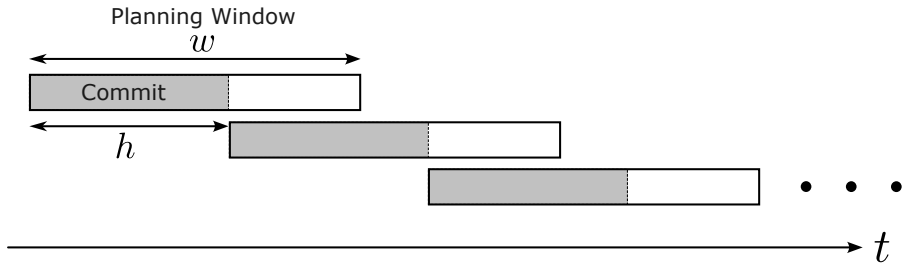


Figure 3.3: Windowed planning can be done by planning w timesteps into the future, committing $h < w$ timesteps of actions, and then repeating from h timesteps in the future.

onward.

Rolling Horizon Collision Resolution (RHCR) [14] uses a windowed approach similar to WHCA* to solve LMAPF. This approach is favorable in LMAPF as it keeps agents continuously engaged, avoiding idle time and thus increasing throughput, while generating flexible plans that adapt to continuously incoming tasks. Multiple solvers are tested for planning within the w-timestep window: CA* [75], PBS [77], CBS [60], and *Enhanced CBS* [67]. RHCR demonstrates two key contributions. First, it provides a flexible framework for adapting MAPF solvers to the LMAPF problem, through windowed planning. Second, the work highlights the scalability of windowed approaches compared to solving the full problem, demonstrating that resolving collisions over long time-horizons does not necessarily lead to significantly higher quality solutions. This windowed approach enables solving problems with up to 1 000 agents in large warehouse maps.

Windowed MAPF with Completeness Guarantees (WinC-MAPF) [86] solves MAPF using a windowed approach while ensuring completeness. A drawback of most windowed methods is that they can get stuck in deadlocks and livelocks, typically when the size of the window is smaller than the length of an escape plan. WinC-MAPF addresses these issues by introducing a memory mechanism that prevents agents from repeating unproductive movement patterns. It achieves this through an idea from single-agent path finding: when an agent visits a state that does not lead to progress, the state is marked with a penalty. Future planning therefore avoids these unproductive states. However, to make this approach computationally tractable, WinC-MAPF fo-

cuses its tracking on small groups of agents that *actually* interact with each other. This makes the approach practical while maintaining completeness. The framework is flexible enough to work with extremely short planning horizons, one of their key contributions being *Single-Step CBS* [86] which plans only one timestep ahead.

Similar to WinC-MAPF, TSWAP [84] is also able to find solutions — although, for the Anonymous MAPF problem — by planning with a window of one timestep into the future and also maintaining completeness. It does this by leveraging the problem's inherently flexible goal assignments. Due to its one timestep strategy, TSWAP can apply to both one-shot and lifelong Anonymous MAPF. For offline problems, the short planning horizon allows for rapid computation, allowing for problems with up to 2 000 agents being solved [84]. In online settings, the one-timestep approach provides inherent robustness and flexibility to execution delay uncertainty and incoming tasks. That is, by re-planning at every timestep, delayed agents that fall behind schedule or encounter obstacles are naturally accommodated for, making TSWAP suitable for many real-world applications.

These windowed approaches highlight both the appeal and challenges of horizon-limited planning for MAPF. The fundamental trade-off is the same as for prioritized planning and remains consistent across methods: windowed planning offers significant computational advantages — enabling faster computation times, adaptability to dynamic environments, and scalability to hundreds and thousands of agents — though at the cost of optimality and, in some cases, completeness. However, the diversity of approaches shows different ways of managing these tradeoffs. WHCA* and RHCR represent classical planning within fixed horizons, prioritizing computational speed while accepting the risk of deadlocks. PIE decouples execution from planning, continuously refining the solution while committing only to short action sequences, effectively blending advantages of windowed and full-horizon planning. WinC-MAPF tackles the completeness problem, showing that windowed planning — even with a window of only one timestep — does not necessarily sacrifice theoretical guarantees. Together, these methods highlight that windowed planning is not a single technique but a spectrum of approaches, each suited to different applications with different requirements on computational efficiency, solution quality, and robustness.

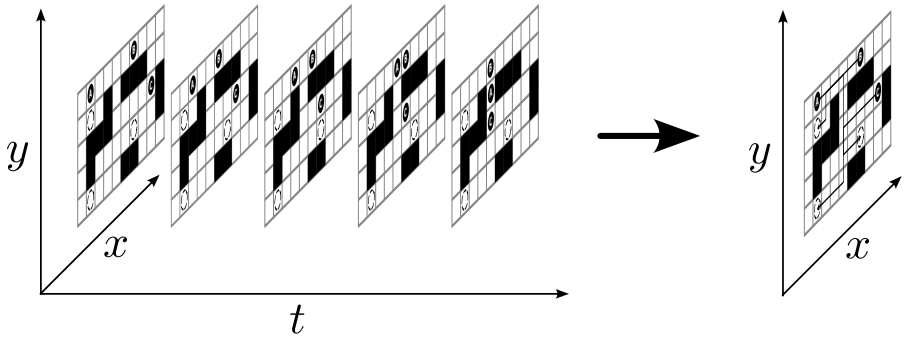


Figure 3.4: Illustration of dimensional simplification, where 3-dimensional space-time (x, y, t) is simplified to 2-dimensional space (x, y).

Dimensional Simplification

In this section — the last with a dedicated solution approach — we discuss three methods: *Space-Order CBS* [87] and the already-introduced WHCA* [75] and TSWAP [84]. These methods leverage simplifications in the typically three-dimensional space in which agents move, consisting of two spatial dimensions and one time dimension. Figure 3.4 visualizes dimensional simplification, showing how a 3-dimensional space-time is simplified by removing the time dimension.

The hierarchical part of WHCA* computes heuristics by discarding the time dimension and only considering the two spatial dimensions. More specifically, it uses an A* search going backwards through the map, computing the shortest spatial distance from every position to that goal. This precomputed spatial distance serves as an admissible heuristic for the space-time A*— that is, an agent can never reach the goal faster than this spatially computed lower bound allows. The heuristic is computed once per goal and then used repeatedly throughout all windowed space-time searches, providing an early example of dimensional simplification for achieving computational efficiency.

TSWAP essentially eliminates the time dimension entirely from the search, with the one-timestep planning being done entirely in the spatial dimension. That is, since the planning is done with a horizon of one timestep, agent's actions do not depend on planning through the time dimension but instead depend on rules based on the current spatial configuration. Concretely, the

action that an agent takes at some timestep depends entirely on its current location and the current locations of other agents. When agents move, they move according to time-independent planning — abandoning all timing assumptions such as synchronization and delay probabilities. This approach represents a fundamental reduction from the three-dimensional space-time problem to a purely spatial problem. As a result, the algorithm runs exceptionally fast, solving instances with up to 2 000 agents near-optimally within seconds. However, these results must consider that TSWAP solves the *Anonymous* MAPF problem, which allows for goal swapping. Nonetheless, the algorithm provides an important perspective on dimensional simplification and demonstrates how problem specific flexibility can compensate for extreme temporal short-sightedness.

Space-Order CBS (SO-CBS) [87] is an adaptation of CBS [60], where high-level branching is done on space-visitation orderings rather than on spatial-temporal constraints. The key insight is to view paths as sequences where agents visit specific locations in relative orders (e.g., 1st vs 2nd) as opposed to specific timesteps. When a conflict is detected between two agents at a location, resulting in branching in the high-level constraint tree, each branch enforces an ordering between the two agents: one specifies that agent A visits the location before agent B, while the other specifies the reverse ordering. The low-level search then finds paths for each agent, satisfying the ordering constraints present at the current high-level node. In this way, the high-level search replaces absolute time with relative orderings, operating in the space of all possible visitation orders rather than the three-dimensional space-time problem. This directly produces a *Temporal Planning Graph* that explicitly minimizes coordination requirements, leading to greater robustness when agents experience execution delays.

These three approaches demonstrate distinct strategies for dimensional simplification, each leveraging different problem-specific attributes. WHCA* uses pre-computed spatial heuristics to guide space-time search, TSWAP eliminates temporal reasoning to instead utilize purely spatial-based rules, and SO-CBS transforms the time-dimension into a space of relative orderings. These methods, and those discussed in previous sections, illustrate a key principle: problem simplification requires identifying which aspects of the problem that are essential for solution quality and which can be approximated or eliminated based on specific problem characteristics.

Learning-Based Approaches

Recent times have seen an immense growth in the use of machine learning (ML) based methods, commonly attributed to the exponential increase in data availability and cheap computing power [88]. Consequently, ML is finding its way into the field of MAPF, with challenges related to representing a MAPF state, planning agent actions, and executing actions in uncertain real-world environments [89]. In this section, we discuss the relatively new field of ML-based methods for solving MAPF.

PRIMAL [62] is a *decentralized* framework for solving MAPF, where the actions of each agent are governed by a policy based on the agent's observable environment. The policy is trained with *Reinforcement Learning* (RL) to find efficient single-agent plans, and *Imitation Learning* (IL) from a centralized planner that considers the movements of all agents jointly. This method is for grid-worlds, where an agent's state is the 10×10 grid centered at the agent's position, containing information about obstacles, the agent's goal, and other agents' positions and goals. The policy is approximated using a Deep Neural Network (DNN) that takes the agent's state as input and returns, among other values, a five-vector with one element for each possible action (to move in each of the four cardinal directions, or stay). Since this is a decentralized framework, the number of agents has little impact on the computation time as they each compute their own actions. Instead, the limitation lies in the kinds of problems can be solved. Results are promising on maps with relatively few obstacles, with high success rates with up to 1 024 agents. However, performance degrades in obstacle-dense environments compared to sparse maps where PRIMAL can match or exceed centralized planners.

PRIMAL$_2$ [63] extends PRIMAL to MAPF *and* LMAPF, scaling to thousands of agents on obstacle-dense maps. The approach incorporates richer state representations and relies more heavily on IL from centralized planners to learn jointly favorable behaviors that would be difficult to acquire through RL alone. Results demonstrate improvements over PRIMAL, however, the gains vary depending on scenario complexity and agent density.

Recent work [64] explores using *Graph Neural Networks* (GNNs) [26] for solving MAPF. They too focus on decentralized control, however, extending to allow communication between nearby agents. A *Convolutional Neural Network* [90] is used to extract features from the agent's state, which are then use as node-features in the GNN to allow for the communication between agents.

Finally, an action-vector (similar to PRIMAL) is returned. Although no comparison is done with PRIMAL, results do show how communication using the GNN substantially increases the success rate compared to without.

FOLLOWER [65] addresses decentralized LMAPF by combining RL with more a more traditional search algorithm. First, each agent plans a path using a heuristic search algorithm which indirectly avoids potential collisions with other agents. Specifically, A* is used to plan the agent's path, with the cost of transitioning to a cell consisting of both a static and dynamic component. The static cost of a cell is inversely proportional to the average cost of the paths starting in the cell and ending in all other cells. This is motivated by considering that cells with lower path costs are more likely to be included in shortest paths from one node to another, and therefore more likely to be congested, such cells should therefore be avoided. The static cost depends only on the map topology and therefore does not change. The dynamic cost of a cell is unique to an agent, simply being 1 if the agent observes another agent there, 0 otherwise. A *Recurrent Neural Network* [91, 92] is used to approximate a policy function, taking as input a $2 \times m \times m$ tensor (m being the observation range), with one channel containing obstacle locations and the other containing the current path and other agent locations. Thus, the policy is trained to follow the agent's path while avoiding other agents. Experiments show superior throughput compared to, among others, PRIMAL_2, with up to approximately 200 agents. However, RHCR with a 10 second time limit outperforms FOLLOWER by a large margin on warehouse maps.

MAPF-GPT [66] relies entirely on IL to solve both MAPF and LMAPF in a decentralized fashion. A *transformer network* [27] to taught to approximate an expert MAPF solver by learning from recorded observation-action pairs. LaCAM is used to solve MAPF problem to generate 1 billion observation-action pairs, and RHCR is used to solve LMAPF problem to generate 90 million observation-action pairs. Thus, LaCAM and RHCR are used as expert MAPF solvers. Results show comparable performance compared to LaCAM on MAPF problems with fewer agents, in the $8 - 32$ range. However, the success rate declines with more agents. Additionally, they test MAPF-GPT — with and without fine-tuning on the LMAPF expert data — on LMAPF problems. Both versions of MAPF-GPT is outperformed by both RHCR and FOLLOWER on all maps.

The emerging ML-based MAPF methods show potential, however, current

approaches typically handle tens to hundreds of agents compared to the hundreds to thousands routinely solved by classical methods like CBS, RHCR, and LaCAM*. While methods such as PRIMAL, FOLLOWER, and MAPF-GPT show promise in specific scenarios, they generally under-perform state-of-the-art classical solvers. As the field matures through improved architectures, larger datasets, and hybrid approaches, learning-based methods may develop complementary strengths to traditional MAPF solvers.

3.4 Answering Research Question 1

To answer **RQ1**— *How can Lifelong MAPF algorithms achieve real-time performance for large-scale industrial fleet management?* — we draw conclusions based on the surveyed MAPF and LMAPF methods in Section 3.3, with Paper A applying several of the identified strategies for a computationally tractable LMAPF solver.

We identified three main strategies for achieving scalable MAPF and LMAPF methods: prioritized planning, windowed planning, and dimensional simplification. Several methods use combinations of these strategies. For instance, WHCA* applies prioritized planning and windowed planning at its core and dimensional simplification for computing heuristics, resulting in one of the earliest computationally tractable algorithms. TSWAP exemplifies a more recent application of these principles by applying prioritized planning and one-step windowed planning using PIBT, and operating entirely in the spacial dimension with rule-based movements. TSWAP is complete and thereby shows that these strategies do not necessarily lead to the loss of theoretical guarantees. Therefore, these three main strategies provide a selection of ways to achieve real-time performance for large-scale industrial fleet management.

Paper A verifies these insights by applying several of the identified strategies, resulting in a computationally tractable *Fleet Manager* (FM) for solving the LMAPF problem. The FM applies prioritized planning, with agents being ordered by distance to the next task in the queue. Furthermore, FM applies dimensional simplifications in its *conflict manager* when handling cascading conflicts (where a conflict arises from an agent moving to avoid an earlier conflict). Such conflicts are handled entirely in the spatial domain to reduce complexity, with a strategy similar to PIBT. However, in FM this leads to potential deadlocks, as described with more detail in Paper A. Finally, the *path*

planner in FM applies a strategy similar to FOLLOWER's dynamic cost. That is, when an agent's path is planned from its current location to its goal location, transition costs are temporarily increased if another agent is scheduled to occupy it. This is also done entirely in the spatial domain, thereby reducing complexity by avoiding the time dimension. Experiments comparing the Fleet Manager to RHCR show how the Fleet Manager is able to achieve superior throughput, with computation times in the milliseconds.

CHAPTER 4

Lifelong Multi-Agent Path Finding in Continuous Time

The LMAPF problem introduced in Chapter 3 builds upon several underlying assumptions: time is discrete, every agent executes an action at every timestep and takes exactly one timestep to complete it, and agents are represented as sizeless points. In the real world, time is continuous, agents take space, and not all environments can be represented by neat grid-networks with equal-length edges. Therefore, the assumptions made in LMAPF simplify the problem by ensuring that agents move synchronously instead of asynchronously, and that collision detection does not depend on the actual shapes of the agents but instead on their location on the graph. Additionally, artificially constraining the problem to only allow for actions at fixed timesteps can potentially lead to worse solutions than without such constraints, particularly if agents must wait idly until the next timestep to begin the next action when they otherwise could have proceeded along the path to their goal. As we will see in this chapter, extending LMAPF to the continuous-time domain expands the representational power of the graph by allowing any-length edges and incorporating real-world spatial properties that otherwise are abstracted away. However, as there are no free lunches, we will explore several complications that emerge when moving from discrete to continuous time.

This chapter begins with a formal definition of the continuous-time LMAPF problem (LMAPF_R) and comparisons between it and LMAPF. Thereafter, several challenges that emerge when time is continuous are highlighted. Finally, research question **RQ2** is answered through the contents of this chapter and Paper B.

4.1 A Formal Definition of LMAPF_R

Beginning with formal definitions, an LMAPF_R problem can be defined by a tuple

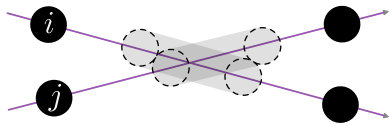
$$\langle \mathcal{G}, \mathcal{M}, N, s, \mathcal{T} \rangle$$

where $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}, \mathcal{W} \rangle$ is a directed, weighted, and connected graph with vertices $\mathcal{V}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and weights $\mathcal{W} : \mathcal{E} \to \mathbb{R}^+$ mapping edges to traversal times; $\mathcal{M}$ is a metric space, with $\mathcal{M}(v)$ mapping a vertex v to a d-dimensional position in $\mathbb{R}^d$; N agents each start at a unique vertex by $s : \{1, .., N\} \to \mathcal{V}$; and $\mathcal{T}$ is a possibly unbounded multiset over $\mathcal{V} \times \mathbb{R}$ containing tasks, where task $\tau = \langle v, t \rangle \in \mathcal{T}$ is released to the system at time t and is completed once an agent is located at v at some time $\geqslant t$.

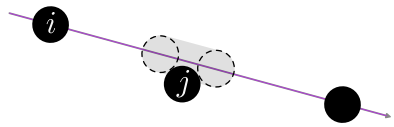
There exists two types of actions: *move* actions and *wait* actions. Move actions can be formulated in various ways. We adopt a simplified formulation where agents are assumed to traverse edges in straight lines. Thus, a move action $m = \langle e, t \rangle \in \mathcal{E} \times \mathbb{R}$ means that an agent traverses $e = (v, v')$ in a straight line from $\mathcal{M}(v)$ to $\mathcal{M}(v')$, starting at time t, taking $\mathcal{W}(e)$ time to complete. An alternative definition of a move action is provided in Chapter 5, where each move action corresponds to one of potentially several trajectories starting at $\mathcal{M}(v)$ and ending at $\mathcal{M}(v')$. Although this formulation of a move action could likely apply to the LMAPF_R problem with relatively little modification, we adopt the straight-line formulation. Let $\mathit{from}^v(m) = v$ and $\mathit{to}^v(m) = v'$ denote the start and end vertices, and $\mathit{from}^t(m) = t$ and $\mathit{to}^t(m) = t + \mathcal{W}(e)$ denote the start and end times.

A *wait* action $w = \langle v, t_1, t_2 \rangle \in \mathcal{V} \times \mathbb{R} \times \mathbb{R}$, with $t_1 < t_2$, means that an agent remains idle at $\mathcal{M}(v)$ from time t_1 until t_2. For such a wait action, let $\mathit{from}^v(w) = \mathit{to}^v(w) = v$, $\mathit{from}^t(w) = t_1$, and $\mathit{to}^t(w) = t_2$.

Each agent $i \in [1, .., N]$ follows a *plan* $\pi_i = \langle a_1^i, \ldots, a_n^i \rangle$ consisting of actions, with each action being either a move or wait action. A plan π_i is *valid* if it



(a) A move-move conflict $\langle m^i, m^j \rangle$, with both agents moving.



(b) A move-wait conflict $\langle m^i, w^j \rangle$, with agent i moving while agent j waits.

Figure 4.1: Illustration of a move-move and a move-wait conflict between two agents i and j. Each agent's position at the start and end time of the collision is shown by a dashed circle.

starts at i's starting vertex at time 0,

$$from^v(a_1^i) = s(i) \quad \wedge \quad from^t(a_1^i) = 0,$$

and each subsequent action is connected in space and time to the previous,

$$from^v(a_k^i) = to^v(a_{k-1}^i) \quad \wedge \quad from^t(a_k^i) = to^t(a_{k-1}^i), \quad \forall k = 2, .., n.$$

Each agent is associated with a hypervolume in $\mathcal{M}$, describing the physical space that its body occupies. LMAPF$_\text{R}$ commonly considers 2D $\mathcal{M}$ space where agent shapes are described by areas. For the remainder of the text, we remain consistent with MAPF literature by referring to agent shapes as *volumes* [15]. We say that the agents are *volumetric*, sometimes referred to as *large agents* [93]. Two agents *collide* when their volumes intersect; if the volumes of agents i and j intersect while executing the actions a^i and a^j, respectively, then $\langle a^i, a^j \rangle$ is a *conflict*. A conflict $\langle m^i, m^j \rangle$ containing two move actions is a *move-move* conflict, and a conflict $\langle m^i, w^j \rangle$ containing a move and a wait action is a *move-wait* conflict. Figure 4.1 illustrates these two conflict types. Two plans π_i and π_j conflict if there exists a conflict in $\pi_i \times \pi_j$. A *joint plan* $\Pi = \{\pi_1, \ldots, \pi_N\}$ contains a plan for each agent. A *solution* is a joint plan containing only valid plans and no two of its plans conflict with each other.

The main difference with discrete-time LMAPF is that in LMAPF$_\text{R}$ agents can execute actions at any real-valued time instead of at every timestep, and tasks can be released to the system at any real-valued time. There are a number of consequences that follow from this. In the discrete-time formulation, the number of unique configurations which the system may be in is countable; at any timestep, each agent is located at a unique vertex. On the other

hand, the set of possible configurations in the LMAPF_R problem is dense and uncountable; agents can be located at any vertex or on any part of an edge, since they may begin traversing edges with any real-valued length at any real-valued time. Consequently, the search space for the LMAPF_R problem is dense, therefore requiring more sophisticated search routines than in the discrete-time variant.

Another notable difference when going from LMAPF to LMAPF_R is that volumetric agents must be considered. In LMAPF, the physical shape of the agents do not matter since conflicts only occur when agents occupy the same vertex or traverse the same edge in opposite directions at the same time. Based on this formulation, the graph does not need to encode spatial information about the environment; only the graph's topology is used when computing plans and detecting collisions, not the physical positions that each vertex and edge correspond to in the real world. Moving to continuous time, if these simplified definitions of conflicts were to be used, then agents can move arbitrarily close to each other without a conflict occurring. For instance, an agent can leave a vertex at some arbitrarily small time before another agent arrives there, thus avoiding a collision despite the agents being arbitrarily close to each other. A solution including such an event is practically useless, since robots in the real world take space and would therefore collide. Thus, the LMAPF_R problem must consider volumetric agents if the solutions are they have practical value. An agent's volume in metric space $\mathcal{M}$ describes its physical shape, allowing for a natural definition of a conflict: two agents collide when their volumes intersect. Therefore, two agents' plans conflict if the agents collide at any time while following these plans. With this definition of a conflict, agents can no longer move arbitrarily close to each other, leading to solutions that at least consider the physical size of agents.

This definition of a conflict between two volumetric agents requires the explicit consideration of the agents' positions in $\mathcal{M}$. Therefore, the graph must now encode spatial information about the environment. Hence, this is the reason why every vertex $v \in \mathcal{V}$ maps to a position in $\mathcal{M}$, and a traversal along an edge $(v, v') \in \mathcal{E}$ represents a trajectory from $\mathcal{M}(v)$ to $\mathcal{M}(v')$.

Figure 4.2 illustrates an LMAPF_R problem with three agents on a graph with non-unit-length edges. Since agents can begin and finish traversing edges at any real-valued time, and tasks enter the system at real-valued times, the agents' movements are not synchronized to start and end at regular timesteps.

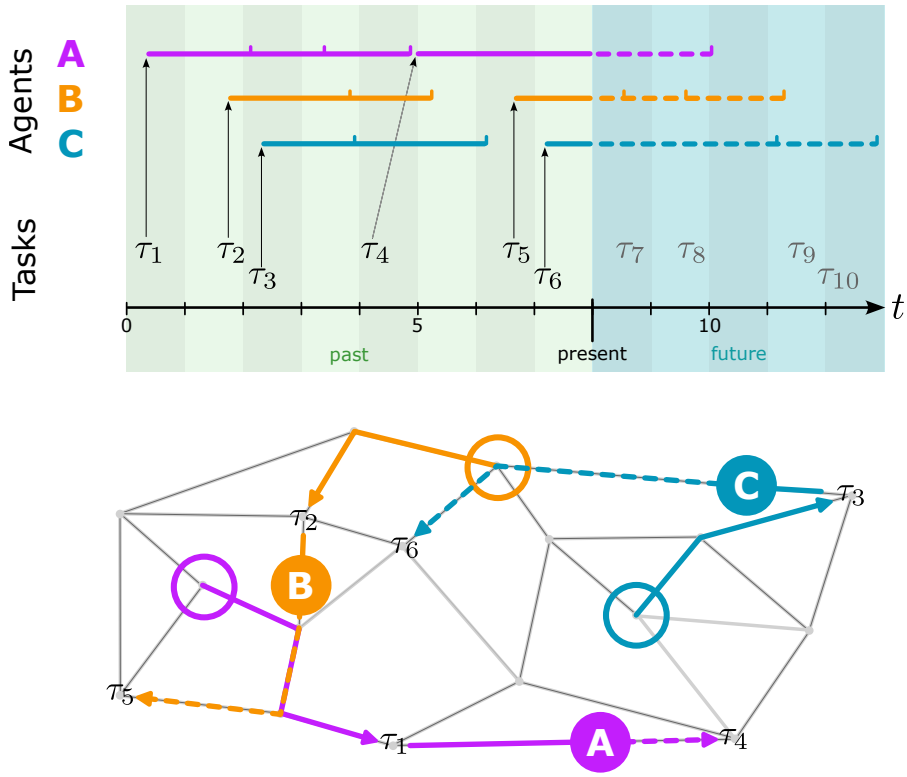


Figure 4.2: An LMAPF$_R$ problem with three agents (A, B, C) and the current system time set to $t = 8.0$. **Top:** A timeline of agent movements and task releases. Completed edge traversals are marked on each timeline, and task assignments are denoted by arrows. The tasks in the future, such as τ_7 and later, are unknown at the current time and therefore cannot be acted on. **Bottom:** An illustration of the agents in the environment as they attend to tasks, showing at time $t = 8.0$ their starting positions (rings), current positions (circles), completed movements (full lines), and planned future movements (dashed lines).

Agents are also volumetric, unlike in LMAPF, such that movements must be planned to avoid agent volumes from intersecting.

To our knowledge, Paper B is the first to address the LMAPF_R problem. We have since discovered one other related work [94] that has emerged since Paper B's dissemination, which we briefly introduce here. This work considers a variant of LMAPF_R with certain restrictions and extensions. The approach restricts graphs to grid-networks where circular agents have diameters equal to the unit-length of edges. However, it extends the LMAPF_R problem by incorporating kinematic constraints, modeling agents that move with constant acceleration, maximum speed, and constant rotational speed. Additionally, the work addresses tasks containing both pickup and delivery components. That is, this is a form of *continuous-time MAPD*. The method employs a Dijkstra-like search strategy that incrementally finds paths for agents while avoiding collisions with previously scheduled routes. In other words, both Paper B and [94] share a common approach by applying strategies related to prioritized planning and *Safe Interval Path Planning* [33] (described in the next section).

4.2 Challenges with Extending LMAPF to Continuous Time

The four main challenges encountered when extending discrete-time LMAPF to continuous-time are discussed here. The first two challenges are that actions now occur asynchronously between agents, and continuous time causes the search space to be dense and uncountable. The third challenge is that conflict resolution becomes a more complex problem to solve — as a consequence of agents being volumetric — since a moving agent can cause conflicts with agents in more ways than when only considering collisions on the same edge or at the same vertex. This expanded range of possible conflict occurrences hinders the use of many existing discrete-time methods. Lastly, collision detection between two volumetric agents is computationally demanding. Lifelong problems, with the known information evolving over time, inherently model real-time systems where runtime is limited. Thus, the computational complexity of collision detection must be addressed to achieve scalability.

Asynchronous Actions in a Dense Search Space

Agents move asynchronously in LMAPF_R. This fundamental departure from discrete-time planning introduces two interconnected challenges that many existing discrete methods cannot address, as will be discussed here.

The first challenge relates to the coordination of agent actions. Discrete-time MAPF methods, such as PIBT [72] and RHCR [14], rely on synchronization points where all agents' states are considered simultaneously. For instance, PIBT assumes that every agent is available to execute the next action at the current timestep. From that state, it plans the next action for each agent sequentially. However, when edge traversals take arbitrary durations, no natural synchronization point exists — agents complete actions at different times. This problem persists with larger time windows, as in RHCR which applies windowed planning. While agents might start a planning window synchronously, their actions complete at different times, leaving them unsynchronized at the window's end. One might artificially impose synchronization by adding wait actions so that all agents finish simultaneously, but such unnecessary delays degrade solution quality.

While PIBT and RHCR are two examples of methods that plan within a time window, TSWAP [84] instead plans the next actions based purely on the spacial configuration. However, this approach still fails for the same fundamental reason — it requires a well-defined "current state" where all agents are ready to execute the next action. With asynchronous agents completing actions at different times, there is no single spatial configuration to plan from.

The second challenge relates to the configuration space in continuous time being dense. In discrete time, every action takes exactly one timestep to execute. Thus, the set of all possible configurations at any timestep is countable. On the other hand, the configuration space in continuous time is dense since at any given time, agents need not only be located at some vertex but can also be located at any part of an edge. How should planners navigate such a dense configuration space? Fixing all wait actions to a constant duration and chaining them for longer delays, for instance, simply reintroduces time discretization, abandoning the continuous-time formulation.

To address both challenges — asynchronous actions and dense configuration spaces — many continuous-time MAPF approaches use *Safe Interval Path Planning* (SIPP) [33], a method for planning the continuous-time movement of a single agent in an environment with dynamic obstacles. In broad terms,

SIPP finds a single-agent shortest path in a graph while treating already-planned agents as moving obstacles that must be avoided. SIPP uses *safe intervals* — a time interval during which an agent may occupy a vertex or begin traversing an edge without collision.

Let the movements of a number of dynamic obstacles on a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}, \mathcal{W} \rangle$ be known. For each vertex $v \in \mathcal{V}$, we let the set $\mathbf{t}_v^s \subseteq \mathbb{R}$ contain every time during which it is safe for the agent to occupy v without colliding with any of the dynamic obstacles. For instance, if an obstacle passes through v during some time interval, then all times in that time interval are *unsafe* and therefore not included in $\mathbf{t}_v^s$. Likewise for all edges $e \in \mathcal{E}$, let $\mathbf{t}_e^s$ contain every time when it is safe for the agent to begin traversing e without colliding with any of the dynamic obstacles. Provided that the agent only occupies vertices and begins traversing edges during safe intervals, it will not collide with any of the dynamic obstacles.

In the SIPP search, a *state* is defined by a tuple $\langle v, I \rangle \in \mathcal{V} \times \mathbb{R}^2$ containing a vertex v and a maximally connected safe interval $I \subseteq \mathbf{t}_v^s$. Suppose that the agent arrives at vertex v at time t, this corresponds to the state $\langle v, I^v \rangle$ where $t \in I^v$. From this state, we now aim to find all times when is it is safe to traverse a connected edge $e = (v, v') \in \mathcal{E}$.

Since the agent arrived at v at t, the agent can only traverse an edge to move away from v at some time $\geqslant t$. Additionally, letting $I^v = [t_1^v, t_2^v)$, the agent must leave v before t_2^v else a collision will occur. Furthermore, the agent can only traverse an edge e at some safe time in $\mathbf{t}_e^s$, and arrive at the destination vertex v' at some safe time in $\mathbf{t}_{v'}^s$. Thus, the set of all times when the agent can safely traverse e is

$$\mathbf{t}_{v,e}^s = [t, t_2^v) \cap \mathbf{t}_e^s \cap \mathbf{t}_{v'-\mathcal{W}(e)}^s$$

where $\mathbf{t}_{v'-\mathcal{W}(e)}^s = \{t' - \mathcal{W}(e) \mid t' \in \mathbf{t}_{v'}^s\}$ to compensate for e taking $\mathcal{W}(e)$ time to traverse. For every maximally connected interval $[t_1, t_2) \in \mathbf{t}_{v,e}^s$, a move action $\langle e, t_1 \rangle$ connects the state $\langle s, I^v \rangle$ to the state $\langle v', I^{v'} \rangle$ where $t_1 + \mathcal{W}(e) \in I^{v'}$. Notably, SIPP always selects the earliest time t_1 within a safe interval to traverse the edge e.

To find a shortest-duration plan π to move the agent from a starting vertex $s \in \mathcal{V}$ at time t_0 to a goal vertex $g \in \mathcal{V}$, SIPP performs a graph search (such as Dijkstra's [29] or A* [30]) from the state $\langle s, I^v \rangle$ with $t_0 \in I^v$ to any state $\langle g, \cdot \rangle$. The resulting path contains the move and wait actions constituting π.

The figure on the cover of this thesis provides a visualization of the results from applying SIPP. For the agent at the center of the graph, SIPP is used to find the earliest time when the agent can arrive and remain indefinitely at every vertex in the graph, while avoiding collisions with the other three agents following their respective plans. Each vertex's color represents the computed time.

SIPP is used in many continuous-time MAPF method, such as Papers B and C, *Continuous-time CBS* (CCBS) [16], and *Prioritized SIPP* (PSIPP) [95]. SIPP allows for the discretization of the continuous-time domain without enforcing fixed timesteps, thereby offering an approach to exploring the dense continuous-time configuration space in a structured and non-constraining way.

Windowed planning is particularly suitable for lifelong problems since planning with a limited depth into the future allows for a higher degree of adaption to new incoming information. To our knowledge, however, the handling of asynchronous actions by way of windowed planning is relatively unexplored in the existing continuous-time MAPF literature. Both CCBS and PSIPP solve the one-shot continuous-time MAPF problem — CCBS is based on CBS [60] and PSIPP plans each agent's complete path by prioritized planning using SIPP. Thus, CCBS plans all agents jointly (intractable in real-time settings with many agents) and PSIPP computes entire agent plans (less suitable for constantly changing environments). Therefore, neither of these methods address the online planning of LMAPF_{R} where asynchronous agent plans are extended in real-time.

Resolving Collisions between Volumetric Agents

When agents are modeled as points on a graph, a move action creates conflicts with only a limited set of other agents — specifically, agents at the destination vertex or traversing the same edge in the opposite direction. However, the sweep volume generated during movement by volumetric agents can collide with any agent within the spatial vicinity. This expanded range of conflicts fundamentally changes the nature of collision resolution.

Common conflict resolution strategies from discrete-time sizeless-agent methods cannot directly account for this increased potential for conflicts. Consider PIBT: when an agent evaluates a potential edge traversal, it checks only whether another agent will occupy the destination vertex at the next timestep. If that vertex is occupied, the agent seeks an alternative; if an agent is cur-

rently there but unplanned, PIBT attempts to move that agent away. With volumetric agents, however, conflicts can occur not only at the destination vertex but also with agents at nearby vertices or on nearby edges. The localized conflict-checking assumption that makes PIBT efficient simply does not hold.

Existing MAPF solvers for volumetric agents employ different strategies, each with distinct limitations for online planning. CBS based methods, such as CCBS and *Multi-Constraint CBS* [93], progressively add constraints to resolve conflicts as they are detected, until no conflicts occur. While these methods are complete and optimal, they suffer from scalability issues that make them less suitable for online planning with limited computation time.

PSIPP combines prioritized planning with SIPP, sequentially planning each agent's complete path while treating already-planned agents as dynamic obstacles. This approach is well-suited for one-shot MAPF, where all agents must reach their goals and computational resources are sufficient to plan complete paths. However, in online settings such as LMAPF_R, planning each agent all the way to its goal creates inefficiencies for two reasons. First, computation time is limited in online problems. PSIPP's sequential approach risks exhausting the computational budget before all agents have been planned, leaving some agents without plans entirely. It is more effective to allocate computation power such that *all* agents receive plans extending into the near future, rather than *some* agents receiving complete plans while others remain unplanned. Second, LMAPF_R operates in a dynamic environment where available information constantly updates as new tasks enter the system. Planning paths far into the future forces a dilemma: either abandon already-computed plans to incorporate new information (wasting computational effort), or delay the system's response to new information until the precomputed plans are executed (reducing responsiveness). By planning only into the near future and not beyond that, the system can more readily adapt to incoming information.

Therefore, online settings require a fundamentally different planning paradigm from existing continuous-time methods — one that extends plans for all agents incrementally and uniformly into the future, rather than planning agents sequentially to completion.

Real-Time Collision Detection between Volumetric Agents

With volumetric agents, collision detection becomes significantly more computationally intensive [96]. Unlike point-based collision detection, which simply checks whether two agents occupy the same vertex or traverse the same edge, volumetric collision detection requires checking for overlapping volumes. Detecting intersections between arbitrary volumes is a computationally expensive operation, making it unsuitable for real-time systems where computation time is strictly limited.

This computational challenge can be addressed by shifting the burden from runtime to preprocessing [95, 96]. Rather than computing volume intersections in real-time, we can leverage beforehand knowledge to pre-compute collision information which can then be used in real-time to compute safe intervals for SIPP, thereby enabling faster runtime performance.

Consider an agent executing a move action m starting at time $t = 0$. For every other possible move action m', we can pre-compute the time interval $I_{m,m'}$ such that if another agent begins executing m' at some time $t \in I_{m,m'}$, a collision will occur. Similarly, for every vertex $v \in \mathcal{V}$, we can pre-compute $I_{m,v}$ representing the time interval during which another agent waiting at v would collide with the first agent executing m. By computing all such intervals for every move-move and move-vertex pair, we obtain a complete lookup structure for collision detection during runtime. When two agents execute actions at specific times — for instance, agent 1 executes move action m at time t and agent 2 executes m' at time t' — collision detection reduces to a simple lookup and interval check: retrieve $I_{m,m'}$ and check whether $t' - t \in I_{m,m'}$. If so, a collision occurs. The memory and computation complexity of such a lookup table is $\mathcal{O}\left((m + |\mathcal{V}|)^2\right)$ with m being the number of move actions. However, in most environments the majority of move-move and move-vertex pairs are too spatially distant compared to the size of the agents for a collision to occur. This is useful for both memory and computation; no value needs to be stored for such pairs, and using standard graphical collision detection methods [97] can be used to reduce the number of pairs that are evaluated. Therefore, this method is likely even more viable than what the quadratic complexity from above suggests.

The total computation and memory requirements of this approach are inflated by fleet heterogeneity. Intersection intervals are based on agent volumes; if all agents share the same shape, a single lookup table suffices. However, for

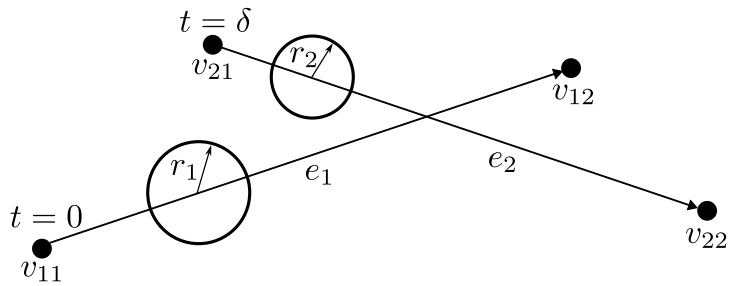


Figure 4.3: Two agents respectively traversing $e_1 = (v_{11}, v_{12})$ at time $t = 0$ and $e_2 = (v_{21}, v_{22})$ at time $t = \delta$. Both agents are circular, respectively with radii r_1 and r_2.

a heterogeneous fleet with n distinct agent types (each with a unique shape), collision intervals must be computed for every type pair, requiring n^2 lookup tables. With each lookup table's memory and computation complexity being $\mathcal{O}\left((m + |\mathcal{V}|)^2\right)$, if every one of the N agents have a unique volume then the upper-bound on the total complexity is $\mathcal{O}\left((m + |\mathcal{V}|)^2 \cdot N^2\right)$. Therefore, although this method is most efficient for homogeneous fleets, the memory complexity is bounded by a polynomial in the size of the graph and number of agents and is therefore likely a viable method to achieve real-time performance. It is also possible in practice to approximate all agents with the same volume, such as a sufficiently large circle, ensuring that all agents' true volumes are contained within the approximation.

For specific agent volumes, efficient algorithms exist for computing collision intervals. In [95], a method is proposed for circular agents traveling along straight edges at constant speed, with computation complexity $\mathcal{O}\left((|\mathcal{E}| + M) \log |\mathcal{E}|\right)$ where M is the number of edge pairs close enough for potential collisions. Their approach builds on the Bentley-Ottmann algorithm [98] for detecting line segment intersections, thereby avoiding computations for every $|\mathcal{E}|^2$ edge pairs. For any two intersecting edges in $\mathcal{E}$, the corresponding collision interval can be computed analytically [96]:

Suppose that two agents each traverse a respective edge, shown in Figure 4.3: agent 1 traverses $e_1 = (v_{11}, v_{12})$ starting at time $t = 0$ and agent 2 traverses $e_2 = (v_{21}, v_{22})$ starting at time $t = \delta$, both traveling at constant speed s. Let both agents be circular with radii r_1 and r_2, respectively. The

agents' velocity vectors are

$$V_1 = \frac{v_{12} - v_{11}}{|v_{12} - v_{11}|} s \quad \text{and} \quad V_2 = \frac{v_{22} - v_{21}}{|v_{22} - v_{21}|} s.$$

The agents' positions over time are therefore

$$P_1(t) = v_{11} + V_1 t \quad \text{and} \quad P_2(t) = v_{21} + V_2(t - \delta).$$

Observe that these position functions do not consider that the edges have finite length; $P_1(t)$ is only valid for $t \in \left[0, \frac{|e_1|}{s}\right]$ and $P_2(t)$ is only valid for $t \in \left[\delta, \delta + \frac{|e_2|}{s}\right]$. This will be addressed at a later stage. The squared distance between the agents' boundaries over time is

$$\begin{aligned}
d^2(t, \delta) &= |P_1(t) - P_2(t)| - (r_1 + r_2)^2 \\
&= (P_1(t) - P_2(t))^2 - (r_1 + r_2)^2 \\
&= (v_{11} + V_1 t - v_{21} - V_2(t - \delta))^2 - (r_1 + r_2)^2 \\
&= (\underbrace{v_{11} - v_{21}}_{\bar{v}_1} + \underbrace{(V_1 - V_2)}_{\bar{V}} t + V_2 \delta)^2 - (r_1 + r_2)^2 \\
&= \bar{V}^2 t^2 + 2\bar{V} V_2 t\delta + V_2^2 \delta^2 + 2\bar{v}_1 \bar{V} t + 2\bar{v}_1 V_2 \delta + \bar{v}_1^2 - (r_1 - r_2)^2 \\
&= At^2 + Bt\delta + C\delta^2 + Dt + E\delta + F
\end{aligned}$$

where

$$\begin{aligned}
A &= (V_1 - V_2)^2, & B &= 2(V_1 - V_2)V_2, \\
C &= V_2^2, & D &= 2(v_{11} - v_{21})(V_1 - V_2), \\
E &= 2V_2(v_{11} - vv_{21}), & F &= (v_{11} - v_{21})^2 - (r_1 + r_2)^2.
\end{aligned}$$

This forms a conic section in the (t, δ)-plane. The boundary at $d^2(t, \delta) = 0$ separates the space implying that a collision occurs from the space where no collision occurs, shown in Figure 4.4 as an ellipse. A point (t', δ') within the $d^2(t, \delta) = 0$ ellipse implies that if agent 1 begins traversing its edge at $t = 0$ and agent 2 begins traversing its edge at $t = \delta'$, then at time $t = t'$ the agents' volumes will be intersecting. Agent 1 traverses e_1 only during the time interval $\left[0, \frac{|e_1|}{s}\right]$ and agent 2 traverses e_2 only during $\left[\delta, \delta + \frac{|e_2|}{s}\right]$. Therefore, a collision can only occur during these times, shown as a parallelogram in

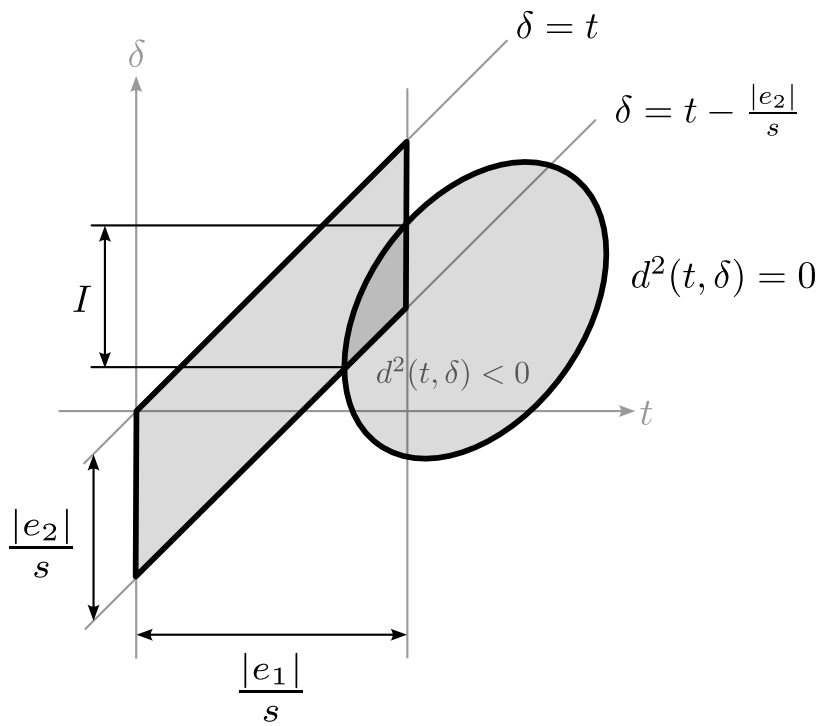


Figure 4.4: In (t, δ)-plane, all interior points of the ellipse $d^2(t, \delta) = 0$ imply that the agents collide. The parallelogram contains all points when both agents are on their respective edges. Thus, the intersection between the ellipse and parallelogram contain points when a collision occurs while the agents are traversing their respective edges. The minimum and maximum δ-values in this region define the collision interval I.

Figure 4.4. The space in the (t, δ)-plane within the interior of both the parallelogram and the ellipse contains all (t, δ)-points where a collision is both possible and occurs. Thus, the minimum and maximum δ-value found among these points defines the collision interval — the time interval during which if agent 2 traverses e_2, a collision will occur with agent 1.

This specific method for calculating collision intervals applies only to circular agents, traveling along edges at constant speed. However, the same general framework can be used for arbitrarily shaped agents and move actions following arbitrary trajectories: for each move-move and move-wait pair, compute the time intervals (using any appropriate method) during which the agents' volumes intersect. For circular agents traveling in straight lines with non-negative speed, there will exist only one interval. However, in the general case of non-straight-line move actions, there may be several disjointed intersection intervals. In such cases, all intervals must be stored.

4.3 Updated Experiments for Paper B

After the publication of Paper B which introduces the *Continuous-time Prioritized Lifelong Planner* (CPLP), we discovered an error in the implementation that lead to results indicating better performance than what would be possible in practice. This section is dedicated to highlighting these implementation errors and detailing their corrections, performing new experiments, and discussing the results.

Implementation Corrections

The computation and execution timing of CPLP is illustrated in Figure 4.5. At some time t, CPLP is given a computation budget Δt to compute agent plans starting at time $t_{plan} = t + \Delta t$. For a single prioritized agent-task-pair, an entire plan from the agent's current location to the task location is computed. For all other agents, plans extending until at least $t_{plan} + \bar{t}$ are computed to move the agents *toward*, but not necessarily to, their task locations. That is, $\bar{t}$ is the approximate size of the planning window. Due to edges in the environment graph having any real-valued weight, the agents' plans do not necessarily end at the same time. It is also possible in some cases that an agent's plan does not extend all the way until at least $t_{plan} + \bar{t}$. This can

happen if, for instance, there are no more tasks to assign to agents. However, we can assume that the movements of *most* agents are planned until at least $t_{plan} + \bar{t}$. CPLP returns the newly computed plans, and a time t_{next} denoting the start time of the next plans to compute. The time t_{next} is computed as the maximum of $t_{plan} + \bar{t}$ and t_{min}, with t_{min} being the minimum end-time over all newly computed agent plans.

The time that CPLP actually needs to compute the plans, t_{comp}, should ideally be less than the time available, $t_{comp} \leqslant \Delta t$. If $t_{comp} > \Delta t$, then when CPLP returns plans, the time when those plans are supposed to begin would have already passed, thereby invalidating them. On the other hand, if $t_{comp} \leqslant \Delta t$, then the plans are valid and start at some time after they have been computed. CPLP can then begin computing new plans at time $t + t_{comp}$. In the original CPLP implementation, no guards ensured that CPLP only begins computing new plans at the earliest from time $t + t_{comp}$. This is an error pertaining to the simulation of using CPLP in practice. Instead, new plans were computed from time $t_{next} - \Delta t$, without ensuring that $t + t_{comp} \leqslant t_{next} - \Delta t$. All experiments in Paper B use $\bar{t} = 1$ and $\Delta t = \max(N^{1.25}, 500)$ ms. Assuming that $t_{next} \approx \bar{t}$, the increasing value of Δt with agent count N almost certainly leads to $t + t_{comp} > t_{next} - \Delta t$ for higher N.

Additionally, since $\bar{t} = 1$ and $\Delta t = \max(N^{1.25}, 500)$ ms in all experiments, it holds that $\Delta t > \bar{t}$ for $N > 1000^{1/1.25} \approx 251$. This would then mean that — assuming CPLP is ensured to only begin computing new plans after time $t + t_{comp}$ — the available time to compute new plans is longer than the duration of those plans, meaning that constant engagement of agents to complete tasks is not ensured. Instead, agents would occasionally have to wait idly while new plans are being computed. To achieve continuous, never-ending movement so that agents are always engaged with uncompleted tasks, we must ensure that $\Delta t \leqslant \bar{t}$.

Experimental Setup and Discussions

The experimental setup is the same as in Paper B, however with some differences. Most notably, CPLP's computation time t_{comp} is used to ensure that it is called earliest at $t + t_{comp}$, better matching how it would be used in practice. Additionally, we set $\bar{t} = \Delta t$. We ran 10 instances for each $N \in \{10, 25, 50, 100, 200, \ldots, 1000\}$, with $\rho = \{5, 10, 15\}$. A different schedule for Δt is used to better match the observed computation times with the

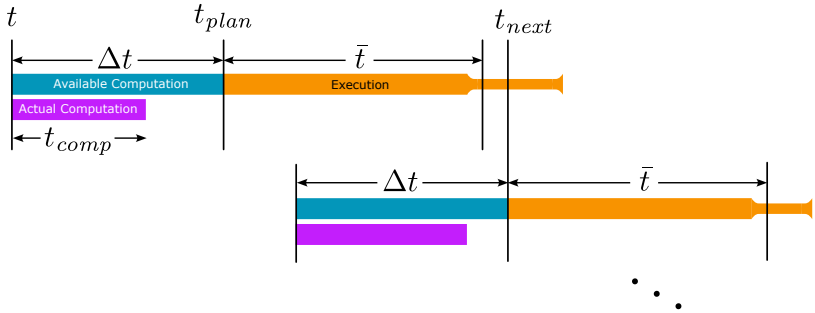


Figure 4.5: CPLP's computation and execution timing. At t, CPLP is given Δt time to compute plans starting at $t_{plan} = t + \Delta t$. CPLP also computes the time t_{next} when new plans should begin.

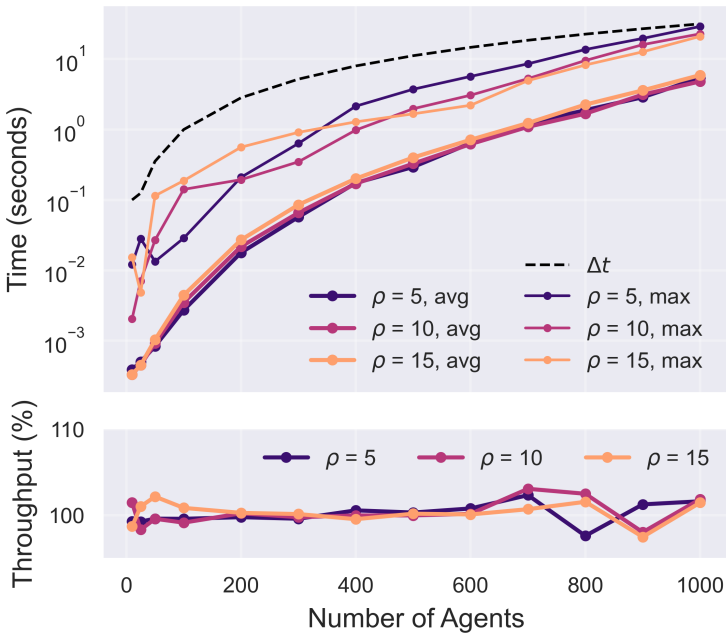


Figure 4.6: Benchmark results. **Top**: CPLP's average and maximum computation time (t_{comp}) for each agent count and number of vertices per agent (ρ). The available computation time Δt is also shown. **Bottom**: the average throughput as a percentage of the incoming tasks.

longer planning horizon $\bar{t}$: $\Delta t = \max(N^{1.5}, 100)$ ms. All experiments are run on a 2025 Mac Studio, 16-core M4 Max CPU, 64 GB RAM, macOS Sequoia (15.3). Besides these changes, all other parameters remain the same as in Paper B.

The results are presented in Figure 4.6. The top part of the figure shows the average and maximum computation time per call to CPLP, along with the available computation time Δt. Since the all computation times remain below Δt, this indicates that a real-world deployment of CPLP is viable for up to 1000 agents. The lower part of the figure shows the achieved throughput which remains around 100 % (with some deviation due to randomness in the task release times), implying that CPLP is able to plan agents sufficiently well to meet the specific task demand tested here.

These results suggest that CPLP is viable for real-world deployment, particularly for the agent counts often seen in industrial settings, which we believe typically remain in the hundreds, if not less. We did not test for agent counts beyond 1000. However, if the same trend continues then the computation time will likely exceed Δt when using the same schedule, $\Delta t = \max(N^{1.5}, 100)$ ms. Higher values for Δt could be used, but that would also mean that the planning horizon $\bar{t}$ would also increase, leading to higher computation times as well. Future work could look into quantifying the relationship between Δt and the observed maximum computation time, to obtain a function specifying what value of Δt would confidently result in CPLP's computation time not exceeding it. Further experimentation could also look into increasing the rate of released tasks — currently at 1 tasks per agent every 20 seconds — to find CPLP's throughput limit.

4.4 Answering Research Question 2

To answer **RQ2**— *What is required to extend discrete-time Lifelong MAPF to continuous time while maintaining computational tractability?* — four key requirements emerge.

First, handling asynchronous agent actions. When actions take arbitrary durations, agents no longer move in lockstep, eliminating the synchronized timesteps that many discrete-time methods rely on. Typical formulations of windowed approaches require synchronized start states, and even when such states exist, this cannot be maintained since agents become desynchronized

at the window end as actions generally complete at different times. This necessitates planning methods that do not depend on global synchronization points.

Second, discretizing the dense search space. With actions executable at any real-valued time, the search space becomes dense. SIPP provides a solution by discretizing continuous time into safe intervals, enabling systematic search through this otherwise intractable space. However, SIPP is inherently a single-agent planning algorithm, treating other agents as dynamic obstacles. Therefore, SIPP must be combined with multi-agent coordination strategies (typically prioritized planning) to achieve tractability.

Third, managing a larger range of possible conflicts between volumetric agents. When agents have volume, their sweep volumes as the follow their plans can collide with any agent in the spacial vicinity, not just those on adjacent vertices or edges. This unbounded conflict space invalidates methods like PIBT and TSWAP that rely on localized conflict checking. Existing continuous-time methods using prioritized planning typically compute entire plans sequentially — an approach poorly suited for lifelong settings where computing complete plans risks exhausting available computational resources before all agents have been attended to, and where constantly updating information renders plans extending far into the future potentially suboptimal. Tractability requires planning strategies that extend all agents' plans incrementally, rather than sequentially to completion.

Fourth, efficient collision detection for volumetric agents. Runtime collision detection between arbitrary volumes is computationally prohibitive. Tractability requires offloading computation to preprocessing: by pre-computing collision intervals for all action pairs and storing them in lookup tables, runtime collision detection reduces to simple interval lookups. This approach works best for homogeneous fleets with limited shape variance.

Our approach in Paper B addresses these requirements by planning agents incrementally beyond time windows using prioritized planning, employing a form of truncated SIPP to navigate the dense search space, and assuming circular agents that travel at constant speed to enable pre-computed collision intervals.

CHAPTER 5

Solving Continuous-Time MAPF for Optimal Solutions in Finite Time

The MAPF_R problem extends classical MAPF to continuous-time, allowing actions to begin at any real-valued time, edge traversals to occur along any trajectory in space-time, and agents to have arbitrary volumes. While methods exist for solving simplified versions of the problem or for finding suboptimal solutions, only one method was thought to guarantee finding optimal solutions to the full MAPF_R problem: *Continuous-time Conflict-Based Search* (CCBS) [16]. Numerous continuation works have been published since CCBS's introduction, such as [99] for performance enhancements, *T-Robust CCBS* [100] for solutions that are robust to delays, [101] for *Any-Angle MAPF*, and *Continuous-time Prioritized Lifelong Planner* (Paper B) for LMAPF_R. Recent finding [32], however, suggest that CCBS's theoretical description does not guarantee termination, while its publicly available implementation[1] does not guarantee the return of an optimal solution. This reopens an important question that the existing literature does not answer: **RQ3** — *How can the continuous-time MAPF problem be solved in finite time with guaranteed so-*

[1] https://github.com/PathPlanning/Continuous-CBS

lution optimality? This chapter establishes the theoretical foundations for understanding the shortcomings of CCBS and the resolution provided in Paper C, thereby answering **RQ3**.

5.1 MAPF in Continuous Time

The MAPF_R problem introduced here and in [16] combines several components from the classical MAPF and the LMAPF_R formulations introduced in previous chapters, while incorporating further generalizations. In this section, we formally define the MAPF_R problem, introduce a few relevant methods, and provide discussions surrounding aspects of the MAPF_R search space in the context of **RQ3**.

A Formal Definition of MAPF_R

A MAPF_R problem [16] can be defined by a tuple

$$\langle \mathcal{G}, \mathcal{M}, N, s, g, \mathcal{A} \rangle$$

where $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ is a connected and directed graph with vertices $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$; $\mathcal{M}$ is a metric space, with $\mathcal{M}(v)$ mapping a vertex $v \in \mathcal{V}$ to a position in $\mathcal{M}$; each of the N agents start at a unique vertex by $s : \{1, .., N\} \to \mathcal{V}$ and are assigned a unique goal by $g : \{1, .., N\} \to \mathcal{V}$; and the set $\mathcal{A}$ contains all *move actions*. The notation used here differs slightly from that in [16] and Paper C, however, the underlying problem remains the same.

Move actions in this formulation take on a more general form than in the previously introduced MAPF formulations. A move action $m = \langle m_\varphi, m_D \rangle \in \mathcal{A}$ specifies one of possibly several ways to traverse an edge $e = \langle v, v' \rangle \in \mathcal{E}$. The *motion function* $m_\varphi : [0, m_D] \to \mathcal{M}$ defines a connected trajectory in $\mathcal{M}$ and time, with duration $m_D \in \mathbb{R}^+$. Such a trajectory can take any path through $\mathcal{M}$ and at any variable speed, so long as it starts at $\mathcal{M}(v)$ and ends at $\mathcal{M}(v')$. Let $\mathit{from}^v(m) = v$ and $\mathit{to}^v(m) = v'$. There exists at least one move action associated with each edge in $\mathcal{E}$. Figure 5.1 illustrates an edge with three associated move actions (m, m', and m'').

A *wait action* $w = \langle w_\varphi, w_D \rangle$ (not included in $\mathcal{A}$) takes the same form as a move action, however, with two noticeable differences: the motion function $w_\varphi : [0, w_D] \to \mathcal{M}(v)$ maps only to the position of single vertex $v \in \mathcal{V}$, and

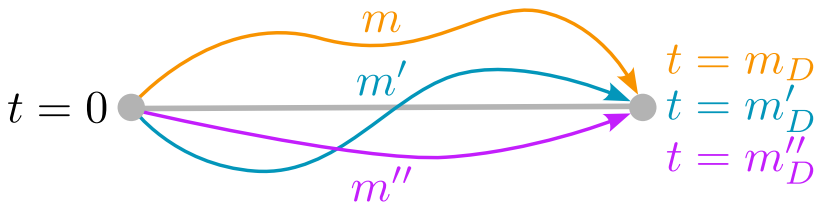


Figure 5.1: An illustration of an edge with three associated move actions, m, m', and m''. Each move action takes a certain duration (m_D, m'_D, m''_D) to complete by following a certain trajectory (m_φ, m''_φ, m''_φ).

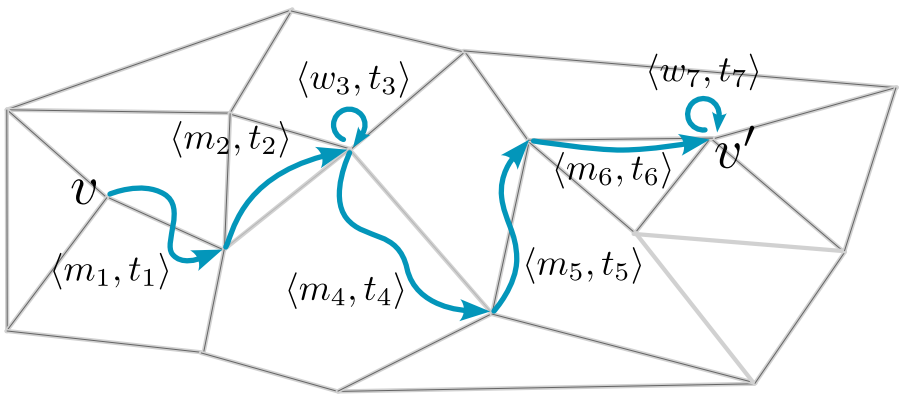


Figure 5.2: A visualization of a plan, consisting of five timed move actions and two timed wait actions. For this plan to be valid for an agent, v and v' must respectively be the agent's start and goal vertices, every consecutive timed action must be connected to the previous in space and time, and the final wait action w_7 must have infinite duration, $w_{7,D} = \infty$.

the duration $w_D \in \mathbb{R} \cup \{\infty\}$ may be infinite (implying that the agent waits at v indefinitely). Let $\mathit{from}^v(w) = \mathit{to}^v(w) = v$.

Agents execute *timed actions*. A timed action $\langle a, t \rangle$ (with a either a move or wait action) specifies not only which action is executed, but also when. Thus, the position of an agent executing the timed action $\langle a = \langle a_\varphi, a_D \rangle, t \rangle$ is $a_\varphi(t' - t)$ for $t' \in [t, t + a_D]$. A plan π_i for an agent i is a sequence of timed actions,

$$\pi_i = \langle \langle a_1^i, t_1^i \rangle, \langle a_2^i, t_2^i \rangle, \ldots, \langle a_n^i, t_n^i \rangle \rangle,$$

where n is individual for each agent and varies over solutions. For π_i to be *valid*, it must start and end at i's start and goal vertices,

$$\mathit{from}^v(a_1^i) = s(i) \quad \wedge \quad \mathit{to}^v(a_n^i) = g(i),$$

every subsequent timed action must be connected to the previous in time and space,

$$\mathit{from}^v(a_k^i) = \mathit{to}^v(a_{k-1}^i) \quad \wedge \quad t_k^i = t_{k-1}^i + a_{k-1,D}^i, \qquad \forall k = 2, .., n,$$

and the final action a_n^i must be a wait action with infinite duration, $a_n^i = w = \langle w_\varphi, w_D \rangle$ with $w_D = \infty$, to capture that agents wait indefinitely at their final position. We define a plan π_i's duration as the sum of all timed actions' durations, naturally excluding the final infinite timed wait action:

$$\pi_{i,D} = \sum_{k=1}^{n-1} a_{k,D}^i$$

Figure 5.2 visualizes an example of a plan, moving an agent from v to v', consisting of five timed move actions and two timed wait actions.

All agents are volumetric for the same reasons as described in Chapter 4 for the LMAPF_R problem: if agents do not have a volume, then with continuous time they can be moved arbitrarily close to each other without conflicting. Therefore, each agent has a volume in $\mathcal{M}$ and two agents collide if their volumes intersect. For two agents i and j, each respectively executing the timed actions $\langle a^i, t^i \rangle \in \pi_i$ and $\langle a^j, t^j \rangle \in \pi_j$, if the agents collide while executing these timed actions then $\langle \langle a^i, t^i \rangle, \langle a^j, t^j \rangle \rangle$ is a *conflict*. We then say that π_i and π_j conflict with each other. A *move-move* conflict $\langle \langle m^i, t^i \rangle, \langle m^j, t^j \rangle \rangle$ contains two move actions, and a *move-wait* conflict $\langle \langle m^i, t^i \rangle, \langle w^j, t^j \rangle \rangle$ contains one move and one wait conflict.

A *joint plan* $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ contains one plan for each agent. If all plans $\pi \in \Pi$ are valid and no two plans in Π conflict with each other, then Π is a *solution*. With all solutions contained in the set $\mathbb{S}$, an optimal solution $\Pi^* = \arg\min_{\Pi \in \mathbb{S}} \sigma(\Pi)$ minimizes an objective function σ over $\mathbb{S}$. Typically, σ is the makespan or SOC. In Paper C, we only assume that $\sigma(\Pi)$ is strictly monotonically increasing with the maximum duration over all plans $\pi \in \Pi$, which both makespan and SOC satisfy.

Relevant Methods

To the best of our knowledge, CCBS is the only exact method addressing the full MAPF_R formulation. However, several other methods exist for solving either simplified MAPF_R variants or for finding sub-optimal solutions. We present a selection of these alternative methods here and defer the detailed introduction of CCBS to Section 5.3.

Prioritized Safe Interval Path Planning (PSIPP) [95], which was introduced in detail in Chapter 3, employs prioritized planning for a simplified MAPF_R problem where agents are modeled as circles moving at constant speed along straight-line trajectories between vertices. The time to traverse edge $e = (v, v') \in \mathcal{E}$ is determined by the Euclidean distance $|\mathcal{M}(v) - \mathcal{M}(v')|$. We argue that nothing with PSIPP inherently relies on any of these simplifications. In broad terms, PSIPP simply applies SIPP [33] to compute conflict-free plans sequentially for each agent, which could apply to the full MAPF_R problem if arbitrary move actions and agent volumes are used instead. Due to its prioritized nature, this method sacrifices both exactness and completeness [73], though it achieves significantly better scalability than CCBS.

Satisfiability Modulo Theory Conflict-Based Search (SMT-CBS$^\mathcal{R}$) [102] addresses a similar geometric simplification, assuming circular agents (with possible extensions to other shapes) traveling in straight lines at constant speed on undirected graphs. Rather than branching as in traditional CBS, SMT-CBS$^\mathcal{R}$ reformulates conflict resolution within the SMT framework, adding disjunctive constraints to eliminate conflicts without high-level branching. In other words, SMT-CBS$^\mathcal{R}$ offloads the high-level branching done in CBS to whichever SMT solver is used, Glucose 4 [103] in their case. This approach can find both makespan-optimal and SOC-optimal solutions while generating significantly fewer search tree nodes than direct adaptions of CBS to conti-

nuous time. Experimental results demonstrate that SMT-CBS$^{\mathcal{R}}$ outperforms CCBS by factors of 2–10 in runtime on tested instances [102].

The work in [104] searches for bounded sub-optimal solutions to the full MAPF_R problem. Their approach encodes MAPF_R as *satisfiability modulo linear real arithmetic*, enabling the use of off-the-shelf SMT solvers (MathSAT5 [105] in their case). The method demonstrates notably better runtime scaling than CCBS as time budgets increase, particularly excelling on bottleneck scenarios. Experimental comparisons show mixed results on success rates across different problem classes, with the SMT-based approach achieving superior performance in tightly constrained domains but occasionally underperforming on simpler instances where CCBS terminates quickly.

Extended Increasing Cost Tree Search (E-ICTS) [106] continues on the original ICTS algorithm [59] to handle non-unit cost domains. Rather than assigning fixed integer costs to each agent's plan, E-ICTS defines cost ranges for each agent with an incremental granularity. This formulation enables agents to perform actions at non-integer timesteps and traverse edges with arbitrary positive costs, applying to arbitrary weighted graphs with circular agents moving at constant speeds. While this approach handles non-unit costs and allows finer temporal discretization than traditional discrete MAPF, it still requires discretizing wait durations with fixed increments rather than permitting truly continuous waits where agents can initiate actions at arbitrary real-valued times.

The MAPF_R Search Space

We will now show that the search space for both the classical MAPF and the MAPF_R problem is infinitely large. We consider candidate solutions to be joint plans *containing only valid plans*. Recall that a valid plan is a sequence of actions connected in time and space, starting at $t = 0$ at the agent's start vertex and ending with ending with an infinite-duration wait action at the agent's goal vertex. Valid plans are easy enough to generate using, for instance, SIPP [33]. Thus, the challenge is to find a *collision-free* joint plan — a solution.

Consider any joint plan $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ in the search space. We can construct a nearly identical joint plan Π' by delaying the start of every plan

in Π by some arbitrary amount $\Delta \in \mathbb{R}$: for every plan

$$\pi = \langle\langle a_1, t_1\rangle, \ldots, \langle a_n, t_n\rangle\rangle \in \Pi,$$

let there exists a corresponding plan

$$\pi' = \langle\langle w, 0\rangle, \langle a_1, t_1 + \Delta\rangle, \ldots, \langle a_n, t_n + \Delta\rangle\rangle \in \Pi'$$

with the initial wait action $\langle w, 0\rangle$ — where $\mathit{from}^v(w) = \mathit{from}^v(a_1)$ and $w_D = \Delta$ — prepended and each action delayed by Δ. These joint plans differ only in when actions begin, not in their spatial trajectories or relative timing between agents. The same transformation applies to classical MAPF by letting $\Delta \in \mathbb{N}$ and prepending Δ-many wait actions instead of just one. Since there is no upper bound on Δ, infinitely many such joint plans can be constructed, proving that the search space is infinite for both classical MAPF and MAPF_R.

By the same reasoning, the feasible set is either empty or also infinitely large. If at least one solution exists, the same delay transformation can be applied to construct infinitely many solutions — each differing only in absolute timing while preserving all spatial trajectories and relative action sequences. Therefore, whenever the MAPF_R problem is feasible, it admits infinitely many solutions.

Unlike the classical MAPF formulation, however, the MAPF_R search space is *dense*. In classical MAPF, time is discrete and all action times must be integers. Consequently, Δ from above must take integer values, $\Delta \in \mathbb{N}$, yielding a *countably* infinite set of joint plans — infinite, yes, but with gaps between successive values. In MAPF_R, on the other hand, $\Delta \in \mathbb{R}$ can take any arbitrary real value. This means that between any two joint plans (say, with Δ_1 and Δ_2), there exists infinitely many other joint plans with delays $\Delta \in (\Delta_1, \Delta_2)$. Therefore, the MAPF_R search space contains *uncountably* infinite joint plans with no gaps between them — that is, it is dense. By the same reasoning, the feasible set is also dense if it is not empty. As we will see in the next section, this density has critical implications for ensuring that solutions to the MAPF_R problem can be found in finite time.

5.2 The Challenges of Algorithmic Guarantees

Chapter 2 introduced the terms *soundness*, *completeness*, and *exactness*. For simplicity, Paper C uses the term "soundness" to mean both soundness and exactness. However, for clarity we keep these terms separate here.

CCBS is not complete [16] for the same reason that CBS [60] is not complete. Instead, [16] claims that CCBS satisfies only one of the two completeness conditions: a solution will be returned if one exists. However, CCBS is not able to detect an infeasible instance; in fact, CCBS applied to an infeasible instance will never terminate. Thus, we define the weaker completeness variant:

- **Solution Completeness**: a solution-complete algorithm returns a solution if one exists, but does not necessarily identify when no solution exists.

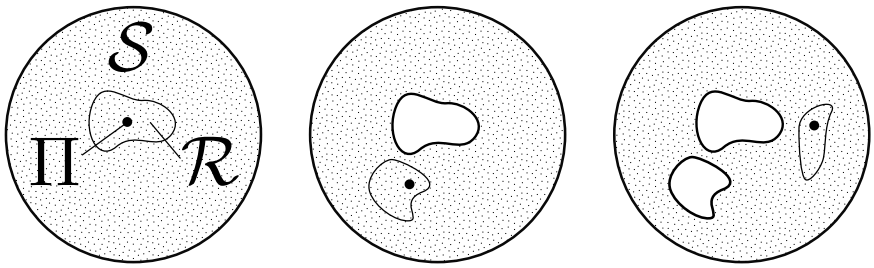
In the remainder of this section, we discuss in general terms how the infinite MAPF_R search space can be navigated. The approach is similar to CCBS's strategy, however, we refrain from describing CCBS until the next section. Thus, here we focus on how a general approach can and cannot guarantee soundness, completeness, exactness, and solution completeness. We divide these into two challenges: exactness and termination.

The Exactness Challenge

To achieve exactness, soundness is a good place it start. It is relatively easy to ensure an algorithm's soundness: return a candidate solution only if it is feasible, otherwise, discard it and keep searching the search space. This relies on it being relatively inexpensive to verify that a candidate solution is a solution — a reasonable assumption in MAPF_R. Therefore, we can construct a search algorithm that explores each candidate solution in the search space, and only returns a candidate solution if it is indeed a solution.

Ensuring an algorithm's exactness presents a greater challenge than ensuring soundness. With an infinitely large search space, determining whether a candidate solution is optimal requires a systematic approach to navigate this space effectively. One effective strategy — employed by CCBS [16] — is to use a *greedy search* to progressively narrow down the set $\mathcal{S}$ of reachable candidate solutions until an optimal solution is identified.

This approach relies on two key properties. First, any joint plan in the search space can be evaluated on the objective function σ, regardless of its feasibility. Second, we assume that there exists a mechanism for efficiently finding the candidate solution $\Pi = \underset{\Pi' \in \mathcal{S}}{\arg\min}\ \sigma(\Pi')$ that minimizes σ over $\mathcal{S}$. The search begins with $\mathcal{S}$ equal to the entire search space and then proceeds as follows:



(a) At the first iteration, $\mathcal{S}$ is equal to the entire search space.

(b) If Π is not a solution, $\mathcal{R}$ is removed from $\mathcal{S}$ and a new Π and $\mathcal{R}$ are found.

(c) The set of reachable candidate solutions $\mathcal{S}$ is progressively reduced until a solution is found.

Figure 5.3: An illustration of three iterations of the search strategy to find an optimal solution in $\mathcal{S}$. At each iteration, a candidate solution $\Pi \in \mathbb{S}$ minimizing σ is selected. If Π is not a solution, a region $\mathcal{R}$ containing Π and other infeasible candidate solutions is removed from $\mathcal{S}$.

1. If Π is collision-free, return it as the optimal solution. Since Π has the lowest objective value among all candidate solutions in $\mathcal{S}$ (which initially equals the entire search space), it must be optimal.
2. Otherwise, identify a region $\mathcal{R} \subset \mathcal{S}$ containing Π and other *infeasible* candidate solutions.
3. Remove $\mathcal{R}$ from the search space: $\mathcal{S} \leftarrow \mathcal{S} \backslash \mathcal{R}$.
4. Obtain a new joint plan $\Pi \leftarrow \underset{\Pi' \in \mathcal{S}}{\arg\min}\, \sigma(\Pi')$ minimizing σ over $\mathcal{S}$.
5. Return to step 1.

Figure 5.3 illustrates this procedure, demonstrating how $\mathcal{S}$ is reduced by $\mathcal{R}$ at every iteration where Π is not a solution. Crucially, if $\mathcal{R}$ contains only infeasible candidate solutions, then the first solution to be found is guaranteed to be an optimal solution. If instead $\mathcal{R}$ contains solutions, then those solutions are removed from $\mathcal{S}$ without ever being considered. Unless it is ensured that any solution in $\mathcal{R}$ is sub-optimal, there is a possibility that all optimal solutions end up being removed from the search. In that case, only sub-optimal solutions remain. Thus, this search algorithm's exactness relies on $\mathcal{R}$ never containing solutions.

The Termination Challenge

Ensuring solution completeness requires guaranteeing that an algorithm will find a solution and return it whenever one exists. Full completeness further requires that the algorithm also terminates when no solution exists, reporting that the instance is unsolvable. Algorithms that fail to terminate in certain cases are fundamentally problematic — without termination guarantees, it becomes impossible to distinguish between a long-running search that will eventually succeed and one that will never terminate. We now examine the greedy search algorithm from above and identify three scenarios that can prevent termination.

First, note that the greedy algorithm searches for solutions from the bottom up, by always selecting the candidate solution minimizing σ. Suppose now that $\mathcal{R} \subset \mathcal{S}$ contains solutions. Then, it is possible at every iteration that the algorithm selects an infeasible candidate solution and removes solutions in $\mathcal{R}$, such that the next candidate solution in the new search space $\mathcal{S} \leftarrow \mathcal{S} \backslash \mathcal{R}$ minimizing σ is also infeasible. In other words, the algorithm would essentially remove all solutions in front if it in its search. This would cause the algorithm to never terminate.

Second, if the feasible set is empty, then the algorithm will never trigger its termination criteria: return only when a solution is found. In that case the search will indefinitely navigate the infinite search space. It is for this reason that CBS and CCBS are are not complete, but instead solution complete.

Finally, suppose that the feasible set is not empty and $\mathcal{R}$ never contains solutions. Then the algorithm is sound and exact. Under these conditions, however, termination may still fail as a consequence of the search space being dense. This crucially depends on the shape of $\mathcal{R}$ and if it results in a significant part of $\mathcal{S}$ being removed at every iteration. We illustrate this with an example. Let Π^* be the optimal solution. Consider at some iteration an infeasible candidate solution Π is selected, with $\sigma(\Pi) < \sigma(\Pi^*)$. Since the search space is dense, it will always be possible to construct a region $\mathcal{R}$ containing all candidate solutions Π' with an objective value between $\sigma(\Pi)$ and $\sigma(\Pi^*)$,

$$\sigma(\Pi) < \sigma(\Pi') < \sigma(\Pi^*).$$

By only removing all such candidate solutions, the search will approach but never reach Π^*. Figure 5.4 visualizes the search asymptotically approaching — but never reaching — Π^* in the search space. Thus, if $\mathcal{R}$ does not remove

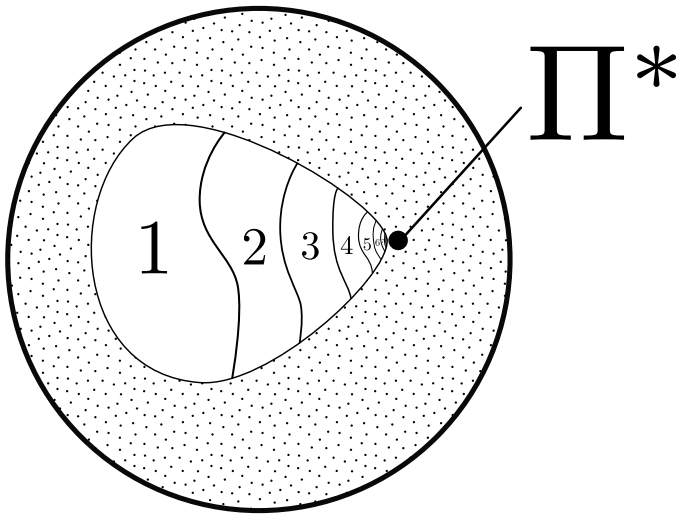


Figure 5.4: An illustration of the search progressively removing regions from the search space while asymptotically approaching the optimal solution Π^*, without ever reaching it.

a "substantial" part of the search space, such that progress toward an optimal solution is asymptotic toward zero, then even a sound and exact algorithm is not guaranteed to terminate. Consequently, such an algorithm is not complete or solution complete.

5.3 Continuous-Time Conflict-Based Search

CCBS's approach to solving MAPF_R for optimal solutions consists of a high-level search and a low-level planner. The high-level search constitutes the core of CCBS, which is done in a binary *constraint tree* (CT) with nodes representing candidate solutions and constraint sets. A leaf node is selected at each iteration, if it does not represent a solution then two child nodes are created using a *branching rule*. The theoretical description in [16] of CCBS presents one branching rule, while the publicly available implementation[2] applies another. In this section, we introduce CCBS's high-level search and

[2]https://github.com/PathPlanning/Continuous-CBS

low-level planner in detail, and thereafter discuss the limitations of each of the branching rules as identified in [32] and formalized in Paper C.

Each node N in the CT is associated with a set of constraints N_C and a joint plan N_Π containing only valid plans that satisfy all constraints in N_C. N_Π is computed using the low-level planner, *Constrained Safe Interval Path Planning* (CSIPP), which accepts a set of constraints and applies SIPP [33] to find the shortest-duration valid plan satisfying those constraints. CCBS considers two types of constraints: *motion constraints* $\langle i, a, [t_1, t_2)\rangle$ forbidding agent i from executing action a starting at any time in $[t_1, t_2)$, and *vertex constraints* $\langle i, v, [t_1, t_2)\rangle$ forbidding agent i from occupying vertex $v \in \mathcal{V}$ during $[t_1, t_2)$. In the CSIPP search, motion constraints form unsafe intervals on the move actions between states and vertex constraints form unsafe interval at vertices. Thereafter, SIPP is applied to find an agent's shortest-duration valid plan satisfying all constraints.

The high-level search starts at the root node of the CT, R, with $R_C = \varnothing$. Since R_C is empty, the joint plan R_Π contains for each agent the shortest-duration valid plan over all valid plans, which is essentially the shortest path from its start vertex to goal vertex without considering collisions with other agents. At each iteration of CCBS, the high-level search performs the following steps:

1. The CT leaf node
$$N = \underset{N' \in \text{ CT leaves}}{\arg\min} \ \sigma\left(N'_\Pi\right)$$
minimizing σ is selected. Let $\mathcal{S}^N$ contain all candidate solutions satisfying N_C, illustrated in Figure 5.5a.

2. If N_Π is collision-free, then it is returned as the optimal solution.

3. If N_Π is not collision-free, then an arbitrary conflict
$$\langle\langle a^i, t^i\rangle, \langle a^j, t^j\rangle\rangle \in \pi_i \times \pi_j, \quad \pi_i, \pi_j \in N_\Pi,$$
is selected. A branching rule is applied to $\langle\langle a^i, t^i\rangle, \langle a^j, t^j\rangle\rangle$ to obtain a pair of constraints $\langle c_i, c_j\rangle$, where c_i at least forbids agent i from executing $\langle a^i, t^i\rangle$ and c_j at least forbids agent j from executing $\langle a^j, t^j\rangle$.

4. Two new children to N are created, N^i and N^j, each respectively with $N^i_C = N_C \cup \{c_i\}$ and $N^j_C = N_C \cup \{c_j\}$. Let $\mathcal{S}^i \subset \mathcal{S}$ and $\mathcal{S}^j \subset \mathcal{S}$ respectively contain all candidate solutions satisfying N^i_C and N^j_C, illustrated

in Figure 5.5b along with the region $\mathcal{R} = \mathcal{S} \backslash \mathcal{S}^i \backslash \mathcal{S}^j$. $\mathcal{R}$ contains N_Π, along with all other candidate solutions not satisfying at least one of c_i or c_j.

5. The low-level planner is used to compute each child node's joint plan, N^i_Π satisfying N^i_C and N^j_Π satisfying N^j_C.

6. Repeat from step 1.

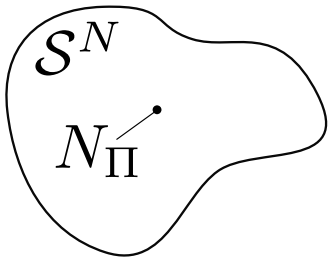
When branching on N, its children N^i and N^j inherit all constraints and additionally gain new constraints. Constraints are never removed from parent to child. Consequently, all candidate solutions contained in $\mathcal{R}$ are removed from the set of reachable candidate solutions in the sub-tree below N. Figure 5.5c visualizes the evolution of the set of reachable candidate solutions in the CT. For each branching, the reachable candidate solution set ($\mathcal{S}$) is divided into two subsets ($\mathcal{S}^i$ and $\mathcal{S}^j$) with some region ($\mathcal{R}$) removed.

CCBS is sound because it only returns collision-free joint plans, i.e., solutions. However, exactness requires that if $\mathcal{R}$ contains any solutions, all of those solutions must be sub-optimal — otherwise, the algorithm risks pruning all optimal solutions from the set of reachable candidate solutions. Furthermore, termination demands two conditions: first, all solutions must not be removed from the set of reachable candidate solutions as the search progresses; and second, it must remove sufficiently large portions of the reachable candidate solution set to prevent infinite convergence toward the optimal solution without ever reaching it. Since the branching rule determines $\mathcal{R}$ through the constraint pair $\langle c_i, c_j \rangle$, both the exactness and solution completeness of CCBS depend on the branching rule.

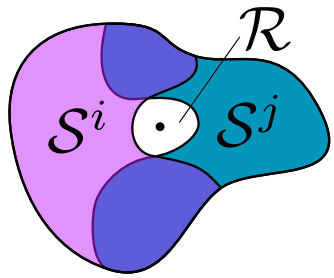
The Theoretical Branching Rule

A branching rule takes as input a conflict $\langle \langle a^i, t^i \rangle, \langle a^j, t^j \rangle \rangle$ and returns a constraint pair $\langle c_i, c_j \rangle$. The theoretical branching rule (TBR) is described in [16]. The actions a^i and a^j may be move or wait actions, however, the TBR does not differentiate between these. Given a conflict $\langle \langle a^i, t^i \rangle, \langle a^j, t^j \rangle \rangle$ between two agents i and j, the TBR applies the following steps to generate $\langle c_i, c_j \rangle$:

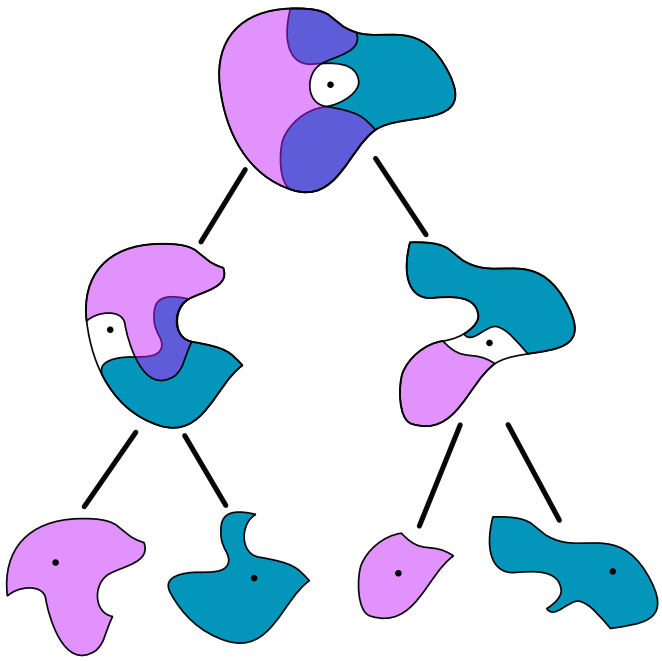
1. Find the earliest time $t^i_u > t^i$ such that $\langle \langle a^i, t^i_u \rangle, \langle a^j, t^j \rangle \rangle$ is not a conflict. That is, t^i_u is the earliest time when agent i can execute $\langle a^i, t^i_u \rangle$ without



(a) For a CT node N, $\mathcal{S}^N$ contains all candidate solutions permitted under N_C, one of them being N_Π.



(b) Branching on N creates two children N^i and N^j. $\mathcal{S}^i$ and $\mathcal{S}^j$ contain all candidate solutions satisfying N_C^i and N_C^j, respectively. The region $\mathcal{R}$ is removed from the search.



(c) At every branch in the CT, the set of reachable candidate solutions is split into two subsets, with a region $\mathcal{R}$ not included in either.

Figure 5.5: An illustration of how branching nodes in the CT progressively reduces the size of the set of reachable candidate solutions.

colliding with agent j executing $\langle a^j, t^j \rangle$. We denote $[t^i, t^i_u)$ as the *unsafe interval* for i to execute action a^i.

2. Similarly for agent j, find the earliest time $t^j_u > t^j$ such that $\langle\langle a^i, t^i \rangle, \langle a^j, t^j_u \rangle\rangle$ is not a conflict, forming the unsafe interval $[t^j, t^j_u)$.

3. Create two motion constraints: $c_i = \langle i, a^i, [t^i, t^i_u) \rangle$ and $c_j = \langle j, a^j, [t^j, t^j_u) \rangle$, forbidding each agent from executing their respective actions during their respective unsafe intervals. This results in the constraint pair $\langle c_i, c_j \rangle$.

It is shown in [16] that the region $\mathcal{R}$ of candidate solutions not satisfying at least one of c_i or c_j does not contain any solutions, implying that CCBS using the TBR is exact. However, the work in [32] suggests that CCBS under the TBR does not guarantee termination due to $\mathcal{R}$ not always removing sufficiently large portions of the set of reachable candidate solutions. The issue, as described in [32], originates from TBR's handling of move-wait conflicts. Consider a move-wait conflict $\langle\langle m^i, t^i \rangle, \langle w^j, t^j \rangle\rangle$. The TBR generates the constraint pair $\langle c_i, c_j \rangle$ with $c_i = \langle i, m^i, [t^i, t^i_u) \rangle$ and $c_j = \langle j, w^j, [t^j, t^j_u) \rangle$. In particular, c_j forbids agent j from executing the specific wait action $w^j = \langle w^j_\varphi, w^j_D \rangle$ at any time in $[t^j, t^j_u)$. However, this does not forbid a nearly identical wait action $w = \langle w^j_\varphi, w^j_D + \epsilon \rangle$ for any arbitrarily small $\epsilon \in \mathbb{R}$. In other words, although the constraint pair $\langle c_i, c_j \rangle$ removes all joint plans containing the specific conflict $\langle\langle m^i, t^i \rangle, \langle w^j, t^j \rangle\rangle$, it does not remove joint plans with arbitrarily similar conflicts. Thus, [32] argues that CCBS using the TBR is not guaranteed to terminate, implying that it is not solution complete.

We believe that the work in [32] suggests that CCBS using the TBR *may* not be solution complete, however, they do not prove that CCBS using the TBR *is* not solution complete. The termination proofs for the proposed branching rule in Paper C relies on how the underlying path planner, CSIPP, plans individual agent plans. It may be so that CCBS using the TBR is indeed solution complete, perhaps even for similar reasons that the branching rule in Paper C is guaranteed to terminate. However, the termination proofs in Paper C do not directly apply to CCBS using the TBR since the TBR forbids wait actions of a *singular-valued duration* and therefore does not produce vertex constraints, whereas the termination proofs rely on both motion and vertex constraints to ensure that substantial progress toward a solution is maintained at every iteration. Nonetheless, [32] points toward a crucial yet unaddressed aspect of

CCBS using the TBR for guaranteeing solution completeness.

The Implemented Branching Rule

The implemented branching rule (IBR) is used in the implementation of CCBS — made available by the authors of [16]. Move-move conflicts are handled in the same way in both the TBR and the IBR. However, the TBR and IBR differ in how they handle move-wait conflicts.

Given a move-wait conflict $\langle\langle m^i, t^i\rangle, \langle w^j, t^j\rangle\rangle$, the IBR constructs $\langle c_i, c_j\rangle$ through the following steps:

1. The same procedure as in the TBR is used to construct c_i: find the earliest time $t^i_u > t^i$ where $\langle\langle m^i, t^i_u\rangle, \langle w^j, t^j\rangle\rangle$ is not a conflict. The unsafe interval $[t^i, t^i_u)$ results in the motion constraint $c_i = \langle i, m^i, [t^i, t^i_u)\rangle$.

2. The *intersection interval* I is defined as the time interval during which the moving agent i would be in collision with the waiting agent j, if i executes $\langle m^i, t^i\rangle$ and j waits indefinitely at vertex $\mathit{from}^v(w^j)$. I is then used to create the vertex constraint $c_j = \langle j, \mathit{from}^v(w^j), I\rangle$, forbidding agent j from occupying vertex $\mathit{from}^v(w^j)$ at any time in I.

The work in [32] provides two counterexamples showing how the IBR removes solutions from the set of reachable candidate solutions. That is, they show that $\mathcal{R}$ contains solutions. As discussed, $\mathcal{R}$ containing solutions does not necessarily imply that CCBS is not exact since all solutions in $\mathcal{R}$ could be sub-optimal. However, without any assurances that $\mathcal{R}$ contains only sub-optimal solutions, these findings do indicate that CCBS may not be exact.

In Paper C, we complement these counterexamples by formalizing why solutions are inadvertently removed from the set of reachable candidate solutions. This is done by decomposing c_i and c_j into sets C_i and C_j, respectively, of *forbidding constraints*. A forbidding constraint $\langle i, a, t\rangle$ forbids agent i from executing the specific timed action $\langle a, t\rangle$. We then show for some pair $\langle c'_i, c'_j\rangle \in C_i \times C_j$ that an assignment not satisfying either of c'_i or c'_j does not necessarily contain a conflict — and therefore may be a solution — consistent with [32] that $\mathcal{R}$ may contain solutions. In other words, we show together with [32] that $\mathcal{R}$ may contain solutions in certain cases and additionally *why* this happens. Ultimately, however, Paper C provides definitive evidence that CCBS using the IBR does remove *all optimal solutions* in some cases. Therefore, CCBS using the IBR is not exact.

The limitations shown in this and the previous section — that CCBS using the TBR is suspected to not terminate on solvable instances and CCBS using the IBR is not exact — reveal the core challenge for CCBS-styled optimal MAPF_R algorithms: the constraints produced by a branching rule must simultaneously preserve all solutions (for soundness and exactness) while removing significant portions of the assignment space to ensure that a solution (if one exists) will be reached in finite time (for termination). Since CCBS, to our knowledge the only method for finding optimal solutions to the full MAPF_R problem, is shown in Paper C to not be exact, we therefore conclude that no existing algorithm addresses MAPF_R for optimal solutions within finite time. This opens a gap in the research that Paper C addresses.

5.4 Answering Research Question 3

To answer research question **RQ3** — *How can the continuous-time MAPF problem be solved in finite time with guaranteed solution optimality?* — Paper C introduces the CCBS branching rule δ-BR, based on the concept of *shifting constraints* from [32]. δ-BR is shown to preserve all solutions in the search by ensuring that a solution must satisfy at least one constraint in the resulting constraint pair. Thus, δ-BR is guaranteed to not remove any solutions from the search. Consequently, CCBS using δ-BR is sound and exact. Furthermore, CCBS using δ-BR is guaranteed to terminate. This relies on CSIPP always returning plans with action times at the start of safe intervals (as is consistent with SIPP [33]) and that every branching removes a significant part of the search space. This leads to the search space between the start of the search and an optimal solution to be exhausted within finite time, guaranteeing termination within finite time under the existence of a solution. Therefore, CCBS using δ-BR is also solution complete. To our knowledge, Paper C provides the first provably sound, exact, and solution complete MAPF_R solver, thereby answering **RQ3**.

CHAPTER 6

Summary of Included Papers

This chapter provides a summary of the included papers.

6.1 Paper A

Francesco Popolizio, Martina Vinetti, **Alvin Combrink**, Sabino Francesco Roselli, Maria Pia Fanti, Martin Fabian
Online Conflict-Free Scheduling of Fleets of Autonomous Mobile Robots
20th International Conference on Automation Science and Engineering (CASE), IEEE, Bari, Italy, 2024, pp. 3063-3068, doi: 10.1109/CASE59546.2024.10711693

This paper presents the *Fleet Manager* (FM) to address a generalized variant

of the Lifelong Multi-agent Path Finding problem, where edges may require multiple timesteps to traverse and tasks have service times (requiring agents to remain at task vertices for specified durations). The FM operates online, being called at every timestep to iteratively assign tasks to idle agents and coordinate three key components: the *path planner*, *scheduler*, and *conflict manager*. The path planner computes a spatial path from the agent's current location to the task vertex using Dijkstra's algorithm [29] on the map graph with modified edge weights that penalize paths passing through locations of assigned tasks and idle agents. The scheduler then computes temporal assignments for the path, minimizing the arrival time at the task vertex while avoiding collisions with previously scheduled agents. Two scheduler variants are implemented: one using the SMT solver Z3 [107], and another using a *Tailor-Made Scheduler* (TMS), which schedules iteratively by starting with a no-wait schedule and adding wait actions in chronological order until all collisions are resolved. When the scheduler cannot find a collision-free schedule, the conflict manager moves idle agents out of the way using a strategy analogous to the pebble motion problem [108]. The FM is not a complete solver and may encounter rare deadlock situations. However, experimental results demonstrate computation times in the milliseconds per task with up to 750 agents when using the TMS. Furthermore, the FM achieves solution throughput 1.7–2.8 time higher than *Rolling Horizon Collision Resolution* [14].

Contributions: This paper extends the master thesis work of FP and MV, supervised by SFR, MPF, and FM. FP and MV contributed to conceptualization, software implementation, validation, and writing the original draft. AC contributed to software implementation, validation, and writing the original draft. SFR and MF provided supervision and contributed to writing through review and editing, while MPF provided supervision.

6.2 Paper B

Alvin Combrink, Sabino Francesco Roselli, Martin Fabian
Prioritized Planning for Continuous-time Lifelong Multi-agent Pathfinding
11th International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, Split, Croatia, Jul. 2025.
To appear in conference proceedings.
Available online: `https://arxiv.org/abs/2503.13175`.

This paper presents the *Continuous-time Prioritized Lifelong Planner* (CPLP) to address the Lifelong Multi-agent Path Finding problem in continuous time with volumetric agents. CPLP is, to the authors' knowledge, the first method to address this combined problem and currently the only one applicable to general graphs. CPLP employs prioritized planning with asynchronous windowed planning, with agent movements planned within a time window. Task-agent pairs are ordered by priority — specifically by the time since task release — and planned one at a time. The highest-priority task-agent pair is planned entirely to task's vertex using a modified *Continuous-time Conflict-Based Search* (CCBS) [16] that handles idle agents, ensuring eventual task completion. For all other task-agent pairs, *Safe Interval Path Planning* [33] plans agent movement toward their task vertices within a time horizon, though not necessarily reaching them. To enable fast real-time performance with volumetric agents, collision information is pre-computed using geometric lookup tables. CPLP is computation-time aware, allowing a computation budget to be specified. Importantly, the planning architecture ensures that agents will not collide even if the computation budget is exceeded, as long as they execute their decided plans. This is achieved by assuming agents remain idle indefinitely at their last planned position, preventing other agents from planning paths through those positions. CPLP does not guarantee that all agents will eventually reach their task vertices, as the CCBS version used is not solution complete. To address potential deadlocks, the algorithm employs random movement to shuffle agents when no valid plan is found, which experimentally proved sufficient for the tested instances. Experimental results demonstrate real-time performance with up to 800 agents on graphs with up to 12 000 vertices, with computation times consistently within the allocated budget and throughput near 100% for the tested task release rate.

Contributions: AC contributed to the original conceptualization, methodology, software, validation, and writing of the original draft. SFR and MF contributed with supervision and writing through review and editing.

6.3 Paper C

Alvin Combrink, Sabino Francesco Roselli, Martin Fabian
Optimal Multi-agent Path Finding in Continuous Time
Under review in *Artificial Intelligence*, Elsevier, 2025.
Preprint available online: `https://arxiv.org/abs/2508.16410`.

This paper addresses optimal Multi-agent Path Finding problem in continuous-time (MAPF_R), motivated by recent findings [32] suggesting critical shortcomings in *Continuous-time Conflict-Based Search* (CCBS) [16] — to the authors' knowledge, the only method claimed to find optimal MAPF_R solutions. Specifically, it is suggested that the theoretically described CCBS fails to guarantee termination on solvable problems, while the publicly available reference implementation can return sub-optimal solutions. To address this reopened research gap, this paper introduces an analytical framework for understanding CCBS-style branching rules, establishing sufficient (but not necessary) conditions for ensuring that such algorithms are sound, exact, and solution complete. Applying this framework to CCBS's implemented branching rule reveals violations of these conditions, with experimental evidence confirming that CCBS is indeed not exact. The paper proposed δ-BR, a new branching rule that provably satisfies all sufficient conditions, ensuring that CCBS using δ-BR is sound, exact, and solution complete. To the authors' knowledge, this represents the first MAPF_R solver matching the algorithmic guarantees of discrete-time *Conflict-Based Search* [60] — returning only optimal solutions and guaranteed to terminate on any solvable MAPF_R problem. The guarantees hold for sum-of-costs, makespan, and any objective function that is strictly monotonically increasing with respect to maximum agent arrival time. Beyond its practical contribution, δ-BR serves as a drop-in replacement for existing CCBS variants and extensions, requiring modifications to the branching step. Furthermore, the analytical framework provides a tool for rigorous analysis and development of next-generation MAPF_R methods. Experimental results demonstrate that while the reference CCBS implementation often finds solutions faster through aggressive pruning, this comes at the cost

of occasional sub-optimality (up to 16% worse sum-of-costs in a constructed example) and potential non-termination when all solutions are pruned. In contrast, δ-BR preserves optimality and guarantees termination by design.

Contributions: AC contributed to the original conceptualization, methodology, software, validation, and writing of the original draft. SFR and MF contributed to supervision and writing through review and editing.

CHAPTER 7

Conclusions, Reflections and Future Work

7.1 Conclusions

MAPF is the problem of moving multiple agents in a shared space, from where they are to where they should be, while avoiding collisions. The MAPF research field has a growing range of real-world applications, from warehouses and industrial manufacturing, to video games and office robots.

This thesis addresses three distinct challenges along the frontiers of current MAPF research. Research question **RQ1** aims to characterize the current state of Lifelong MAPF by establishing which strategies underpin scalable real-time performance. Three prominent strategies are identified in Chapter 3: prioritized planning, windowed planning, and dimensional simplifications. Paper A then applies a few of these strategies, achieving a Lifelong MAPF solver scalable to hundreds of agents with competitive solution quality.

Research question **RQ2** then aims to extend the insights gained from answering **RQ1** to continuous time — a domain where, to our knowledge, lifelong planning has not yet been explored. Chapter 4 establishes that extending LMAPF to continuous time presents several challenges: asynchronous actions hinder the use of many existing discrete-time methods; the dense search space

requires sophisticated navigation strategies; and volumetric agents make collision detection computationally demanding and collision resolution complicated as the range of possible conflicts is expanded in comparison with non-volumetric agents. Paper B addresses these challenges, providing a scalable continuous-time Lifelong MAPF solver that ensures collision-free movement even if computation budgets are exceeded.

Finally, research question **RQ3** aims to address the recently reopened question of how optimal solutions to the continuous-time MAPF problem can be found, once thought settled by CCBS. Chapter 5 provides necessary background by introducing CCBS and the recent findings regarding its limitations. Paper C then verifies these shortcomings by providing definitive evidence that CCBS is not exact. Furthermore, the CCBS branching rule δ-BR is introduced and subsequently proven to restore soundness, exactness, and solution completeness to CCBS.

Collectively, these contributions map current knowledge boundaries, extend MAPF into continuous-time lifelong domains for the first time, and restore theoretical guarantees to optimal continuous-time MAPF solvers.

7.2 Reflections

The two central themes of this thesis — achieving scalable MAPF and extending it to continuous-time — are deeply interconnected. Real-world systems inherently operate in continuous time, yet discrete-time formulations dominate the MAPF literature. While discrete-time abstractions simplify computation, they constrain the set of solutions, limiting achievable solution quality. Continuous-time formulations lift these artificial constraints, enabling solutions that better reflect physical reality and potentially achieve higher quality. The trade-off, however, is computational complexity. This partially reflects the relative maturity of the two domains; discrete-time MAPF research dates back to at least the 1980's [74], whereas continuous-time MAPF first emerged with SIPP [33] only in 2011. Even then, another eight years passed before the introduction of the first versions of CCBS [109, 110] in 2019. This illustrates either the substantial challenges in coordinating multiple agents without time discretization, the relative difference in research interest aimed toward discrete-time compared to continuous-time MAPF, or both. As continuous-time MAPF matures and benefits from sustained research attention — much

as discrete-time MAPF has over decades — we anticipate future continuous-time algorithms will achieve the scalability of today's discrete-time methods while delivering solutions of fundamentally higher quality than what discrete-time formulations can provide.

7.3 Future Work

There is much work left to be done in the field of MAPF.

Direct Extensions to This Work

The most immediate extensions build directly upon the contributions of this thesis. For CPLP, introduced in Paper B, several improvements would extend its theoretical guarantees and lead to better scalability. Currently, CPLP uses a modified variant of CCBS [16] to compute an entire plan for the prioritized agent-task pair. The modifications pertain to handling agents without goal vertices, a common occurrence in lifelong MAPF when there are, for instance, fewer tasks than agents. However, Paper C shows that this original CCBS variant lacks exactness guarantees. Future work could focus on using Paper C's CCBS with δ-BR in CPLP and ensure that the modifications made in CPLP do not remove the exactness and — more importantly — the solution completeness guarantees. We postulate that these modifications indeed do not affect CCBS's guarantees since they target parts of CCBS's low-level planner that are not involved in Paper C's theoretical proofs. With CPLP using a solution complete modified CCBS, we can then ensure that the prioritized agent will (if a solution exists) reach its goal vertex. This would provide CPLP with similar theoretical guarantees of *reachability* — defined in [72] as all agents eventually reaching their goal vertex — for the LMAPF_{R} problem as is currently enjoyed by, e.g., PIBT for the discrete-time LMAPF problem.

CPLP's scalability can be improved by shortening the length of plans computed by CCBS and thereby reducing CPLP's current bottleneck for scaling to more agents — the maximum computation time-per-call. CPLP uses CCBS to compute an entire plan for a prioritized agent-task pair. Ensuring that the prioritized agent will reach its goal is critical for guaranteeing reachability. However, this can also be achieved by ensuring that the prioritized agent makes strict monotonic progress *toward* its target. For instance, CCBS could

be used to compute a plan for the prioritized agent to any node *closer* to its goal. In essence, CCBS would then be used to compute several short steps instead of one entire plan, thereby spreading computations over several calls. With CPLP's maximum computation time-per-call reduced, more agents can be planned before exceeding the available computation window. This would have additional benefits: a reduced computation time-per-call means that a lower computation window can be used. In turn, that would mean that the planning window (which must be greater or equal to the computation window for continuous movement) can be set to a lower value. This would lead to computed plans being shorter, in general, and thereby lowering the amount of computations performed per agent. CPLP would then be able to plan for larger agent counts while also benefiting from improved reactivity to new incoming tasks.

Even with a reduced maximum computation time, there may always be a risk of CPLP exceeding the available computation time. This would likely not be considered catastrophic in the real-world since CPLP's planning ensures that no collisions occur as long as all agents continue to follow their decided plans and then remain idle. Despite this, CPLP lacks routines for handling situations when the available computation time is exceeded. A simple way to address this would be to abandon the plans that were computed when the available time was exceeded, assume that all agents follow their plans and then remain idle, temporarily increase the available computation time, and compute new plans that start at the end of this larger computation window. This could be repeated, increasing the available computation time for every iteration, until CPLP is able to compute a plan in time. This would in practice lead to sub-optimal movement where agents remain idle for a period of time, however, it would at least ensure that operations continue automatically. Additionally, if Δt is selected appropriately for normal operations then this period of sub-optimal movement should be temporary.

Furthermore, many real-world applications require agents to visit sequences of locations rather than single destinations — transporting goods from pickup to delivery locations as in the MAPD problem, or more generally, the Multi-Goal MAPD problem. Tasks involving multiple goal vertices can be handled by single-goal algorithms, such as CPLP, by using the same approach as in [14] with forced task-agent assignments. However, extending CPLP to *explicitly* handle goal sequences could broaden its practical applicability, particularly

in warehouse and manufacturing scenarios where sequential tasks seem to be the norm rather than the exception, and potentially unlock solutions of higher quality.

The lifelong nature of LMAPF also presents an opportunity to exploit patterns in task arrival. In many applications, tasks exhibit recurring locations — delivery routs, production cycles, or predictable traffic patterns. By applying statistical methods to predict where future tasks are likely to appear, agents could be strategically positioned preemptively, reducing response times and increasing throughput. Such predictive positioning could be incorporated into CPLP by moving idle agents toward high-probability task locations rather than remaining stationary.

Finally, future work could focus on adopting the numerous enhancements and variants of the original CCBS to using δ-BR, thereby ensuring that they too enjoy the same theoretical guarantees that Paper C provides.

Increasing Real-World Applicability

Several fundamental challenges must be addressed to aid the use of MAPF algorithms in the real-world. These challenges stem from, for instance, the following three sources of uncertainty: agent control, environmental knowledge, and execution dynamics.

Most MAPF studies assume idealized agent models with omnidirectional movement and constant velocity. Reality imposes kinematic constraints — agents may have limited turning radii, acceleration bounds, and directional preferences. While discrete-time methods have addressed such constraints [111], continuous-time approaches remain largely unexplored. CCBS's flexible motion representation, which allows move actions to encode arbitrary trajectories, provides a foundation for incorporating kinematic constraints. However, current formulations still assume agents can instantaneously reorient at vertices. Exploring ways to combine high-level MAPF planning with low-level trajectory planners, as done in [112], could yield solutions that better integrate agent kinematics and collision avoidance maneuvers. If not integrating low-level control at the planning stage, then at least quantifying the risks associated with not doing so could offer MAPF algorithm practitioners with valuable information for deciding safety margins.

Uncertainty in both agent control and environment knowledge manifests as delays, unpredictable travel times, and dynamic obstacles. Discrete-time

MAPF addresses execution robustness through k-robustness [49], which ensures schedules remain collision-free despite delays up to k timesteps, and probabilistic robustness [50], which incorporates probability distributions over delays. T-robustness [100] provides a fixed-buffer approach to continuous-time MAPF, analogous to k-robustness. A natural extension would be to also develop the probabilistic approach to continuous-time MAPF, which to our knowledge has yet to be explored. The advantage of probabilistic methods is their adaptability — they incorporate uncertainty directly into scheduling decisions, rather than applying uniform safety margins regardless of actual risk.

Novel Methodological Directions

As with many scientific fields, the capabilities of learning-based methods have also begun to make an impact in the MAPF community [89]. However, applying *machine learning* (ML) to MAPF presents a fundamental challenge: MAPF is a constrained optimization problem where assignments must satisfy strict constraints to constitute valid solutions. Pure ML-based approaches often struggle to guarantee constraint satisfaction [113], making them unreliable for safety-critical applications. Where ML excels for these types of problems, however, is in learning heuristics to guide classical algorithms — combining the guarantee structure of traditional methods with the pattern-recognition capabilities of learned models.

An intriguing hybrid direction draws inspiration from the AlphaZero framework [114], which revolutionized game-playing AI in *chess*, *shogi*, and *go* by combining *Monte Carlo Tree Search* (MCTS) with learned value and policy networks. MAPF shares key structural properties with these domains: state transitions are deterministic (assuming known traversal times), solutions are assumed to be reachable in a finite number of steps, and only a subset of all discrete actions are feasible from any given state. A hybrid MAPF approach could leverage learned heuristics to guide classical search algorithms in more promising directions, thereby potentially discovering novel coordination strategies that humans might overlook. Such methods offer several advantages: they could rapidly compute high-quality solutions, function as anytime algorithms that progressively improve solutions over the available computation time, and learn task location patterns for preemptive agent positioning. To our knowledge, the only method to successfully apply something similar

to this type of framework in MAPF is the recent *LaGAT* [115]. More work in this direction, and additionally addressing continuous-time formulations, could provide algorithms with better scalability and solution quality. Moreover, learned models could implicitly capture kinematic constraint and execution uncertainty by training on data from realistic simulators or real-world deployments, simultaneously addressing multiple challenges outlined earlier.

Another interesting and recent advancement is the *Hierarchical Reasoning Model* [116] (strongly inspired by Danial Kahneman's *automatic thinking* and *deliberate reasoning* model of human thinking [117]), which was shortly after extended to the *Tiny Reasoning Model* (TRM) [118] with with smaller networks and better performance. In broad terms, these architectures perform multiple cycles of solution improvement, where each cycle starts with a constant "current solution", involves several inference steps (supposedly to reason about improvements on the solution), and ends with an update of the current solution by incorporating the improvement reasoning. TRM demonstrates superior performance compared to existing LLMs on problems requiring multi-step reasoning [118], such as the ARC-AGI benchmark [119], difficult Sudoku puzzles, and *finding optimal paths in* 30×30 *grid-based mazes*. The latter is essentially a single-agent path planning problem, easily solved using e.g. Dijkstra's algorithm. However, TRM achieve roughly 75% accuracy using a purely learning-based method. Combining TRM with classical search algorithms — similar to AlphaZero's combination of MCTS with deep neural networks — could be a viable future direction to address multi-agent coordination.

The primary technical challenge which we identify for any learning-based MAPF approach is representation invariance — models should ideally generalize across maps of different sizes, topologies, and agent counts. Many ML architectures are inherently unable to handle non-fixed-sized inputs. Although, there remain many ML architectures, such as the *transformer* [27] and the *graph neural network* (GNN) [26]. Existing methods typically overcome this by fixing the size of each agent's *observable* surroundings and predicting the agent's next action [63, 65, 66, 115]. GNNs provide a natural solution to this challenge since they operate directly on graph-structured data and are inherently invariant to graph size and topology. A GNN could learn to reason about congestion patterns, predict collision likelihood, guide coordination strategies, and encode spatial relationships in ways that transfer across problem instances. Additionally, by leveraging GNN's invariance to topolo-

gies, ML-based methods for the non-grid-based continuous-time MAPF maps could be developed.

To the best of our knowledge, there exist only a handful of ML-based methods for discrete-time MAPF and none for continuous-time MAPF. Therefore, this provides one of the several frontiers in the MAPF research landscape which we intend to explore.

References

[1] Amazon Science. "How Amazon robots navigate congestion," Accessed: Sep. 29, 2025. [Online]. Available: https://www.amazon.science/latest-news/how-amazon-robots-navigate-congestion.

[2] Amazon. "Amazon deploys over 1 million robots and launches new AI foundation model," Accessed: Sep. 29, 2025. [Online]. Available: https://www.aboutamazon.com/news/operations/amazon-million-robots-ai-foundation-model.

[3] ABB Robotics. "ABB opens autonomous mobile robotics training and showroom facility," Accessed: Sep. 29, 2025. [Online]. Available: https://new.abb.com/news/detail/122330/abb-opens-autonomous-mobile-robotics-training-and-showroom-facility.

[4] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[5] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig, "Feasibility study: Moving non-homogeneous teams in congested video game environments," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, 2017, pp. 270–272.

[6] J. Snape, S. J. Guy, M. C. Lin, D. Manocha, and J. Van den Berg, "Reciprocal collision avoidance and multi-agent navigation for video

games," in *MAPF@AAAI*, 2012. [Online]. Available: `https://api.semanticscholar.org/CorpusID:10768014`.

[7] J. Li, T. A. Hoang, E. Lin, H. L. Vu, and S. Koenig, "Intersection coordination with priority-based search for autonomous vehicles," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, pp. 11 578–11 585, Jun. 2023. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/26368`.

[8] C. R. Goenawan, *ASTM : Autonomous smart traffic management system using artificial intelligence CNN and LSTM*, 2025. [Online]. Available: `https://arxiv.org/abs/2410.10929`.

[9] R. Morris, C. S. Păsăreanu, K. S. Luckow, W. A. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations," in *AAAI Workshop: Planning for Hybrid Systems*, 2016, pp. 608–614.

[10] J. Li, M. Gong, Z. Liang, W. Liu, Z. Tong, L. Yi, R. Morris, C. Pasareanu, and S. Koenig, "Departure scheduling and taxiway path planning under uncertainty," in *AIAA Aviation 2019 Forum*. [Online]. Available: `https://arc.aiaa.org/doi/abs/10.2514/6.2019-2930`.

[11] M. Abdull Redha and H. S. Abdulameer, "Multi-agent systems and swarm intelligence for autonomous drone coordination," *Journal of Engineering and Computer Sciences*, vol. 5, pp. 1–7, Aug. 2025.

[12] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal, "CoBots: Robust symbiotic autonomous mobile service robots," in *Twenty-fourth international joint conference on artificial intelligence*, Citeseer, AAAI Press, 2015, pp. 4423–4429.

[13] B. Coltin and M. Veloso, "Online pickup and delivery planning with transfers for mobile robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 5786–5791.

[14] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 11 272–11 281.

[15] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 151–158, Sep. 2021. [Online]. Available: https://ojs.aaai.org/index.php/SOCS/article/view/18510.

[16] A. Andreychuk, K. Yakovlev, P. Surynek, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artificial Intelligence*, vol. 305, p. 103 662, 2022.

[17] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, *Lifelong multi-agent path finding for online pickup and delivery tasks*, 2017. [Online]. Available: https://arxiv.org/abs/1705.10868.

[18] A. Erath, M. Löchl, and K. W. Axhausen, "Graph-theoretical analysis of the swiss road and railway networks over time," *Networks and Spatial Economics*, vol. 9, no. 3, pp. 379–400, 2009.

[19] A. Majeed and I. Rauf, "Graph theory: A comprehensive survey about graph theory applications in computer science and social networks," *Inventions*, vol. 5, no. 1, 2020, ISSN: 2411-5134. [Online]. Available: https://www.mdpi.com/2411-5134/5/1/10.

[20] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to WordNet: An on-line lexical database*," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, Dec. 1990, ISSN: 0950-3846. [Online]. Available: https://doi.org/10.1093/ijl/3.4.235.

[21] R. N. Mantegna, "Hierarchical structure in financial markets," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 11, no. 1, pp. 193–197, 1999.

[22] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," *ACM Comput. Surv.*, vol. 54, no. 4, Jul. 2021, ISSN: 0360-0300. [Online]. Available: https://doi.org/10.1145/3447772.

[23] L. Page, "Method for node ranking in a linked database," US Patent 7,058,628, Jun. 2006.

[24] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[25] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia, "Learning skillful medium-range global weather forecasting," *Science*, vol. 382, no. 6677, pp. 1416–1421, 2023. [Online]. Available: `https://www.science.org/doi/abs/10.1126/science.adi2336`.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[28] C. K. Joshi, *Transformers are graph neural networks*, 2025. [Online]. Available: `https://arxiv.org/abs/2506.22084`.

[29] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[30] P. E. Hart, N. J. Nilsson, and B. 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.

[31] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, 2013, pp. 1443–1449.

[32] A. Li, Z. Chen, M. Vered, and D. Harabor, *Revisiting conflict based search with continuous-time*, 2025. [Online]. Available: `https://arxiv.org/abs/2501.07744`.

[33] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 5628–5635.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, 2022, ISBN: 9780262046305.

[35] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[36] Z. C. Dagdia and M. Mirchev, "Chapter 15 - when evolutionary computing meets astro- and geoinformatics," in *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, P. Škoda and F. Adam, Eds., Elsevier, 2020, pp. 283–306, ISBN: 978-0-12-819154-5. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780128191545000266`.

[37] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Second. Prentice Hall, 2003, ISBN: 0137903952.

[38] A. Botea and P. Surynek, "Multi-agent path finding on strongly biconnected digraphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[39] B. Nebel, "The small solution hypothesis for MAPF on strongly connected directed graphs is true," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, 2023, pp. 304–313.

[40] I. Saccani, S. Ardizzoni, L. Consolini, and M. Locatelli, *Dynamic programming based local search approaches for multi-agent path finding problems on directed graphs*, 2024. [Online]. Available: `https://arxiv.org/abs/2410.07954`.

[41] H. Ma, "Graph-based multi-robot path finding and planning," *Current Robotics Reports*, vol. 3, no. 3, pp. 77–84, 2022.

[42] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hoenig, T. K. S. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon, *Overview: Generalizations of multi-agent path finding to real-world scenarios*, 2017. [Online]. Available: `https://arxiv.org/abs/1702.05515`.

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

[44] H. Ma, G. Wagner, A. Felner, J. Li, T. K. S. Kumar, and S. Koenig, *Multi-agent path finding with deadlines*, 2018. [Online]. Available: `https://arxiv.org/abs/1806.04216`.

[45] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7643–7650, Jul. 2019. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/4758`.

[46] F. Semiz, M. A. Yorgancı, and F. Polat, "Solving an industry-inspired generalization of lifelong MAPF problem including multiple delivery locations," *Advanced Engineering Informatics*, vol. 57, p. 102 026, 2023.

[47] S. Riazi, K. Bengtsson, and B. Lennartson, "Energy optimization of large-scale AGV systems," *IEEE Transactions on automation science and engineering*, vol. 18, no. 2, pp. 638–649, 2020.

[48] S. F. Roselli, M. Fabian, and K. Åkesson, "Solving the conflict-free electric vehicle routing problem using SMT solvers," in *2021 29th mediterranean conference on control and automation (MED)*, IEEE, 2021, pp. 542–547.

[49] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding and executing," *Journal of Artificial Intelligence Research*, vol. 67, pp. 549–579, 2020.

[50] D. Atzmon, R. Stern, A. Felner, N. R. Sturtevant, and S. Koenig, "Probabilistic robust multi-agent path finding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 29–37.

[51] P. Surynek, "Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 12 409–12 417.

[52] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[53] J. Morag, N. Gabay, D. koyfman, and R. Stern, *Transient multi-agent path finding for lifelong navigation in dense environments*, 2024. [Online]. Available: `https://arxiv.org/abs/2412.04256`.

[54] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 9964–9971.

[55] L. Bonalumi, B. Flammini, D. Azzalini, and F. Amigoni, "Multi-agent pickup and delivery with external agents," *Robotics and Autonomous Systems*, vol. 191, p. 105 000, 2025.

[56] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[57] Y. Tang, Z. Yu, Y. Zheng, T. K. S. Kumar, J. Li, and S. Koenig, *Enhancing lifelong multi-agent path finding with cache mechanism*, 2025. [Online]. Available: `https://arxiv.org/abs/2501.02803`.

[58] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 3260–3267.

[59] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013, ISSN: 0004-3702. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0004370212001543`.

[60] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015, ISSN: 0004-3702. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0004370214001386`.

[61] K. Okumura, "LaCAM: Search-based algorithm for quick multi-agent pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, pp. 11 655–11 662, Jun. 2023. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/26377`.

[62] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[63] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "PRIMAL$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.

[64] X. Li, J. Gao, and Y. Li, "Multi-agent path finding based on graph neural network," in *2023 42nd Chinese Control Conference (CCC)*, 2023, pp. 5458–5463.

[65] A. Skrynnik, A. Andreychuk, M. Nesterova, K. Yakovlev, and A. Panov, "Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, pp. 17 541–17 549, Mar. 2024. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/29704`.

[66] A. Andreychuk, K. Yakovlev, A. Panov, and A. Skrynnik, "MAPF-GPT: Imitation learning for multi-agent pathfinding at scale," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 22, pp. 23 126–23 134, Apr. 2025. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/34477`.

[67] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proceedings of the international symposium on combinatorial Search*, vol. 5, 2014, pp. 19–27.

[68] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Meta-agent conflict-based search for optimal multi-agent path finding," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 3, 2012, pp. 97–104.

[69] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "ICBS: The improved conflict-based search algorithm for multi-agent pathfinding," *Proceedings of the International Symposium on Combinatorial Search*, vol. 6, no. 1, pp. 223–225, Sep. 2021. [Online]. Available: `https://ojs.aaai.org/index.php/SOCS/article/view/18343`.

[70] J. Lim and P. Tsiotras, "CBS-Budget (CBSB): A complete and bounded suboptimal search for multi-agent path finding," *Artificial Intelligence*, vol. 346, p. 104 349, 2025, ISSN: 0004-3702.

[71] M. Tang, Y. Li, H. Liu, Y. Chen, M. Liu, and L. Wang, *MGCBS: An optimal and efficient algorithm for solving multi-goal multi-agent path finding problem*, 2024. [Online]. Available: `https://arxiv.org/abs/2404.19518`.

[72] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103 752, 2022, ISSN: 0004-3702. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0004370222000923`.

[73] J. Morag, Y. Zhang, D. Koyfman, Z. Chen, A. Felner, D. Harabor, and R. Stern, "Prioritised planning: Completeness, optimality, and complexity," *Journal of Artificial Intelligence Research*, vol. 84, 2025.

[74] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1, pp. 477–521, Nov. 1987, ISSN: 1432-0541. [Online]. Available: `https://doi.org/10.1007/BF01840371`.

[75] D. Silver, "Cooperative pathfinding," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 1, no. 1, pp. 117–122, Sep. 2021. [Online]. Available: `https://ojs.aaai.org/index.php/AIIDE/article/view/18726`.

[76] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald, "Hierarchical A*: Searching abstraction hierarchies efficiently," in *AAAI/IAAI, Vol. 1*, 1996, pp. 530–535.

[77] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI conference on Artificial Intelligence*, vol. 33, 2019, pp. 7643–7650.

[78] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *International Joint Conference on Artificial Intelligence 2021*, Association for the Advancement of Artificial Intelligence (AAAI), 2021, pp. 4127–4135.

[79] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming — CP98*, M. Maher and J.-F. Puget, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 417–431, ISBN: 978-3-540-49481-2.

[80] J. Li, W. Ruml, and S. Koenig, "EECBS: A bounded-suboptimal search for multi-agent path finding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, pp. 12 353–12 362, May 2021. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/17466`.

[81] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 10 256–10 265.

[82] Y. Zhang, Z. Chen, D. Harabor, P. Le Bodic, and P. J. Stuckey, "Planning and execution in multi-agent path finding: Models and algorithms," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 707–715.

[83] Y. Zhang, Z. Chen, D. Harabor, P. Le Bodic, and P. J. Stuckey, "Concurrent planning and execution in lifelong multi-agent path finding with delay probabilities," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, 2025, pp. 23 387–23 394.

[84] K. Okumura and X. Défago, "Solving simultaneous target assignment and path planning efficiently with time-independent execution," *Artificial Intelligence*, vol. 321, p. 103 946, 2023.

[85] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[86] R. Veerapaneni, M. S. Saleem, J. Li, and M. Likhachev, "Windowed MAPF with completeness guarantees," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, 2025, pp. 23 323–23 332.

[87] Y. Wu, R. Veerapaneni, J. Li, and M. Likhachev, *From space-time to space-order: Directly planning a temporal planning graph by redefining CBS*, 2024. [Online]. Available: `https://arxiv.org/abs/2404.15137`.

[88] R. Pugliese, S. Regondi, and R. Marini, "Machine learning-based approach: Global trends, research directions, and regulatory standpoints," *Data Science and Management*, vol. 4, pp. 19–29, 2021.

[89] J.-M. Alkazzi and K. Okumura, "A comprehensive review on leveraging machine learning for multi-agent path finding," *IEEE Access*, vol. 12, pp. 57 390–57 409, 2024.

[90] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022.

[91] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Readings in Cognitive Science*, A. Collins and E. E. Smith, Eds., Morgan Kaufmann, 1988, pp. 399–421, ISBN: 978-1-4832-1446-7.

[92] M. I. Jordan, "Chapter 25 - serial order: A parallel distributed processing approach," in *Neural-Network Models of Cognition*, ser. Advances in Psychology, J. W. Donahoe and V. Packard Dorsel, Eds., vol. 121, North-Holland, 1997, pp. 471–495. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0166411597801112`.

[93] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7627–7634, Jul. 2019. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/4756`.

[94] R. Wu, M. Zhang, S. Wang, F. K. S. Chan, Y. N. Law, and L. Li, "Continuous lifelong conflict-aware AGV routing with kinematic constraints," *Proc. VLDB Endow.*, vol. 18, no. 7, pp. 2254–2267, Mar. 2025, ISSN: 2150-8097. [Online]. Available: `https://doi.org/10.14778/3734839.3734859`.

[95] K. Kasaura, M. Nishimura, and R. Yonetani, "Prioritized safe interval path planning for multi-agent pathfinding with continuous time on 2d roadmaps," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 494–10 501, 2022.

[96] T. T. Walker and N. R. Sturtevant, *Collision detection for agents in multi-agent pathfinding*, 2019. [Online]. Available: `https://arxiv.org/abs/1908.09707`.

[97] C. Ericson, *Real-Time Collision Detection*. USA: CRC Press, Inc., 2004, ISBN: 1558607323.

[98] Bentley and Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Transactions on Computers*, vol. C-28, no. 9, pp. 643–647, 1979.

[99] T. T. Walker, N. R. Sturtevant, and A. Felner, "Clique analysis and bypassing in continuous-time conflict-based search," *Proceedings of the International Symposium on Combinatorial Search*, vol. 17, no. 1, pp. 152–160, Jun. 2024. [Online]. Available: `https://ojs.aaai.org/index.php/SOCS/article/view/31553`.

[100] W. J. Tan, X. Tang, and W. Cai, "Robust multi-agent pathfinding with continuous time," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 34, 2024, pp. 570–578.

[101] K. Yakovlev, A. Andreychuk, and R. Stern, "Optimal and bounded suboptimal any-angle multi-agent pathfinding," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 7996–8001.

[102] P. Surynek, "Continuous multi-agent path finding via satisfiability modulo theories (SMT)," in *International Conference on Agents and Artificial Intelligence*, Springer, 2020, pp. 399–420.

[103] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, C. Boutilier, Ed., 2009, pp. 399–404. [Online]. Available: `http://ijcai.org/Proceedings/09/Papers/074.pdf`.

[104] T. Kolárik, S. Ratschan, and P. Surynek, *Multi-agent path finding with continuous time using SAT modulo linear real arithmetic*, 2023. [Online]. Available: `https://arxiv.org/abs/2312.08051`.

[105] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, N. Piterman and S. A. Smolka, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 93–107, ISBN: 978-3-642-36742-7.

[106] T. T. Walker, N. R. Sturtevant, and A. Felner, "Extended increasing cost tree search for non-unit cost domains," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 534–540. [Online]. Available: `https://doi.org/10.24963/ijcai.2018/74`.

[107] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3.

[108] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *25th Annual Symposium on Foundations of Computer Science*, 1984, pp. 241–250.

[109] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, Jul. 2019, pp. 39–45. [Online]. Available: `https://doi.org/10.24963/ijcai.2019/6`.

[110] P. Surynek, "Multi-agent path finding with continuous time and geometric agents viewed through satisfiability modulo theories (SMT)," *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 200–201, Sep. 2021. [Online]. Available: `https://ojs.aaai.org/index.php/SOCS/article/view/18490`.

[111] W. Hoenig, T. K. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," *Proceedings of the International Conference on Automated Planning and*

Scheduling, vol. 26, no. 1, pp. 477–485, Mar. 2016. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/13796.

[112] S. F. Roselli, Z. Zhang, and K. Åkesson, *Combining high level scheduling and low level control to manage fleets of mobile robots*, 2025. [Online]. Available: https://arxiv.org/abs/2510.23129.

[113] M. C. Angelini and F. Ricci-Tersenghi, "Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set," *Nature Machine Intelligence*, vol. 5, no. 1, pp. 29–31, Jan. 2023, ISSN: 2522-5839. [Online]. Available: https://doi.org/10.1038/s42256-022-00589-y.

[114] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, ISSN: 1476-4687. [Online]. Available: https://doi.org/10.1038/nature24270.

[115] R. Jain, K. Okumura, M. Amir, and A. Prorok, *Graph attention-guided search for dense multi-agent pathfinding*, 2025. [Online]. Available: https://arxiv.org/abs/2510.17382.

[116] G. Wang, J. Li, Y. Sun, X. Chen, C. Liu, Y. Wu, M. Lu, S. Song, and Y. A. Yadkori, *Hierarchical reasoning model*, 2025. [Online]. Available: https://arxiv.org/abs/2506.21734.

[117] D. Kahneman, *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011, ISBN: 9780374275631.

[118] A. Jolicoeur-Martineau, *Less is more: Recursive reasoning with tiny networks*, 2025. [Online]. Available: https://arxiv.org/abs/2510.04871.

[119] S. Lee, W. Sim, D. Shin, W. Seo, J. Park, S. Lee, S. Hwang, S. Kim, and S. Kim, "Reasoning abilities of large language models: In-depth analysis on the abstraction and reasoning corpus," *ACM Transactions on Intelligent Systems and Technology*, 2025, ISSN: 2157-6904. [Online]. Available: https://doi.org/10.1145/3712701.