Advancements in Understanding the Empirical Hardness of the Multi-Agent Pathfinding Problem

by

Jingyao Ren

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2024

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Nora Ayanian, for her support and kindness throughout my PhD journey. I am especially grateful for the freedom she gave me to pursue the research that truly excites me. Her trust and guidance have been invaluable, and I feel incredibly fortunate to have had her as my advisor. Beyond my academic pursuits, her support extended to all aspects of my life — she was always open to listening to any challenges I faced and ready to offer help. She has done so much for me, and I will always be grateful.

I am incredibly fortunate and grateful to work closely with Sven Koenig and T. K. Satish Kumar. Sven's exceptional kindness, thoughtful advice, and inspiring vision for research have been a constant source of support and motivation, making my PhD journey much smoother and more rewarding. His perspective on research has had a profound impact on me. Satish's insights and enthusiasm for research always leave me inspired and energized after every meeting. They are the people with whom I feel completely comfortable sharing any ideas I'm excited about, without hesitation. I deeply appreciate their mentorship, encouragement, and the opportunity to contribute to a relatively new area of research.

I sincerely thank my other committee members, Stefanos Nikolaidis, and Feifei Qian, for their time and valuable feedback which have inspired new research ideas beyond the scope of my thesis.

A very special thanks to my best research buddy, Eric Ewing, with whom I truly enjoyed working. Eric is the best collaborator that I could dream of. He is the first person I think of whenever I have new ideas to share, and his brilliance and dedication make our collaboration flow effortlessly—smooth like butter.

I would also like to extend my heartfelt gratitude to all my collaborators and the ACT-Lab community. To Wolfgang Honig, Elizabeth Borroson, Kegan Strawn, Baskin Senbaslar, Vikraman Sathiyanarayanan, and Connie Zhang, thank you for being incredibly kind, helpful, and inspiring lab mates who have contributed to and supported my research in so many ways. I am equally grateful to the ACT-Lab members at Brown University — Lishuo Pan, Anoop Kiran, Arjun Prakash, Tabitha Oanda, Will Wu, Stephen Crawford, and Jacqueline Dowling. Thanks for all the valuable feedback on my research and talks. I truly enjoyed my visit to Brown as well.

I want to thank Kevin Tang, Jenny Sabin, Kirstin Peterson, and Joseph Skovira at Cornell University, who inspired me to pursue a PhD and also made this journey possible in so many ways.

To my beloved roommates, Zihang Cheng, Hefei Liu, and Yang Cai: thank you for the countless memories during the pandemic.

To my best friend, Qingkai Zeng: our bond runs so deep that words like *thank you* feel almost unnecessary.

To my family, especially my mom and dad — your unconditional love means everything to me.

Table of Contents

List of Tables

List of Figures

Abbreviations

MAPF	Multi Agent Pathfinding Problem
SAT	Boolean Satisfiability Problem
CSP	Constraint Satisfaction Problem
TSP	Traveling Salesman Problem
MIP	Mixed Integer Programming
CBS	Conflict Based Search
BCP	Branch and Cut and Price
MAPFAST	Multi-Agent Path Finding Algorithm SelecTor
CNN	Convolutional Neural Network
QD	Quality Diversity
CA	Cellular Automata
DLA	Diffusion-Limited Aggregation
CMA-MAE	Covariance Matrix Adaptation MAP-Annealing

Abstract

Multi-Agent Path Finding (MAPF) is the problem of finding collision-free paths for a team of agents in a shared environment. This fundamental problem in artificial intelligence and robotics has numerous real-world applications, such as automated warehouses, trajectory planning, and swarm control. While solving MAPF optimally is NP-hard, existing algorithms can still solve many large real-world instances efficiently. Hard problems can still have easy instances. However, the factors that affect the instance hardness remain unclear, and the performance of MAPF algorithms varies significantly, with no single algorithm consistently outperforming others across all instances.

In this dissertation, we study the empirical hardness of MAPF, which aims to understand how and why the hardness of solving different MAPF instances varies based on the instance features. We investigate the following key questions: What makes a MAPF instance hard? Can we predict the instance hardness without solving it? How can we generate hard instances by manipulating the instance features?

We have made several key contributions to the empirical hardness study of MAPF. First, we establish a new research direction in MAPF by formalizing the study of empirical hardness, addressing key challenges, and proposing preliminary ideas and future research directions. Second, we present MAPFAST, an algorithm selection framework to predict the empirical hardness and help select the best algorithm on a given instance. Third, we provide theoretical and experimental evidence that map connectivity is a key factor influencing empirical hardness. Additionally, we demonstrate a method for generating MAPF instances with varying hardness by manipulating the map connectivity.

Chapter 1

Introduction

Multi-Agent Path Finding (MAPF) [155] is the problem of finding collision-free paths for a team of agents from their start locations to designated goal locations in a shared environment. MAPF is a fundamental problem in robotics and artificial intelligence and plays a crucial role in nearly every aspect of multi-robot systems. The ability of robots to move freely without conflicts is fundamental across all multi-robot applications. Regardless of specific objectives or domains, the robots must first navigate without collisions before performing any further tasks. With the rapid advancement and increasing deployment of real-world robots in various domains, understanding and solving MAPF problems has become critical for developing robust and efficient multi-robot systems.

MAPF has been applied to a wide range of real-world applications. For example, in automated warehouses, large teams of robots must navigate the environment while avoiding conflicts with each other [54, 100, 95, 166]. In addition to basic navigation, the robots often need to perform different tasks, such as picking up items and delivering them to certain destinations. This problem is known as Pickup and Delivery and requires balancing path planning with task allocation and scheduling. It introduces additional complexity to MAPF since we need to compute collision-free paths while optimizing other factors such as maximizing throughput or prioritizing specific tasks [96, 20, 170, 132]. MAPF is also essential beyond automated warehouses. For instance, in trajectory planning for unmanned-aerial-vehicle (UAV) [55, 102,

56], it is crucial to navigate multiple drones flying in a shared environment without collisions. Unlike ground robots, UAVs operate in 3D space and often have more complex dynamics. Various environmental factors, such as wind, altitude, or air pressure can also significantly impact the calculated trajectories, thus making it a more challenging problem. To tackle this, an initial discrete plan is often computed first and then iteratively smoothed into a continuous plan while considering additional dynamics and other external factors. MAPF has also been applied to tasks that require the cooperation of robots for collective goals, such as maintaining team formations in swarm control [94, 103] or searching dangerous areas and rescuing survivors [64, 30]. Additionally, MAPF has been used to navigate virtual characters in computer games [101, 143] and coordinate vehicles in traffic control systems [53, 112, 53].

The MAPF algorithms aim to find collision-free paths for a team of robots while minimizing a specific objective function, such as the makespan (i.e., the time when the last robot reaches its goal) or sum-of-cost (i.e., the sum of the path length for all robots). The choice of objective functions depends on the specific objectives of different applications. For delay-sensitive applications, it is crucial to optimize the makespan. This is especially important for a pick-up-and-delivery system, where delays usually lead to congestion and might decrease the overall throughput. In other applications where minimizing power consumption or total distance traveled is more important, optimizing on sum-of-cost becomes a better choice. For instance, when a team of UAVs is performing surveillance tasks in a forest, a dead battery can not only lead to the loss of physical robots but also more power consumption as the remaining UAVs need to reorganize to cover additional areas.

The MAPF problems can be solved optimally or sub-optimally. Optimal MAPF algorithms provide the best possible solutions, whereas sub-optimal MAPF algorithms trade solution quality for faster runtime. Bounded sub-optimal algorithms guarantee to give solutions no more than a scale factor of the optimal solution [10]. This gives us a chance to balance the runtime and solution quality. Unbounded algorithms have no guarantee of the quality of the solution and focus mainly on solving speed [120]. In this dissertation,

we focus on *optimal* MAPF algorithms, as they offer the best solution quality and are more fundamental to the core of the MAPF problem, providing a deeper understanding of its complexities.

Solving MAPF problems optimally for makespan or sum-of-cost is known to be NP-hard [173], even for planar graph [171] and grid-based problems [9]. This means that the worst-case running times increase exponentially w.r.t. the problem size (e.g., total number of agents). Over the past years, many optimal MAPF algorithms have been developed using various techniques. Most early MAPF solvers use A* [50] based algorithms [153, 34, 47] but suffer from large expansion factors and have poor scalability. Many modern MAPF algorithms are based on Conflict-Based Search (CBS), a two-level search algorithm that iteratively resolves the conflicts between the agents [137, 16, 35]. The performance of CBS-based algorithms has been further improved by various new heuristics and branching techniques [90, 93, 92]. Another approach is compilation-based where MAPF is converted to a different problem such as Boolean Satisfiability (SAT) [158] or Mixed Integer Programming (MIP) [79, 78]. Compilation-based algorithms benefit greatly from the advanced, off-the-shelf solver techniques since the converted problems have been studied more extensively than MAPF itself. It is easy to see that these diverse techniques can lead to different strengths and weaknesses, ultimately leading to different runtimes across different MAPF instances. This indicates that different algorithms may be better suited to different instances and leads to a new area of study known as empirical hardness.

1.1 Empirical Hardness

For many computationally hard problems, such as NP-complete problems, it is well-known that the worst-case complexity can grow exponentially as the problem size increases (e.g., the number of agents for MAPF). However, a significant number of instances of these hard problems can be solved efficiently using existing algorithms. One reason is that real-world instances often represent a narrow range of the overall hardness. For instance, pickup-and-delivery problems often have fixed goal locations rather than randomly

distributed on the entire map. Moreover, it is often unclear why certain groups or families of instances are harder or easier than others. For example, automated warehouses tend to have harder MAPF instances than randomly generated maps. These observations lead to the *Empirical Hardness* research, which focuses on instance-specific hardness rather than the theoretical worst case. More formally, empirical hardness seeks to understand the relationship between the features of problem instances and the hardness of solving them with specific algorithms.

The study of empirical hardness covers a wide range of topics and has seen tremendous success in many NP-complete problems, including but not limited to Boolean satisfiability (SAT) [17], Constraint Satisfaction (CSP) [164], and Traveling Salesman problem (TSP) [43]. Early research seeks to find the correlations between instance features and empirical hardness. Some key questions are: What factors make an instance hard or easy? Is it possible to effectively measure the instance hardness without solving it? Can we generate instances of desired hardness by manipulating certain instance features? For example, research has shown that the clause-to-variable ratio of the SAT problem is correlated with the instance hardness [107]. This clause-to-variable ratio was later used as a controllable parameter to help generate hard instances [135].

Another line of research seeks to leverage the strengths and weaknesses of different algorithms. This stems from the fact that different algorithms tend to outperform each other and there is no overall winner for all problem instances. Selecting the best algorithm on a case-by-case basis will improve the overall performance such as reducing runtime or memory usage. This area of study, known as *Algorithm Selection*, focuses on automatically predicting the best suitable algorithm from a pre-defined portfolio based on different instances. It is often solved as a prediction problem using machine learning techniques. For instance, SATzilla [169] trained an empirical hardness model based on instance features to predict the runtime of different SAT solvers. Other common topics in empirical hardness include algorithm configuration [60], backbone [110], backdoor [164], and phase transition [17]. The study of empirical hardness provides a

deeper understanding of problem difficulty and has driven significant advancements in designing new heuristics and algorithms. We will provide a detailed introduction in Chapter 3.

The empirical hardness of MAPF is a less studied area. One reason is that MAPF has more complex instance features than other iconic NP-complete problems such as SAT or CSP. The inherent multi-agent nature and spatial environment have made studying its empirical hardness particularly challenging. Moreover, MAPF research has primarily focused on developing new algorithms and novel heuristics. While this focus has led to significant advancement in solving MAPF problems, it leaves the empirical hardness of MAPF understudied. In this dissertation, we seek to address this gap by shifting the focus toward understanding and leveraging the empirical hardness of MAPF instances. More specifically, we seek to develop algorithm selection models to predict the most effective algorithm based on specific instances. Additionally, we aim to investigate and identify the key instance features that influence the empirical hardness of MAPF problems.

1.2 Motivation

We further elaborate on our motivations to study the empirical hardness of the MAPF problem.

Motivation #1. When to use which algorithm?

MAPF algorithms use a wide variety of techniques and heuristics, thus resulting in distinct strengths and weaknesses. The runtime of different optimal MAPF algorithms outperforms each other on different MAPF instances and there is no single best algorithm according to many MAPF benchmark studies [33, 139, 68]. Some algorithms tend to perform well on maps with specific layouts. For example, Figure 1.1 shows the percentage of instances where each algorithm runs faster than other algorithms. We can see that CBSH [35] performs well on fractal maps while LazyCBS [41] favors warehouse maps. Figure 1.1 also shows the percentage of instances each algorithm solved within the time limit. Although SMT-CBS [157]

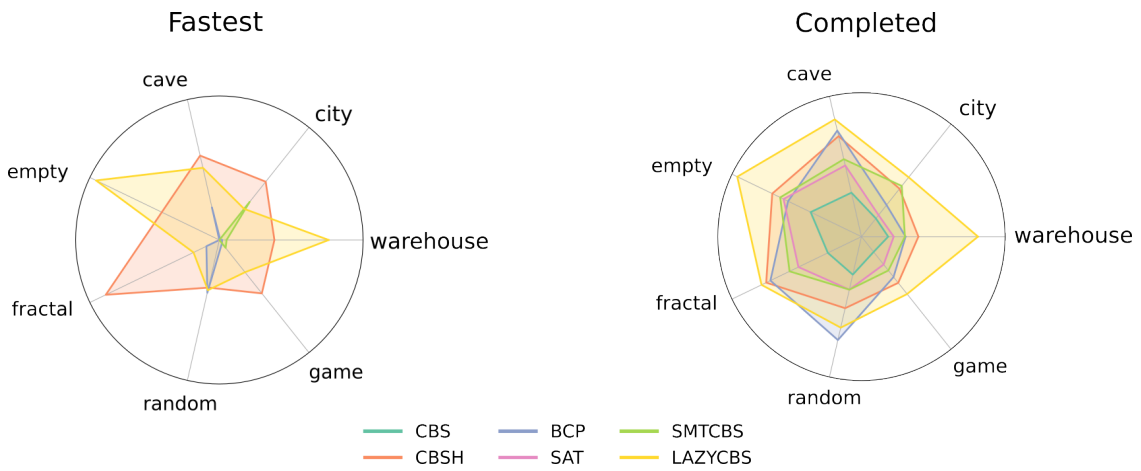


Figure 1.1: Left: percentage of the instances where an algorithm runs the fastest. Right: percentage of the instances where an algorithm successfully solved within the time limit.

solved significantly fewer problems than LazyCBS, it still manages to run faster on a subset of city maps compared to other algorithms. This suggests that different algorithms should be used for different types of instances to achieve better runtime.

Table. 1.1 shows the detailed performance data for game maps, with Oracle representing the result of always choosing the fastest algorithm. Oracle completed 55.43% of the total instances in 2670 minutes, while the best single algorithm, LazyCBS, completed 47.92% in 3180 minutes. Using the best algorithm for each instance will not only solve more instances but also significantly reduce the total runtime. This approach is particularly useful in the context of online MAPF systems, where new tasks and agents continually arrive and frequent replannings are needed. The best algorithm to use may vary over time as the system evolves. Choosing the best algorithm for different instances will no doubt save a considerable amount of runtime.

Instead of choosing the best algorithm on a case-by-case basis, another approach is to run different algorithms in parallel and choose whichever algorithm finishes the first. However, this approach is resource-intensive and computationally costly, especially for large algorithm portfolios. Rather than running ten algorithms in parallel to solve one instance, we could always solve ten different instances in parallel by predicting the best possible algorithms accordingly.

Table 1.1: Algorithm performance data for game maps. *Fastest* is the percentage of instances where an algorithm outperformed others in runtime. *Completed* is the percentage of instances an algorithm solved within the time limit.

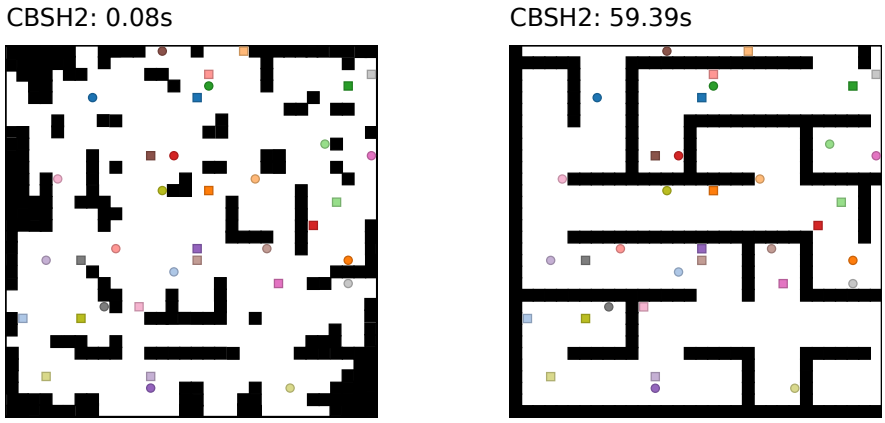
Algorithm	Fastest (%)	Completed (%)	Runtime (min)
CBS	0.00%	12.77%	4899
CBSH	**55.23%**	38.59%	3452
BCP	3.43%	33.70%	3772
SAT	0.00%	23.28%	4498
SMT-CBS	8.17%	28.26%	4113
LazyCBS	33.17%	**47.92%**	**3180**
Oracle	**100.00%**	**55.43%**	**2670**

In this dissertation, we aim to develop algorithm selection models that predict the best algorithm for different MAPF instances. These models leverage the unique strengths and weaknesses of each algorithm and can significantly enhance the overall performance and efficiency of real-world MAPF systems. By selecting the most effective algorithm for each instance, we can reduce the total computational costs.

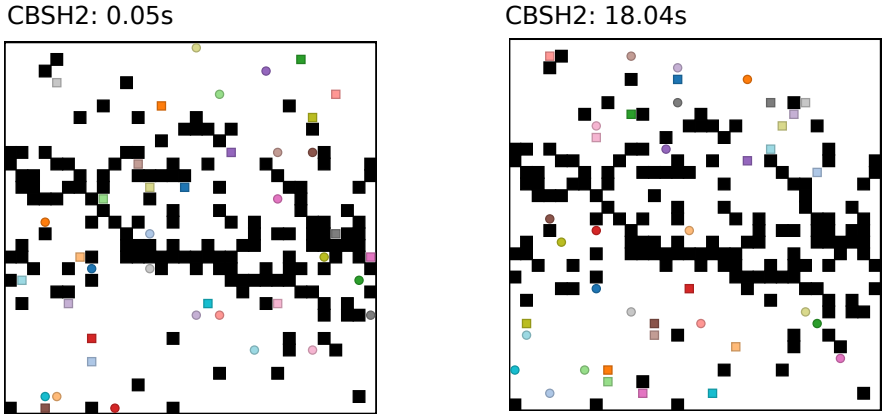
Motivation #2: What makes a MAPF instance hard?

While being NP-hard means MAPF is a very challenging problem, we still see optimal MAPF algorithms solving instances with up to tens or even hundreds of agents within a reasonable amount of time (e.g., several seconds) [139]. It is no secret that computationally hard problems can still have easy instances. Sometimes the real-world instances only cover part of the problem space and such instances are not always the hardest ones. However, we don't fully understand when and why some MAPF instances are significantly easier or harder than others.

A MAPF instance consists of two major components: a map with obstacles and a set of start and goal locations for agents. Both components have a major impact on the hardness of different instances. MAPF instances are complex and must be studied on a case-by-case basis. The influence of these two components on instance hardness can be dramatic. For example, instances with the same start and goal locations of agents can exhibit significantly different hardness on different maps (e.g., Figure 1.2a). This indicates the significant influence of the spatial layout of obstacles on instance hardness. Moreover, even



(a) MAPF instances with the same number of obstacles and same start/goal locations but dramatically different runtime.



(b) MAPF instances with the same map and number of agents but dramatically different runtime.

Figure 1.2: Examples of different MAPF instances. Black squares represent obstacles, circles denote start locations, and squares denote goal locations. Different colors correspond to different agents.

on the same map with the same number of agents, the distributions of start and goal locations can lead to major differences in the hardness as well (e.g., Figure 1.2b). This observation further demonstrates the complex nature of MAPF instances.

In this dissertation, we seek to gain a deeper understanding of the empirical hardness of MAPF and investigate the key instance features influencing empirical hardness. This knowledge could lead to various advancements in MAPF research, such as the generation of hard instances and the enhancement of algorithm selection models.

Motivation #3: How to create a hard MAPF instance?

With the rapid advancement of powerful MAPF algorithms, there is a growing need for more challenging and diverse MAPF instances to thoroughly validate and test these algorithms. Most existing MAPF algorithms are tested using the famous MAPF Benchmark [155]. MAPF Benchmark has 33 maps of various styles such as game, city, or warehouse. Each map has several scenario files, each containing a fixed list of start and goal locations for the agents. To find challenging instances, the authors recommended gradually increasing the number of agents from the scenario files until the algorithm reaches the time limit [155].

There are some obvious limitations of MAPF Benchmark, such as a limited number of maps and fixed agent distributions. There is a great need for diverse maps with varying styles, layouts, and connectivity to better represent the breadth of real-world applications. For example, when testing an algorithm designed for automated warehouses, it is more appropriate to evaluate it in warehouse environments rather than on city maps. Moreover, Multi-Agent Reinforcement Learning (MARL) [3] systems often require training across diverse environments to achieve better generalization and robustness. Another limitation is the fixed distribution of agents, which represents only a narrow subset of possible instances. To thoroughly evaluate the performance of MAPF algorithms, it is important to test with a wider range of agent distributions, reflecting more diverse scenarios and challenges that may occur in real-world applications.

Generating challenging MAPF instances is no easy task without a clear understanding of what makes them hard. Our investigation into these key factors will enable the development of new techniques for generating hard MAPF instances. In this dissertation, we seek to develop effective tools for generating challenging maps for MAPF problems by manipulating the key factors that influence empirical hardness. These hard MAPF instances will also inspire new heuristics or specialized algorithms that are designed for certain subclass of MAPF problems.

Hypothesis

In this dissertation, we aim to address several key questions:

1. What is MAPF empirical hardness and why is it important?
2. Can we compare or predict the empirical hardness of MAPF instances without solving them?
3. What are the key instance features that influence the empirical hardness of MAPF?
4. How to effectively generate hard MAPF instances?

By exploring these crucial questions, we seek to develop a deeper understanding of MAPF empirical hardness. This knowledge could potentially lead to more efficient algorithm selection, better benchmark design, and improved strategies for tackling real-world MAPF scenarios across various applications.

We propose and validate the following hypothesis:

- *The empirical hardness of MAPF instances can be effectively predicted using algorithm selection techniques and map connectivity is a key factor influencing the instance hardness.*

1.3 Contributions and Thesis Structure

This dissertation makes several significant contributions to the empirical hardness research of MAPF:

- We formalize empirical hardness research in MAPF and present the first comprehensive taxonomy, explaining what empirical hardness is and why it is important to MAPF. We provide an in-depth introduction to major research areas and highlight our contributions to algorithm selection and key instance factors influencing hardness. Additionally, for less explored topics such as algorithm configuration, phase transition, and instance generation, we identify existing challenges and propose preliminary solutions along with future directions. Our Study aims to bring more attention to empirical hardness and inspire new MAPF research. More details are in Chapter 3.

- We show it is possible to predict the empirical hardness of MAPF instances without solving them. First, we introduce MAPFAST, a state-of-the-art deep learning framework for algorithm selection. MAPFAST predicts the best algorithm for a given instance from a predefined portfolio. We justify the need for algorithm selectors by conducting extensive benchmark studies of existing algorithms. We empirically show that no single algorithm consistently outperforms others across all instances. We introduce a new instance encoding method using the single-agent shortest path, which significantly improves prediction quality. Additionally, we present a graph-based algorithm selection model that relies solely on single-agent shortest paths. We demonstrate that this approach can achieve performance comparable to MAPFAST, even without the information of map topology. More details are provided in Chapter 4.

- Our theoretical and empirical study reveals the relationship between map connectivity and the empirical hardness of MAPF instances. We show that map connectivity can be effectively measured using the second smallest eigenvalue (known as λ_2) of the normalized Laplacian matrix. Well-connected maps have a larger λ_2 while poorly connected maps have a smaller λ_2. We theoretically show that when the start/goal locations of agents are generated using uniform random sampling, poorly connected maps (smaller λ_2) tend to result in more challenging instances. We then empirically validate the correlation of empirical hardness and λ_2 using multiple state-of-the-art optimal MAPF algorithms. This highlights map connectivity as a key feature influencing the hardness of MAPF instances. More details are in Chapter 5.

- We demonstrate how to use map connectivity to generate challenging MAPF instances. We develop a map generator using the Quality-Diversity method, which is capable of generating maps within a desired range of λ_2 values. This enables us to create maps within a specific range of connectivity and further affect the instance hardness. Additionally, we show that λ_2 can be used to predict empirical hardness and compare its significance with other instance features. More details are in Section 5.7.

This thesis is organized as follows: Chapter 2 provides the necessary background information on MAPF and a detailed survey of optimal MAPF algorithms. Chapter 3 presents our comprehensive taxonomy of empirical hardness research in MAPF. Chapter 4 presents our algorithm selection framework, MAPFAST. Chapter 5 explores the relationship between map connectivity and MAPF instance hardness. Finally, Chapter 6 concludes the thesis, summarizing our findings and discussing potential future research directions.

Chapter 2

Backgrounds

In this chapter, we formalize the definition of the Multi-Agent Pathfinding (MAPF) problem and review existing optimal MAPF algorithms. Additionally, we provide a performance analysis of these algorithms to highlight their strengths and limitations.

2.1 Multi-Agent Path Finding Problem

The multi-agent path finding problem calculates collision-free paths for a team of agents on a graph. A MAPF instance is normally represented as a tuple $< G, s, t >$. G is an undirected graph that represents the environment. $s = \{s_1, \dots s_k\} \rightarrow V$ and $t = \{t_1, \dots t_k\} \rightarrow V$ map the start and goal locations of k agents to the vertices of graph G [154]. In the classical MAPF settings, the time is discrete and viewed as time steps. At each time step, the agents will *move* to one of its available neighbors or *wait* at the current location. The move actions of an agent can be viewed as traversing an edge of G. The possible choices of move actions depend on the configuration of the graph connectivity. For example, on a 4-connected grid graph, the possible actions are $\{\text{UP}, \text{DOWN}, \text{LEFT}, \text{RIGHT}, \text{WAIT}\}$.

There are two ways to represent a plan of agent. One is to use a sequence of actions π. Let $\pi_i[x]$ denote the location of agent i after x actions. We have $\pi_i[0] = s_i$ since all the agents will start from their initial

Figure 2.1: Example of two types of conflicts.

locations. For a sequence of actions π_i, if $\pi_i[|\pi_i|] = t_i$, then agent i has reached its goal locations and we call this sequence of actions a single-agent plan [155].

The other way is to use the path. A path p_i is a sequence of vertices of graph G which indicates the location of agents i at a certain time step. For example, if $p_i[t] = v_i$ and $p_i[t+1] = v_j$, agent i will move from vertex v_i to v_j by traversing the edge $e_{ij} \in E$. When the agents need to take an action, we need to make sure there are no conflicts with other agents.

There are two types of conflicts in the classical MAPF problem:

- **Vertex Conflict:** When two agents try to occupy the same vertex of the G at the same time step, a vertex conflict will occur. For instance, $p_i[t+1] = v_i$ and $p_j[t+1] = v_i$ will cause a vertex conflict between a_i and a_j since both of them try to occupy v_i at the same time step. We use the tuple $< a_i, a_j, v_i, t+1 >$ to represent a vertex conflict, where a_i and a_j have conflict at vertex v_i at time step $t+1$.

- **Edge Conflict:** When two agents try to traverse the same edge at the same time step, an edge conflict will occur. For example, $p_i[t] = v_i, p_i[t+1] = v_j$ and $p_j[t] = v_j, p_j[t+1] = v_i$ will cause an edge conflict since both agents will try to traverse edge e_{ij} at the same time step. We use the tuple $< a_i, a_j, v_i, v_j, t+1 >$ to represent an edge conflict, when a_i and a_j attempt to traverse the same edge e_{ij} at time step $t+1$.

Figure 2.1 shows an example of two different types of conflicts. Depending on the specific variants of MAPF, the types of conflicts can be further expanded to include additional scenarios, such as following conflicts, cycle conflicts, or swapping conflicts [155].

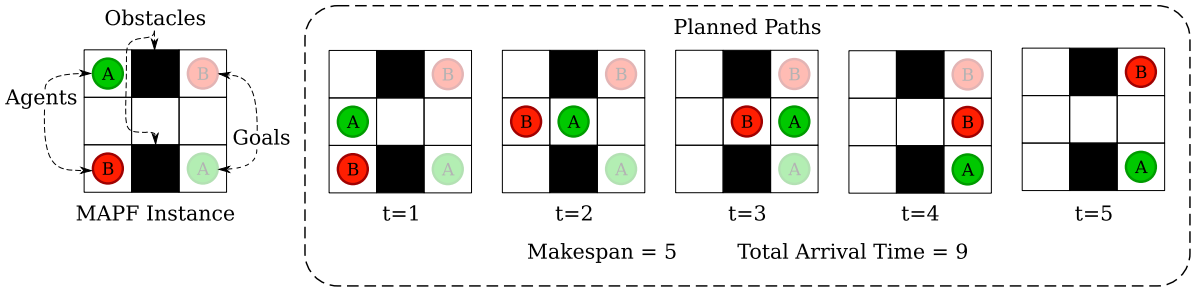


Figure 2.2: Example of a grid-based MAPF instance with two agents.

There are two major objectives for optimal MAPF problem, *makespan* and *sum-of-cost*:

- **Makespan:** The makespan is the time step when the last agent reaches its goal location: $\max_{\forall p_i \in P} |p_i|$, where $|p_i|$ represents the length of path p_i.
- **Sum-of-Cost:** The sum-of-cost is the total path cost of all the agents: $\sum_{\forall p_i \in P} |p_i|$.

A *solution* of a MAPF instance is a set of conflict-free paths $P = \{p_1, p_2 \dots p_n\}$ while minimizing the *makespan* or *sum-of-cost*. Figure 2.2 shows an example MAPF instance with 2 agents. The makespan of this instance is 5 and the sum-of-cost is 9.

In the scope of this dissertation, we assume that the graph G is a 4-connected grid graph and the time is discretized into time step. Our goal is to find collision-free paths while minimizing the *sum-of-cost*.

2.2 Review of Optimal MAPF Algorithms

In this section, we present a comprehensive review of existing optimal MAPF algorithms. Optimal MAPF algorithms provide the best possible solutions by minimizing the objective (e.g., sum-of-cost). We adopt a categorization approach similar to the one proposed by Stern [154], classifying the algorithms based on the high-level frameworks or novel techniques being used. However, it is important to note that algorithms within a specific category may also incorporate techniques from other categories, especially for newly developed algorithms.

2.2.1 Search Based Algorithm

Many early MAPF algorithms search the paths in a coupled fashion where the paths for each agent are planned together in a joint space. Most of these search-based algorithms utilize the A* algorithm [50] to find optimal paths. These joint-space methods suffered a lot from the branching factors and have very poor scalability. For a k-connected graph with n agents, the branching factor of search space is $O(k^n)$, which grows exponentially w.r.t. the number of agents.

A*+OD+ID Standley proposed Operator Decomposition (OD) [153] to handle the branching factor problem. In OD, only one agent's actions are considered at each branch. This effectively reduces the branching factor to just one agent's state space. An operator consists of a move action assigned to an agent. The move actions for the remaining agents will be decided as tree node descendants, thus increasing the solution depth. Although OD increases the solution depth, this trade-off is beneficial since heuristic functions in MAPF can effectively avoid expanding unnecessary subtrees. Another technique is Independence Detection (ID) [153], which partitions agents into disjoint sub-groups such that the optimal paths for each sub-group do not interfere with each other.

EPEA* Another limitation of the A*-based algorithms is the large number of nodes being generated during the search process. Sometimes, generating nodes can take significantly more time than expanding nodes that contribute to finding a solution. The Enhanced Partial Expansion A* (EPEA*) [34] uses the Operator Selection Function (OSF) to reduce the number of nodes generated by A*. Goldenberg et al. proposed two new variants of EPEA* by applying additional strategies to determine which nodes are necessary to be generated or can be skipped to further boost the performance [47].

ICTS Sharon et al. [138] introduced a two-level algorithm, ICT-search (ICTS), which uses an increasing cost tree (ICT) to represent each agent's possible paths in a high-level search tree. This method significantly

outperformed traditional A* approaches by more efficiently managing the search space and minimizing collisions.

2.2.2 Conflict Based Algorithm

CBS The Conflict Based Search (CBS) [137] is a two-level search algorithm, with a high-level search over a Conflict Tree (CT) and low-level searches for paths of individual agents that are constrained by the CT nodes. The two-level search structure allows CBS to work more efficiently by reducing the number of states being examined than A*-based algorithms. The computation complexity is mainly affected by the number of CT expansions (i.e., the number of conflicts being resolved). Many variants of CBS have been proposed to further improve its performance.

ICBS The ICBS [16] enhances the efficiency of high-level search by prioritizing CT node expansions based on the concept of *cardinal conflicts*. The cardinal conflicts occur when all shortest paths for the two conflicting agents involve traversing the conflicting vertex or edge at the same timestep.

CBSH/CBSH2 CBSH [35] introduced different admissible heuristics for CBS by aggregating cardinal conflicts among agents. By using heuristics that focus on high-impact conflicts, CBSH significantly improved the performance of the original CBS algorithm. CBSH2 [90] extended CBSH by combining more sophisticated heuristics beyond cardinal conflicts. By considering potential collisions in future solutions and reasoning about the pairwise dependencies between agents, CBSH2 further improved the scalability of MAPF algorithms.

CBSH2-RTC Pairwise symmetry occurs when two agents have multiple paths to their goals, but every path combination results in conflicts. This often leads to exponential search space growth and very slow solving time. CBSH2 has many variants with different symmetry-breaking techniques such as rectangle reasoning [93], corridor reasoning [91], and mutex propagation [176]. The original CBSH2 and its variants

are commonly referred to as CBSH2-RTC [92]. CBSH2-CHBP [140] further improved CBSH2-RTC by introducing advanced techniques for calculating heuristics for more than two agents.

2.2.3 Compilation Based Algorithm

The compilation-based algorithms are the algorithms that convert the MAPF problem into a different known problem such as Answer Set Programming (ASP) [32], Constraint Satisfaction Problem (CSP) [130] and Satisfiability Problem (SAT) [156]. These algorithms are also known as reduction-based algorithms and the MAPF problem can be reduced to a different problem in polynomial time. Most early reduction-based algorithms are initially designed to optimize the makespan of MAPF and it is non-trivial to develop the variant version that optimizes for the sum-of-cost.

SAT The performance of SAT-based MAPF algorithms is largely affected by the number of variables of the converted problem. Thus, it is crucial to find a more practical way of encoding the MAPF instances. Surynek et al. develop the first SAT-based algorithm for sum-of-cost variant of MAPF [158] (will be referred as SAT for later sections). SAT algorithms use the cardinality constraints [6, 104] to bound the sum-of-cost. The known lower bounds of the sum-of-cost can help reduce the number of variables needed to encode SAT instances, making them more solvable for SAT solvers. The encoding size can be further reduced by using the multi-value decision diagrams (MDDs) [152] which was originally proposed by the ICTS algorithm [138]. SAT-based algorithms generally perform quite well on smaller maps but struggle with large maps.

SMT-CBS SMT-CBS [157] combines the search-based method with the SAT-based method by using the satisfiability modulo theories (SMT) [15] for high-level search. SMT-CBS can find collision-free paths without introducing all constraints, resulting in faster solving speed for instances with sparse agents in large environments.

LazyCBS One of the key problems affecting the performance of CBS-based algorithms is the repeated resolution of identical subproblems across different branches of the constraint tree. This leads to a significant waste of processing time. LazyCBS [41] solves this problem by using a constraint programming model based on lazy clause generation (LCG) sovlers [119]. This approach avoids repeatedly solving the infeasible subproblems. Different from the CBS algorithm where high-level search branches on conflicts between agents, LazyCBS uses core-guided search [5] and constructs a database to keep track of the constraints where the previous search failed. This database allows for more efficient solving of previously encountered subproblems.

MIP Optimal MAPF problems can be solved as optimization problems rather than converting them into SAT problems. Yu et al. [172] converted the problem into a multi-commodity flow problem and proposed a mixed integer programming (MIP) solver. This approach uses the time-expanded graph (TEG) to model the environment and represents the conflicts through gadgets [172]. The MIP-based algorithms also suffer from inefficient encoding. However, owing to the improvements in modern MIP solvers, they still perform well in small instances.

BCP Lam et al. [78] combines the strengths of search-based and compilation-based algorithms using a decomposition framework called branch-and-cut-and-price (BCP). This framework was originally developed for mathematical optimization [79, 78]. The BCP algorithm uses a divide-and-conquer strategy to decompose hard optimization problems into easier problems. These individual subproblems are then merged back using a branch-and-bound search tree [26, 97, 25]. Similar to CBS, BCP is also a two-level algorithm. The low-level of BCP calculates the planned paths for each agent individually using the A* algorithm. BCP models the high-level problem as an MIP problem and uses modern optimization technologies to boost the solving performance. BCP also includes the symmetry reasoning techniques used in CBSH2-RTC [92] such as rectangle reasoning [93] and corridor reasoning [91].

Category	Algorithms
Search Based	A*+OD+ID [153], EPEA* [34], ICTS [138]
Conflict Based	CBS [137], ICBS [16], CBSH [35], CBSH2 [90], CBSH2-RTC [92], CBSH2-CHBP [140]
Compilation Based	MIP [172], ASP [32], CSP [130], SAT [158], SMT-CBS [157], BCP [79, 78], LazyCBS [41]

Table 2.1: Summary of optimal MAPF algorithms.

The major advantage of compilation-based algorithms is that they can benefit from the advancements in existing solvers for the converted problems. These specialized solvers have been studied more extensively than MAPF and are more powerful.

2.3 Evaluating and Benchmarking MAPF Algorithms

In this section, we introduce the methodologies and datasets used to evaluate MAPF algorithms and provide a comprehensive analysis of their performance characteristics.

2.3.1 The MAPF Benchmark Set

Most existing MAPF algorithms are evaluated using the MAPF Benchmark Set [155]. It provides 33 maps in various styles such as game, city, random, or warehouse maps. Each benchmark map has a list of predefined start/goal locations, and the authors suggest gradually increasing the number of agents until the algorithm reaches its runtime or memory limits. It also provides two different distributions of agent path length: one is randomly generated and the other is an even mixture of short and long paths.

The MAPF Benchmark makes it easier to compare the maximum capability of different algorithms with its fixed list of problems. However, the fixed problem lists and relatively small collection of maps also have limitations. Algorithms often need to be tested on a broader range of maps and more diverse agent distributions to better represent real-world applications.

2.3.2 Performance Analysis of Optimal MAPF Algorithms

Algorithm Selection research has conducted most of the comprehensive benchmarking and performance studies of MAPF. Some early benchmark studies can be found in [68, 67, 127, 142]. This is because algorithm selection requires a predefined portfolio of candidate algorithms, ideally covering a wide range of algorithms with different strengths and weaknesses. There is no way to build a robust algorithm portfolio without extensive tests of a broader set of existing algorithms. For example, Ewing et al. [33] conducted the largest and most comprehensive benchmarking study of optimal MAPF algorithms to date, including 6 different algorithms and over $14,000$ instances [*]. Shen et al. [139] developed an online interactive database to share and track the performance of state-of-the-art algorithms on the MAPF Benchmark.

2.3.2.1 Strengths and Weaknesses of MAPF Algorithms

As demonstrated in Figure 1.1 of Chapter 1, MAPF algorithms exhibit distinct strengths and weaknesses. Some algorithms perform better on specific map layouts. For example, CBSH performs better on fractal maps, while LazyCBS favors warehouse maps. However, the exact nature of these strengths remains unclear, as performance can still vary within the same type of map (e.g., Table 1.1). Some algorithms perform well on smaller instances and struggle on large ones. This often happens to relatively simple algorithms, since they don't require extra computation overhead of advanced heuristics. For example, SAT-based algorithms [158] perform well on smaller instances with fewer agents but have significantly worse performance compared to BCP or LayzCBS as problem size increases [33]. When the instances are easy, advanced heuristics are not needed to improve the speed and sometimes they can even slow down the performance. However, as instances become more difficult, these simple algorithms tend to take longer to find a solution.

[*] I'm a co-author of this benchmark study. My contribution is the analysis of the correlations between single-agent shortest path and instance hardness.

Table 2.2: Performance of our portfolio algorithms and a hypothetical Oracle algorithm that selects the fastest algorithm for each individual instance. All values are for instances where at least one algorithm successfully completed. Adapted from [33].

Algorithm	Completed (%)	Fastest (%)	Total Runtime (hrs)	Marginal Contribution (hrs)
CBS	31	0	816	0.0
CBSH	68	**45.8**	391	15.24
BCP	66	8.2	428	31.14
ID-SAT	43	0.2	712	1.96
SMT-CBS	47	0.1	657	0.01
LazyCBS	**95**	**45.8**	**115**	**243.41**
Oracle	100	100	55	-

2.3.2.2 Performance Analysis and Algorithm Selection

Choosing the most appropriate MAPF algorithm for each instance is no easy task. In Table 2.2, we show more performance data of MAPF algorithms as we did in Chapter 1. *Completed* represents the percentage of instances an algorithm solved within time-limit (5 minutes). *Fastest* is the percentage of instances where an algorithm outperformed others in runtime. Both of them are measured on solved instances rather than attempted instances. *Oracle* represents the data for always using the best algorithm. *Marginal Contribution* is the difference in total runtime for Oracle without including a specific algorithm.

From Table 2.2, LazyCBS is no doubt the best-performing algorithm. It solved 95% of instances in 115 hours and ranked the fastest for 45.8% of them. CBSH also ranked the fastest for 45.8% of instances but only solved 68% of total instances using 391 hours. Although LazyCBS demonstrates strong performance, the Oracle solved all instances in just 55 hours, significantly faster than LazyCBS. This demonstrates the importance of algorithm selection when solving large sets of instances. The marginal contribution provides insight into how much improvement a new algorithm brings when added to the algorithm portfolio. It also helps MAPF researchers filter out older, less innovative algorithms with low marginal contributions, as these are less competitive and not worth comparing when benchmarking new algorithms.

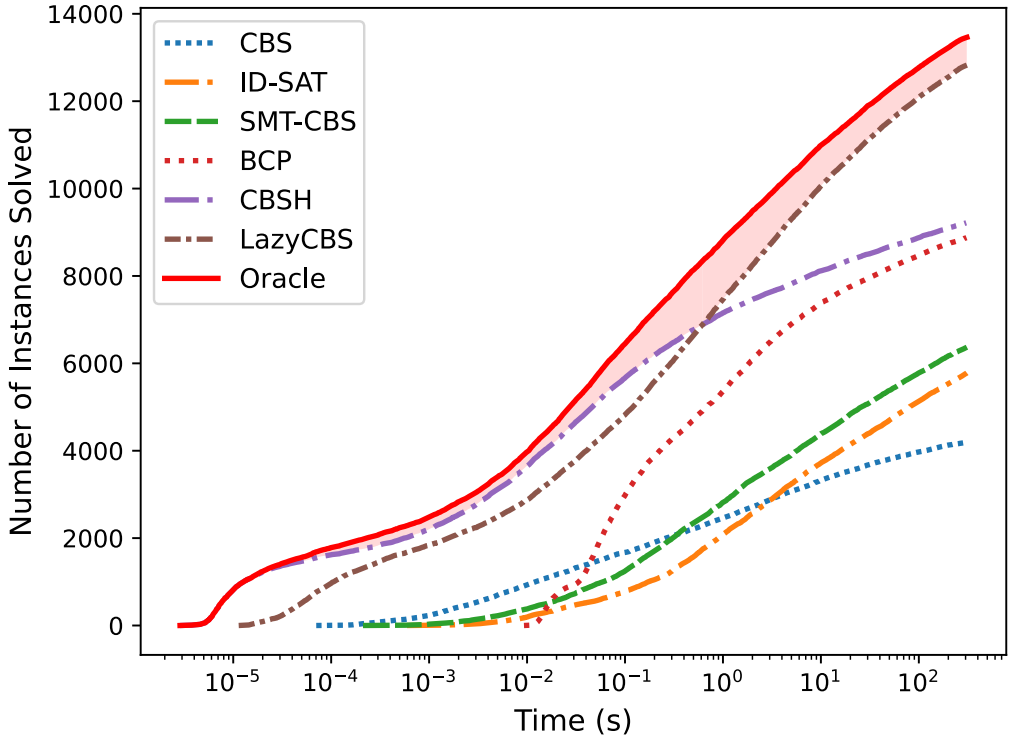


Figure 2.3: Number of instances solved for different MAPF algorithms w.r.t. different cutoff time. Oracle represents the data for always choosing the best algorithm. Adapted from [33].

2.3.2.3 Impact of Cutoff Times on Algorithm Performance

When benchmarking the MAPF algorithms, the choice of cutoff times (i.e., time limit) can also affect the relative performance difference between algorithms. Figure 2.3 shows the number of instances an algorithm solved within a certain time (the log-scaled x-axis). With a smaller cutoff time, CBSH solved more instances than LazyCBS, but as the cutoff time increased, LazyCBS outperformed CBSH. Similarly, BCP solved significantly more instances when using a larger cutoff time. This might caused by how compilation algorithms like LazyCBS and BCP are implemented. They often require additional time to convert the original instance into other forms, such as SAT encoding [157], or to construct complex optimization functions [79]. This process results in slow starts since these algorithms spend extra time on setup before solving the instances.

Chapter 3

Empirical Hardness of MAPF Problem

In this chapter, we formalize the study of empirical hardness and provide a comprehensive introduction to key research topics in this field. We discuss the motivation and importance of this problem and highlight our contributions towards existing challenges. For less studied topics, we offer preliminary ideas and insights. The chapter concludes with the first formal taxonomy of empirical hardness research in MAPF.

3.1 Introduction

Empirical hardness refers to the actual difficulty for an algorithm to solve a problem instance. More specifically, empirical hardness is often defined by the time an algorithm takes to solve a given instance. Unlike theoretical complexity research, which focuses on worst-case scenarios, empirical hardness studies the instance-specific difficulty. For many existing NP-hard problems, knowing the worst-case complexity doesn't help much in improving the algorithm performance or understanding the inherent structure of problem instances. Although the worst-case complexity of many NP-hard problems grows exponentially with problem size, we still see large instances of hard problems being solved by optimal algorithms within a reasonable time [33, 139]. This suggests that even computationally hard problems have relatively easy

This chapter will be submitted to AAMAS-25.

instances, and the real-world instances are not always the hardest ones. However, it is often unclear what factors contribute to the differences in instance hardness.

The study of empirical hardness aims to understand how the instance features correlate with instance hardness, and how to apply this knowledge to enhance other related research areas. Most early studies of empirical hardness focused on satisfiability (SAT) problems [118, 17, 2], traveling salesman problem (TSP) [43, 147], and combinatorial auctions [85, 87]. These studies explored a wide range of topics and achieved significant success. For example, in SAT research, a key discovery was the phase transition phenomenon, where problem hardness changes dramatically as instances shift from solvable to unsolvable [17]. Further research discovered that the clause-to-variable ratio strongly correlates with instance hardness [107] and can be used to generate challenging SAT instances [135]. These findings have led to the development of numerous new heuristics and algorithms. As the number of existing SAT algorithms grew, choosing the most suitable algorithm for different instances became a new challenge. This led to algorithm selection research where machine learning techniques were used to predict the empirical hardness without solving the instances [169].

3.2 Motivation and Contribution

The empirical hardness in the context of MAPF is not a well-studied problem. Most research efforts of MAPF have primarily focused on developing new MAPF algorithms or heuristics to improve algorithm performance. Early studies on the empirical hardness of MAPF have mainly been limited to benchmark studies or performance analyses of existing algorithms from algorithm selection research [127, 33, 67]. This is because algorithm selection models require a robust portfolio that captures the diverse strengths of existing algorithms. And there is no way to build such algorithm portfolios without thorough benchmark studies. These benchmark studies have revealed the complex nature of MAPF instances and offered strong examples of why studying empirical hardness is important. For instance, in Figure 1.1 from Chapter 1, we

showed that different algorithms exhibit significant runtime variations when solving the same instances. And there is no overall winner that performs well in all instances. Moreover, in Figure 1.2b from Chapter 1, we demonstrated that instance hardness can be abruptly changed by simply changing the distribution of start and goal locations, without modifying the map topology or the number of agents. These findings strongly motivate the need to further investigate the empirical hardness of MAPF.

Motivation

Understating the empirical hardness of MAPF is important for several reasons:

- It provides insights into why certain MAPF instances are more challenging than others, potentially guiding the development of more efficient algorithms and specialized heuristics.
- It enables the creation of MAPF benchmark sets that better represent a diverse spectrum of problem hardness.
- It will enhance the design of algorithm selection models and improve the performance of real-world multi-agent systems.

Contribution

In this chapter, we formalize the empirical hardness study in MAPF and present the first comprehensive taxonomy of research in this area. We provide an in-depth introduction to key research areas, identifying existing challenges and proposing preliminary solutions along with future research directions. Our study establishes a new research direction in MAPF on empirical hardness and will draw greater attention to this area for future research.

3.3 Measuring Empirical Hardness

Empirical hardness is typically measured by the computational resources required to solve an instance, such as runtime or memory usage. Since runtime depends on the hardware and can vary over different machines, other more specialized metrics are also used. For example, the number of Davis–Putnam (DP) [23] calls for SAT algorithms [107] or the number of node expansions for the tree-based algorithms [126]. In this dissertation, we use *runtime* as the default metric. This is because our study covers a broad range of MAPF algorithms implemented using various techniques. This makes it impractical to find a single specialized metric that works for all algorithms.

3.4 Major Research Areas in Empirical Hardness

Since empirical hardness is not a well-explored topic in MAPF, we will first review how it has been studied in other research domains. For areas where MAPF empirical hardness has been addressed, we will introduce the relevant research. For unexplored areas, we will address key challenges, propose preliminary approaches, and suggest directions for future work.

3.4.1 Algorithm Selection

Algorithm selection aims to select the best algorithm for a given instance from a pre-defined algorithm portfolio while minimizing certain performance objectives (e.g., runtime) [128]. Given a set of instances and a set of algorithms, algorithm selection maps the algorithm performance with the instance features. This problem stems from the fact that different algorithms have distinct strengths and weaknesses, thus causing significant performance gaps across different instances. Selecting the most suitable algorithm on a case-by-case basis can significantly reduce total runtime and save computational resources (e.g., Table 1.1).

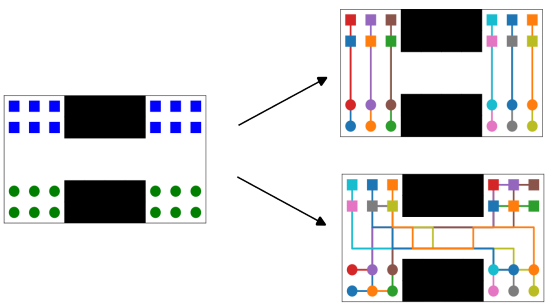
3.4.1.1 Algorithm Selection in Non-MAPF Domains

Algorithm selection has achieved significant success in solving many computationally hard problems such as boolean satisfiability (SAT) [118, 169], combinatorial auction [87], and Mixed Integer Programming (MIP) [168]. Most modern research treats algorithm selection as a classification problem and uses machine learning techniques [76]. This approach, known as Empirical Hardness Model (EHM) [87, 62], learns a mapping from instance features to algorithm performance and can be used to predict how well an algorithm will perform on given instances. For example, SATzilla [117, 169] trains a linear regression model to predict the runtime of SAT instances. Beyond runtime prediction, EHMs are also useful for obtaining insights into key factors that correlate with algorithm performance and identifying the distribution of hard instances [85, 86].

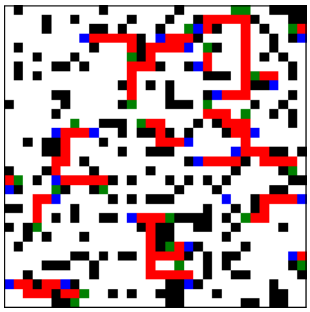
3.4.1.2 Algorithm Selection in MAPF

Algorithm selection has also been successfully applied to MAPF problems. Sigurdson et al. [142] treated the algorithm selection problem as an image classification problem by encoding MAPF instances as RGB images and developed a convolutional neural network (CNN) model based on AlexNet [77]. Kaduri et al. [67] proposed a different approach, using hand-crafted features (e.g., obstacle density, agents' average distance to their goals) and the decision tree model XGBoost [18]. We developed MAPFAST and improved the performance of the CNN-based model by using inception modules [160] and single-agent shortest path encoding [127]. Alkazzi et al. [4] further improved the performance and training speed of MAPFAST by using the variational autoencoder (VAE) [72]. A recent study proposed a hybrid model by combining the hand-crafted features from the XGBoost model with graph-based encoding from MAPFAST [136].

Challenges and Advancements in Feature Selection Algorithm selection requires a good understanding of key instance features that are important to problem hardness. These features are highly



(a) The encoding by [142] fails to distinguish between different agent location permutations.



(b) Improved encoding using single-agent shortest paths (marked in red).

Figure 3.1: Different encoding methods for MAPF instances. Start and goal locations are marked in green and blue, and obstacles are in black.

problem-dependent, thus making it challenging to find the most relevant ones. For simpler problems such as SAT, these features are easy to decide (e.g., clause-to-variable ratio). However, identifying the key instance features is particularly challenging for complex problems. For MAPF problems, features that are commonly used to analyze theoretical complexity such as the number of agents or obstacle density are not effective for individual instances. Both the map topology and the agent distribution influence the hardness of MAPF instances (e.g., instances shown in Figure 1.2). Kaduri et al. [67] proposed several hand-crafted features such as agent sparsity and average distance to goals. However, these features do not reflect enough information about the map topology and there is no guarantee all important features are accounted for.

Challenges and Advancements in Instance Encoding Another key challenge is how to effectively encode MAPF instances in a way that captures the relevant features influencing problem hardness. For early CNN-based algorithm selectors, MAPF instances are encoded as RNG images, with start and goal locations annotated as pixels in different colors [142]. In Figure 3.1a, we show that such encoding fails to distinguish instances with different permutations of start and goal locations, even though these instances have significantly different hardness. To address this, we introduced an additional encoding called single-agent shortest path and significantly improved the performance [127] (shown in Figure 3.1b). The single-agent short path is the shortest path for each agent without considering the conflicts with other agents. It

captures information on both agent distribution and map topology. Single-agent short path encoding and its variants are widely used in most of the CNN-based MAPF algorithm selector [127, 4, 19, 136]. We will further address this in Chapter 4, Section 4.4.

Future Directions Future MAPF algorithm selection research should prioritize improving instance encoding techniques. Most previous improvements in CNN-based models are from the use of more advanced deep learning models [127, 4]. However, Chen et al. [19] showed that the performance differences between different deep learning models are minimal when using the same encoding method and there is no overall winner. Therefore, future research should focus on improving encoding methods to better capture the inherent hardness of MAPF instances rather than solely refining deep learning architectures.

3.4.2 Algorithm Configuration

When an algorithm has numerous heuristics and parameters, it is crucial to adjust the configuration based on the specific instance to achieve better results. Algorithm configuration seeks to find the best parameter setting for a set of instances. More formally, given an algorithm and a set of problem instances, algorithm configuration finds the parameter setting that minimizes an objective metric, such as runtime to solve a problem instance or solution quality achieved within a time limit [59]. These parameters can be of various types, such as categorical, boolean, or continuous, as long as they affect the algorithm's behavior.

3.4.2.1 Algorithm Configuration in Non-MAPF Domains

Algorithm configuration has been successfully applied to SAT [57, 61, 60] and MIP [58, 59]. Early research approaches algorithm configuration as an optimization problem and searches for configurations with good performance [70]. For instance, ParamILS [60] uses iterated local search (ILS) to find the best configuration for a given set of benchmark instances. SMAC [59] constructs a random forest model to search for promising configurations and iteratively improve the performance based on the actual runtime.

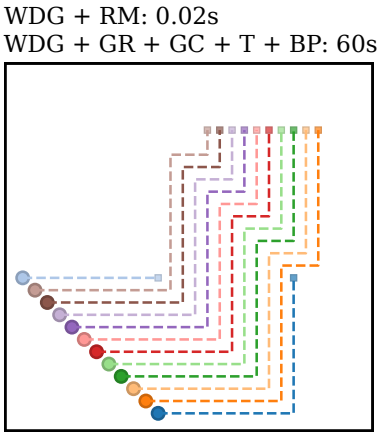


Figure 3.2: Runtime of CBSH2-RTC algorithm for a MAPF instance using different heuristic configurations. Circles and squares represent the start and goal locations for different agents. Planned paths are marked in dashed lines.

3.4.2.2 Algorithm Configuration in MAPF

Algorithm configuration has not yet been applied to MAPF problems. Most existing optimal MAPF algorithms have fewer heuristics than other combinatorial problems, so using algorithm configuration might be unnecessary. Additionally, the performance differences of these heuristics haven't been fully explored.

Challenges and Future Directions One possible future direction is to decide the configuration of different heuristics for MAPF solvers such as CBSH2-RTC [92]. For instance, Figure 3.2 shows a corner case where two different configurations of the CBSH2-RTC algorithm have significantly different runtimes. These configurations indicate which high-level search heuristics and symmetry-breaking techniques are applied. Detailed information about the search heuristics can be found online [*]. However, it remains unclear why certain combinations of heuristics can sometimes negatively impact runtime. The algorithm configuration model will help identify the best heuristic configurations for different instances. Furthermore, instances where all current configurations perform poorly can be used to inspire the development of new heuristics.

[*]https://github.com/Jiaoyang-Li/CBSH2-RTC

3.4.3 Phase Transition

Phase transition refers to the abrupt change of system behavior or properties when some critical parameters are changed [149]. This phenomenon is widely studied in physics and often describes the transition of different states for substances (e.g., water changes from liquid to gas). In a broader concept, phase transition also exists in many computationally hard problems, such as NP-complete problems. Similar to how physical systems demonstrate sudden changes in their properties at critical points, NP-complete problems can exhibit sharp transitions in their difficulty or solvability at certain thresholds [17]. There is no secret that the complexity of NP-complete problems grows exponentially with problem size in worst-case scenarios. However, existing algorithms are still able to solve many instances of NP-complete problems within a reasonable amount of time, suggesting that hard problems still have many easy instances. It is of great interest to understand when the instances of NP-complete problems shift from easy to hard.

3.4.3.1 Phase Transition in Non-MAPF Domains

Phase transition is a fundamental problem for empirical hardness research. The most successful work in this field has been done on the satisfiability (SAT) problem. Researchers found that the hardness of SAT instances exhibits a *easy-hard-easy* pattern, with the hardest instances occurring at the transition between satisfiable and unsatisfiable states [107, 17, 110, 2]. Selman et al. [135] showed a strong correlation between the hardness of k-SAT problem and the clause-to-variable ratio. Instances with too few clauses are easily satisfiable, while instances with too many clauses are over-constrained and can be quickly identified as unsatisfiable. Only when the clause-to-variable ratio is at a critical value, the instances become very challenging. For example, the phase transition for 3-SAT occurs at a clause-to-variable ratio of approximately 4.25 [107]. Phase transitions have been proven to exist in many other NP-complete problems including graph coloring [1, 175], Hamiltonian circuit [17], constraint satisfaction (CSP) [124, 146] and traveling salesman problem (TSP) [43, 42]. There are other types of phase transition beyond the satisfiability or

solvability of the problem. For instance, Achlioptas et al. [2] showed a different type of phase transition for the SAT instances without solvable/unsolvable phase by controlling the percentage of backbone. This will be further discussed in Section 3.4.5.

Phase Transition and Hard Instance Generation Phase transitions are highly valuable in the study of empirical hardness, as they can be used to systematically generate hard problem instances. For example, we can generate hard SAT instances by controlling the clause-to-variable near phase transition point [135]. This is particularly useful for creating challenging benchmark sets that test the limits and robustness of algorithm performance.

Challenges in Studying Phase Transitions for Non-Decision Problems It is difficult to study phase transitions for non-decision problems. For optimization problems, the goal is to find the best solution while satisfying the given constraints. This process normally won't incur a phase transition in solvability since a feasible solution can always be found by relaxing the constraints. Early research handles this problem in two different ways:

- **Convert to Decision Problem:** One approach is to study the decision version of the optimization problem by asking the question: "Does a solution exist for cost $\geq k$?". This approach has been successfully applied to TSP problems [43] but suffers from high-dimensional parameter space.
- **Analyzing Average-Case Hardness:** The other approach is to analyze the average-case complexity. This requires making strong assumptions about the distribution of problem space or branching factors [177]. For example, assume the instances are generated through uniform random sampling.

Leyton-Brown et al. [87] combined both approaches and proposed a machine learning model to predict the empirical hardness of combinatorial auctions.

3.4.3.2 Phase Transition in MAPF

Phase transition has not been officially explored in MAPF. The inherent complexity of MAPF instances makes it difficult to find a simple parameter to characterize the hardness like SAT problems. However, several existing studies have shown abrupt changes in the hardness of specific instances. These findings provide preliminary insights into potential phase transitions in MAPF.

Abrupt Changes in MAPF Instance Hardness Ewing et al. [33] found that maps with larger average betweenness centrality tend to have harder instances. Similarly, our study [126] showed that the average hardness of instances on poorly connected maps is significantly harder than on well-connected maps (shown in Figure 3.3). Many state-of-the-art MAPF algorithms are conflict-based, and anything that will lead to an abrupt change in the number of conflicts will make the instances significantly harder. For instance, Li et al. [92] found certain arrangements of agents' locations will cause an exponential explosion of the search space, making these instances extremely hard. Using different heuristics will help solve some hard problems but there are still corner cases and no overall winners (e.g., the corner case shown in Figure 3.2). MAPF instances can be highly complex, and there is no guarantee that all possible heuristics have been discovered. Studying phase transitions can provide deeper insights into the inherent structure and complexity of MAPF instances. These understandings can lead to the discovery of new heuristics and boost the development of new algorithms.

Preliminary Approaches and Future Directions There are two possible directions to study the MAPF phase transition problem:

- **Convert to SAT Problem:** One is to study the decision version of the MAPF problem. The SAT-based MAPF solvers [156, 158] that are originally designed for make-span minimization have provided a solid foundation for this approach. By asking "Does a solution exist for make-span equal to

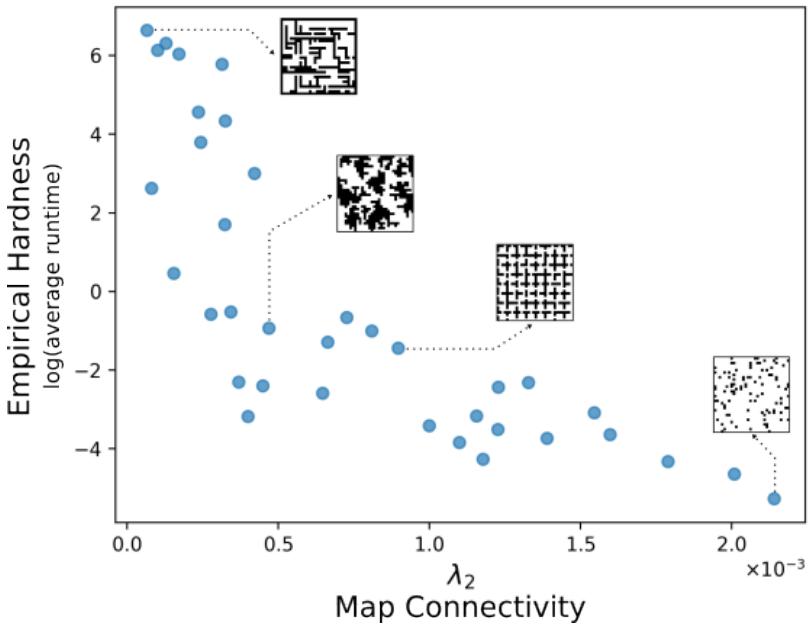


Figure 3.3: A sharp increase in empirical hardness for the maps with lower connectivity.

k?" and gradually increasing k, we can examine the hardness of generated SAT instances as they shift from unsolvable to solvable.

- **Phase Transition Beyond Solvability:** Another approach is to follow the phase transition study in combinatorial auctions [85] which focuses on the sharp change of hardness on a broader concept beyond solvable/unsolvable state. This approach often requires making general assumptions about the instance features. For example, assume the start and goal locations of agents are generated uniformly at random. By analyzing large sets of MAPF instances under varying parameters (such as agent density or map connectivity), we can identify patterns where small changes in these parameters lead to abrupt changes in empirical hardness. This approach allows for a deeper understanding of the inherent features that influence hardness and helps to reveal thresholds or critical points where MAPF instances transition from easy to hard.

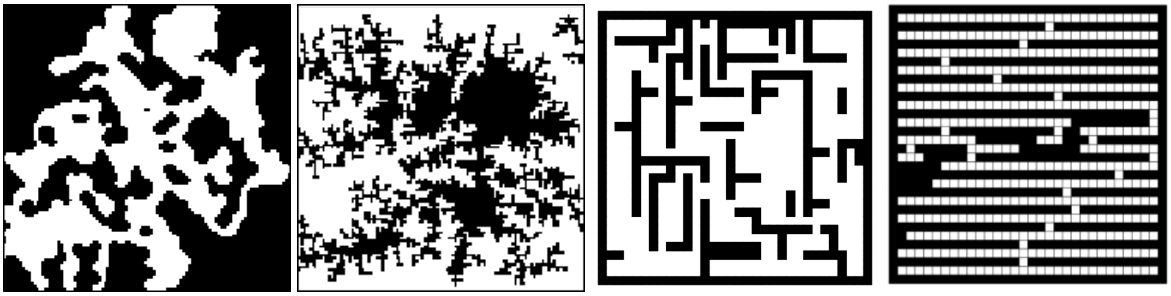
3.4.4 Hard Instance Generation

Knowing how to generate hard instances is important for gaining a deeper understanding of problem hardness. With the development of increasingly powerful algorithms, there is also a growing need for challenging benchmark datasets to thoroughly evaluate their performance. For example, the basic version of the CBS algorithm [137] struggles with some small instances, whereas BCP [79] can solve them instantly [127]. The hard instances will also inspire the development of new heuristics. For instance, Li et al. [91] developed corridor reasoning heuristics by identifying MAPF instances with corridor symmetry conflicts which are considered very challenging for the original CBS algorithm.

3.4.4.1 Hard Instance Generation in Non-MAPF Domains

Generating hard instances is not an easy task, especially without a profound understanding of problem hardness. For example, randomly generated SAT instances are generally easy to solve, but hard instances occur more frequently at specific clause-to-variable ratios [135]. The study of phase transition has inspired numerous methods of generating hard instances, as the most challenging instances often exist in phase transition regions. Selman et al. [135] demonstrated that challenging SAT instances can be generated by setting the clause-to-variable ratio to approximately 4.25. Achlioptas et al. [2] showed that the hardness of satisfiable SAT instances can be finely controlled by the percentage of backbone [†]. Leyton-Brown et al. [84, 86] used the Empirical Hardness Model (EHM) to guide the search for appropriate parameter settings of instance generators to generate hard instances for a predefined set of algorithms.

[†]The concept of backbone will be introduced in Section 3.4.5



(a) Cellular Automata [33] (b) Fractal [33] (c) Maze Diffusion [126] (d) CMA-MAE [178, 125]

Figure 3.4: Example maps generated by different methods.

3.4.4.2 Hard Instance Generation in MAPF

Since phase transition in MAPF has not been formally studied, it cannot be used to guide the generation of hard instances as it does in other research domains. Nevertheless, MAPF researchers have explored various alternative methods for creating challenging instances.

Techniques for Generating Challenging Maps Some researchers make assumptions about the distribution of agents' start and goal locations and focus on the topology of maps. The most common assumption is that agents' start and goal locations are sampled uniformly at random. This scenario studies the average empirical hardness of the MAPF problem on different maps. For instance, Ewing et al. [33] showed that the empirical hardness is correlated with the betweenness centrality of the maps. Betweenness centrality measures how often a map cell is traversed by the shortest paths and a high betweenness centrality often indicates choker points. It also introduced two novel ways of generating maps with different styles: one based on Cellular Automata (CA) [66] and the other on Diffusion-Limited Aggregation (DLA) method [165]. Building on this, we adapted the DLA method to generate maze-like environments [126]. Similarly, our study showed that hard instances occur more frequently on poorly connected maps. We also demonstrated how to generate maps with controllable connectivity using the Quality Diversity method [126]. More details will be discussed in Chapter 5.

Quality Diversity Methods in Map Generation Other research focused on making instances easier also provides valuable insights into the generation of hard instances. Zhang et al. [179] used the Quality Diversity (QD) method to search for the best allocation of shelves and endpoints in the warehouse environment to optimize the throughput. This process can be interpreted as generating maps with easier average empirical hardness. It used Mixed Integer Programming (MIP) to repair and enforce specific requirements of the warehouse environment (e.g., map connectivity and shelf placements). This method was later improved by using Neural Cellular Automata (NCA) [111] and Covariance Matrix Adaptation MAP-Annealing (CMA-MAE) [38, 37] to generate environments with arbitrary scalability [178]. A recent study [125] built on top of [178] presented a novel framework to generate diverse benchmark MAPF datasets with varying levels of difficulty for sub-optimal MAPF algorithms. The study also identified the number of obstacles and the Kullback–Leibler (KL) divergence of the tile pattern distribution as the two most effective metrics for evaluating general hardness and spatial arrangement. Figure 3.4 shows the typical map layouts for the different generators mentioned above.

Preliminary Approaches and Future Directions It is still very challenging to generate hard MAPF instances without making general assumptions about the distribution of the agents. Several potential approaches could address this:

- **Topology Analysis :** Analyze the topology of the maps and then strategically place the agents in poorly connected regions, such as narrow corridors, to increase potential conflicts.
- **Reinforcement Learning :** Train an instance generator using reinforcement learning, with a reward function designed to correlate with the number of conflicts introduced by newly placed agents. This effectively guides the model to generate more challenging instances.

- **Specialized Hard Instances:** This approach focuses on extracting MAPF instances from other problems. For example, one could extract challenging SAT instances from a MAPF problem by converting it to an SAT problem [157]. However, the primary challenge is identifying a more complex problem that can be effectively reduced to MAPF.

3.4.5 Backbone and Backdoor

Backbone and backdoor are important concepts in understanding empirical hardness, especially in SAT and other optimization problems. These concepts provide insights into the structure of problem instances and can guide the development of algorithms.

3.4.5.1 Backbone and Backdoor in Non-MAPF Domains

Backbone The backbone represents a set of variables whose values are identical across all possible solutions of an instance. It has been formally studied in the context of SAT problems [110]. The backbone is often used as a key indicator for empirical hardness. When the number of backbone variables is large, the SAT problem is over-constrained and it is hard to find feasible solutions [2, 122]. Early identification and elimination of the backbones will help improve the algorithm's performance [110, 31]. The concept of backbone has later been extended to many optimization problems such as graph coloring [145] and TSP problem [74, 133]

Backdoor The backdoor represents a set of variables that, when properly assigned, simplifies the problem [164]. The size of the backdoor is an effective indicator for estimating empirical hardness [71, 129]. For example, setting the backdoor variables properly for SAT instances will make the remaining formula solvable in polynomial time by certain SAT solvers [163]. The backdoor can be used to understand the structure of different instances, and well-structured instances typically have a smaller backdoor size. Interian [63] showed the minimum backdoor size is roughly 30% of the total number of variables for the

Random 3-SAT problem. This is much larger than more structured SAT instances and helps explain the poor performance of many SAT solvers on randomly generated instances. Apart from SAT problems, the backdoor approach has also been applied to MIP [27, 36].

3.4.5.2 Backbone and Backdoor in MAPF

There is no existing study on the backbone and backdoor of the MAPF problem. In a broader concept, the backbone could be caused by specific structures that dominate the map connectivity, such as narrow corridors or choker points. These structures might force agents to traverse certain locations in all possible solutions, causing more conflicts between agents, and making the instances harder. The backdoor could be a subset of agents that are responsible for causing the majority of conflicts. Planning their paths first will simplify the remaining problem.

Future Directions Future research should prioritize formalizing and defining the concepts of backbone and backdoors within the MAPF context. The concept of the backbone/backdoor aligns with the idea of independence detection [153], which divides agents into sub-groups. Similarly, backdoors can be viewed as specific sub-groups of agents that contribute to the hardest sub-problems. It would also be interesting to explore the correlation between the percentage of backbone and backdoors in MAPF instances and phase transitions.

3.4.6 Algorithm Competition

Algorithm competition is another valuable source for discovering new heuristics, new algorithms, and hard benchmark datasets. For example, the SAT Competition [144] [‡] originally started in 2002 and aimed to encourage the development of new solvers for the propositional satisfiability problem. As SAT solvers have become more powerful, the SAT competition has also begun encouraging the submission of challenging or

[‡]https://satcompetition.github.io/

specialized benchmark datasets to further push the limits of solver performance. Over the past 20 years, it has driven the development of numerous new heuristics and novel solvers [8, 49, 40]. It has also inspired several new research areas such as algorithm selection [167] and hard instance generation [86].

3.4.6.1 MAPF Algorithm Competition

Flatland Challenge The Flatland Challenge [108] [§] focuses on managing large-scale train networks and was launched in the competition track of NeurIPS 2019. The major objective is to calculate collision-free paths for trains to their destinations and avoid congestion. Although the Flatland challenge was originally designed for Reinforcement Learning algorithms, it also accepted solvers from MAPF research. In fact, MAPF algorithms, such as Large Neighborhood Search (LNS) [88], significantly outperformed all reinforcement learning-based approaches in the Flatland 2020 competition [89].

League of Robot Runners The League of Robot Runners (LoRR) [81] [¶], launched in 2023, is the first specialized MAPF competition and covers various topics including lifelong planning, task allocation, real-time execution, and robot dynamics. This competition focuses on real-world environments like Amazon fulfillment centers and aims to bridge the gap between research and industry by encouraging practical solutions. It also provides an online progress database for different MAPF algorithms on existing benchmark datasets [‖]. The first competition received over 800 submissions from 25 teams worldwide. It also inspired new research ideas on scaling life-long MAPF problems to more realistic settings [65, 83]

3.5 Taxonomy of MAPF Empirical Hardness Research

After exploring various MAPF research topics, we now categorize different aspects of empirical hardness in MAPF. Table 3.1 presents a taxonomy of empirical hardness research across various domains. For the

[§] https://www.aicrowd.com/challenges/flatland
[¶] https://www.leagueofrobotrunners.org/
[‖] https://tracker.pathfinding.ai/

Topic	Reference	Key Findings/Contribution
MAPF Domain		
Algorithm Selection	Sigurdson et al. [142]	First MAPF algorithm selector, CNN-based model
	Kaduri et al. [67]	XGBoost-based model with hand-crafted features
	Ren et al. [127]§	MAPFAST: CNN-based model with single-agent shortest path encoding and custom scoring
	Alkazzi et al. [4]	MAPFASTer: improved MAPFAST with enhanced training speed and stability
	Chen et al. [19]	Different algorithm selection models show minimal performance gaps when using the same encoding methods
	Shabalin et al. [136]	Graph-based model with hand-crafted features and shortest path embedding
Instance Generation	**Ewing et al. [33]**§	Cellular automata and diffusion-limited aggregation map generators
	Ren et al. [126]§	Quality diversity generator with tunable map connectivity
	Zhang et al. [179, 178]	Neural cellular automata map generator with throughput maximization
	Qian et al. [125]	Neural cellular automata map generator with KL-divergence as sparsity measurement
Hardness Transition*	**Ewing et al. [33]**§	Abrupt increase in hardness for maps with high average betweenness centrality
	Ren et al. [126]§	Abrupt increase in hardness for maps with poor connectivity
Algorithm Competition	Flatland Challenge [108]	Competition for managing large-scale train networks
	League of Robot Runners [81]	First specialized MAPF competition for online and lifelong planning with turn actions
Non-MAPF Domain		
Algorithm Configuration	Hutter et al. [60]	ParamILS: algorithm configuration for SAT/MIP
	Hutter et al. [59]	SMAC: general algorithm configuration framework
Backbone and Backdoors	Monasson et al. [110]	Backbone of SAT
	Williams et al. [164]	Backdoor of SAT/CSP
Phase Transition	Cheeseman et al. [17]	Phase transition for SAT
	Gent et al. [43]	Phase transition for TSP
	Achlioptas et al. [2]	New type of phase transition based on the percentage of backbones in SAT
	Selman et al. [135, 107]	Refined k-SAT phase transition based on clause-to-variable ratio

§ Publications where I am a first author or co-author.

* Hardness transition serves as a preliminary study of phase transition. It identifies abrupt changes in empirical hardness w.r.t. certain instance features.

Table 3.1: Taxonomy of existing empirical hardness research in MAPF.

topics that have not been studied in MAPF, we present the research from other domains. This taxonomy highlights key areas such as algorithm selection, phase transitions, and hard instance generation. Each area represents a distinct approach to understanding or measuring MAPF empirical hardness.

3.6 Summary

In this chapter, we formalized the study of empirical hardness in MAPF and explained the importance of studying this problem. We explored major research areas such as algorithm selection, hard instance generation, and phase transitions while highlighting current challenges and future directions. We also presented the first formal taxonomy of existing empirical hardness research in MAPF.

We will further expand our contributions to algorithm selection in Chapter 4 and explore the correlation between map connectivity and the empirical hardness of MAPF in Chapter 5.

Chapter 4

Algorithm Selection for MAPF Problem

Different optimal MAPF algorithms have various strengths and weaknesses and there is no universal best algorithm (at least for the current state of MAPF research). One approach to tackle this problem is to use different algorithms on a case-by-case basis and this area of study is known as algorithm selection. In this chapter, we introduce our state-of-the-art MAPF algorithm selection model, MAPFAST. We validate our hypothesis that the empirical hardness of MAPF instances can be effectively predicted using algorithm selection techniques.

4.1 Introduction

Algorithm selection is the problem of selecting the best algorithm from a portfolio to solve a given problem instance on a case-by-case basis [128]. The typical criteria for identifying the "best" algorithm usually include faster runtime, less memory usage, or other key performance improvements. In this dissertation, we focus on selecting the best algorithms based on their *runtime*. The study of algorithm selection stems from the fact that different algorithms usually have different strengths and weaknesses and there is no overall winner. Moreover, many newly developed algorithms only manage to outperform the current state-of-the-art algorithm in specific instances. This is mainly because these algorithms are often designed using

This chapter follows closely to [127].

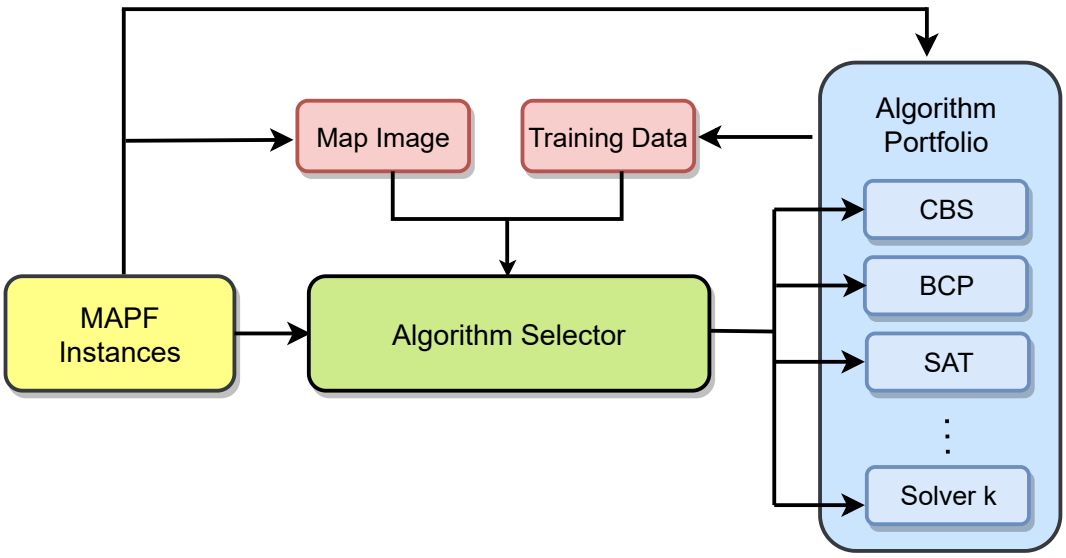


Figure 4.1: Example structure of a MAPF algorithm selector.

specialized heuristics to handle certain types of instances. In Chapter 1, Table 1.1, we briefly demonstrated that using the best algorithm will solve more instances and significantly reduce the total runtime. We will further elaborate on this in Section 4.3.

Algorithm selection can be formulated as a prediction problem, where the goal is to predict the best algorithm (fastest algorithm) from a portfolio for an input instance [148, 76]. Such techniques have been successfully applied to many computational problems, including propositional satisfiability (SAT) [169, 167] and the traveling salesman problem (TSP) [69]. Most of the modern algorithm selection models are based on machine learning techniques [76]. The model will first be trained based on the performance data (runtime, node expansion, etc.) of different algorithms and then predict the best-performing algorithm given different input instances.

Figure 4.1 shows an example structure of a MAPF algorithm selector. An algorithm portfolio is a collection of different state-of-the-art algorithms. Ideally, a good algorithm portfolio should include a diverse set of algorithms with different strengths to cover different scenarios. The algorithm selector will be trained using the performance data of the algorithm portfolio. Given a specific MAPF instance, the algorithm selector will predict the best portfolio algorithm based on the instance features.

Contribution

This chapter makes several key contributions to the MAPF algorithm selection research. First, we introduce MAPFAST, a state-of-the-art MAPF algorithm selection model. Second, we present a novel instance encoding technique, single-agent shortest encoding, which significantly improves the prediction performance. Additionally, we present a costume-scoring metric to help further analyze the performance of algorithm selectors.

4.2 Related Work

Although algorithm selection has been applied to many optimization problems, MAPF algorithm selectors have not been well studied in the literature. Sigurdson et al. [142] first proposed a classification model based on a convolutional neural network (CNN)[82] by representing the MAPF instance as an RGB image. The start and goal locations are marked as pixels in different color anonymously, in other words, there is no information about which start is associated with which goal. Their model is a version of AlexNet [77], which is modified and retrained from image classification to try and predict the fastest solver given an image input. Their model demonstrated that it was possible to predict the fastest algorithms for MAPF instances, although they only achieved relatively limited performance.

Kaduri et al. [67] proposed two different models: one based on CNNs using VGGNet [160], and the other based on a tree-based learning algorithm named XGBoost [18]. The use of the VGGNet improves the image recognition quality from many aspects such as allowing for a much deeper network and thus further boosting the prediction performance of the algorithm selector. Their work uses a MAPF algorithm portfolio that includes only optimal search-based algorithms. Based on our performance analysis in the later section, the best search-based algorithm in our algorithm portfolio, namely CBSH, is the fastest algorithm for only 30% of test cases. Thus, omitting non-search-based algorithms handicaps the performance of an

algorithm selector. Their best model, *XGBoost Cl*, requires hand-crafted MAPF features (e.g., number of agents, obstacle density) as input to their algorithm selector. Although the authors provide analysis of the impact of their hand-crafted features on the performance of their model, the performance for their algorithm selector may still be impaired by their small feature set that may not include some important features of an instance.

4.3 Algorithm Portfolio

The algorithm portfolio is a set of pre-selected candidate algorithms. When run, an algorithm selector will select a single algorithm from the portfolio to run on an instance and report the results of that algorithm. Given that MAPF instances are complex and diverse, a good algorithm portfolio must be diverse. Ideally, an algorithm in the portfolio should have strengths that cover for the weaknesses of the other algorithms. Many optimal MAPF algorithms are built on top of similar approaches with different heuristics (e.g., CBS and its variants). We seek to include optimal MAPF algorithms that are inherently different from each other to find the best algorithms for a variety of MAPF instances.

We select the following four algorithms for our portfolio:

- **Conflict Based**: Conflict-Based-Search (CBS) [137] and its state-of-the-art variant with improved heuristics, CBSH [35].
- **Compilation Based**: A reduction of the MAPF problems to propositional satisfiability problem (SAT) [158].
- **Optimization Based**: Branch-and-Cut-and-Price (BCP) [79], a method based on the decomposition framework for mathematical optimization.

Table 4.1: Performance analysis for portfolio algorithms on the entire dataset.

Algorithm	Accuracy	Coverage	Runtime	Mean	Median	StdDev
CBS	0.1908	0.40	77,091	3.088	5.000	2.344
CBSH	0.2953	**0.91**	**21,380**	0.856	0.133	1.530
BCP	**0.5129**	0.90	22,265	0.892	0.050	1.631
SAT	0.0010	0.38	85,024	3.405	5.000	2.163
Oracle	1.0	1.0	8,867	0.355	0.033	0.771

A more detailed introduction to these algorithms is provided in Chapter 2, Section 2.2. We also tested other algorithms such as EPEA* [46] and ICTS [138], but removed them due to their limited performance compared to the algorithms in our portfolio.

4.3.1 Performance of the Portfolio Algorithms

Next, we show the capabilities and different characteristics of the algorithms in our portfolio by presenting a performance analysis for each individual algorithm. We use three metrics to evaluate the portfolio algorithms:

1. **Accuracy** is the proportion of instances in which an algorithm is the fastest in the portfolio.
2. **Coverage** is the proportion of the instances that an algorithm successfully solves within the time limit (5 minutes).
3. **Runtime** is the overall time taken by the algorithm, in minutes, to solve all instances. A default value of 5 minutes is added to the runtime when the algorithm doesn't solve the input instance within the time limit.

All the algorithms are implemented using C++ and we have built wrappers around each algorithm to ensure consistent input formats and a fair comparison of runtime.

We use the MAPF Benchmarks [155] to analyze our portfolio algorithms. Our dataset of instances contains a wide variety of map types, including cities, video game maps, mazes, random maps and warehouses.

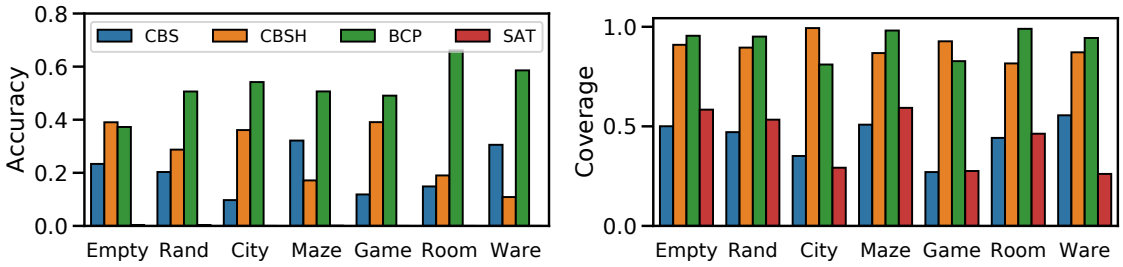


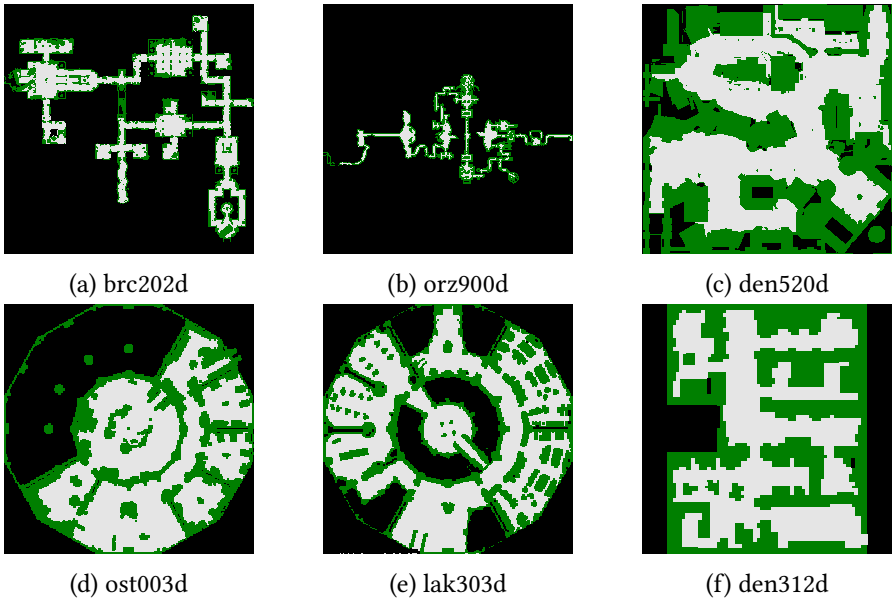
Figure 4.2: Accuracy and coverage data for portfolio algorithms for different map types. Random and Warehouse maps are labeled as Rand and Ware respectively.

When generating the instances, we only keep the instances that at least one algorithm from the portfolio can solve within the time limit (5 minutes). We generate 24967 solvable instances with varying numbers and distribution of agents. The results of the performance analysis are shown in Table 4.1. We also include the mean, median, and standard deviation of the time needed (in minutes) for different algorithms to solve the instances. BCP and CBSH are successful in solving 90% of the instances. However, BCP, the algorithm that is fastest more often than any other algorithms in our portfolio, is only the fastest algorithm for 51% of instances. Selecting only BCP to solve all the instances would take more than twice as long as selecting the best performing algorithm for each instance (shown as Oracle in Table 4.1). This further justifies the claim that there is no dominating optimal MAPF algorithm.

To answer whether a specific algorithm always performs well in certain types of maps, we present accuracy and coverage data with respect to the map types in Figure 4.2. We also present the accuracy and coverage data for a subset of game map instances in Table 4.2. Overall, BCP has the highest accuracy in the game maps. However, neither BCP nor CBSH show dominant performance over each other for the instances on these individual maps. CBSH outperforms BCP in brc202d, orz900d, den520d, ost003d but the gaps for accuracy are small. For den312d and lak303d, BCP is significantly better than CBSH in terms of accuracy. Even for the maps that have a similar topology (e.g., den321d and den520d), the fastest algorithms can still be different.

Table 4.2: Accuracy and coverage data for game maps.

Map	Accuracy				Coverage			
	CBS	CBSH	BCP	SAT	CBS	CBSH	BCP	SAT
brc202d	0.1059	**0.4677**	0.4264	0	0.1719	**0.9917**	0.7318	0.1499
orz900d	0.1073	**0.5181**	0.3746	0	0.1677	**0.9940**	0.5755	0.0997
den520d	0.1033	**0.4813**	0.4154	0	0.3091	**0.9862**	0.7608	0.2835
ost003d	0.1191	**0.4839**	0.3970	0	0.2208	**0.9739**	0.8102	0.2792
lak303d	0.1785	0.2714	**0.5501**	0	0.2459	0.8871	**0.9508**	0.3461
den312d	0.1378	0.2350	**0.6272**	0	0.3922	0.8379	**0.9951**	0.4668
game	0.1185	0.3908	**0.4906**	0	0.2704	**0.9271**	0.8275	0.2757

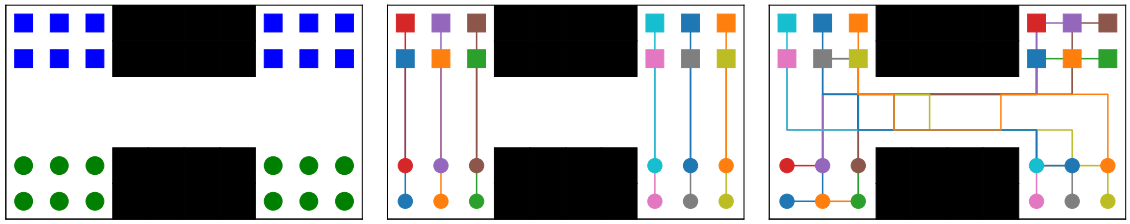


(a) brc202d (b) orz900d (c) den520d

(d) ost003d (e) lak303d (f) den312d

Figure 4.3: Maps in Table 4.2. Collected from the MAPF Benchmark [155].

Based on this data, we do not find a clear relationship between map types and algorithm performance. Some of the maps in Figure 4.3 have both narrow corridors and open spaces, making it challenging to manually choose the algorithm that works well on certain map types (e.g., use CBS for maps with narrow corridors). Owing to the fact that map topologies are usually non-homogeneous, it is necessary to analyze MAPF instances on a case-by-case basis instead of categorizing them by map types.



(a) (b) (c)

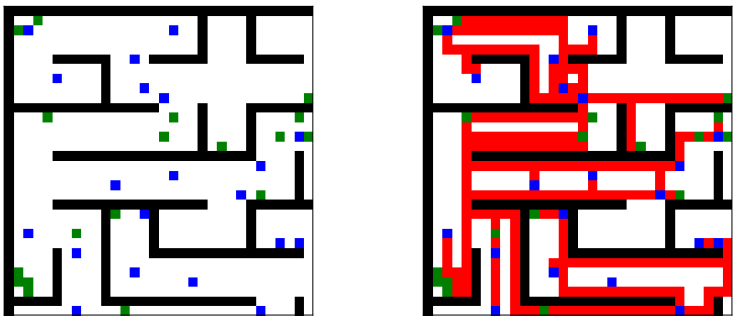
Figure 4.4: (a) MAPF instance marked only with start (green circles) and goal (blue squares) locations. (b)(c) Two different mappings of the start and goal locations with respect to the map in (a). Planned paths are marked in colored lines. For the instance shown in (b), CBS algorithm solved it in 0.02 s and BCP solved it in 0.03 s. For the instance shown in (c), CBS algorithm failed to solve it within the 5-minute time limit and BCP solved it in 0.33 s.

4.4 Instance Encoding

There are two primary methods of encoding MAPF instances for algorithm selectors: using hand-crafted features or encoding them as images.

Feature-Based Encoding Kaduri et al. [67] used hand-crafted features such as obstacle density, number of agents, etc. However, Encoding an instance using hand-crafted features cannot capture as much information as feeding the full map into a deep-learning model. This is mainly because it is hard for hand-crafted features to capture the spatial environment information such as map topology or distribution of the agents.

Image-Based Encoding Another way is to encode MAPF instances as 2D RGB images and annotate the agents' start and goal locations as pixels using different colors [142]. Figure 4.4a shows an example of such encoding. Compared to the hand-crafted feature approach, this image-based encoding can capture the topology of the map and distribution of the agents, which helps the algorithm selectors to better distinguish different instances. However, merely annotating the 2D image with the start and goal locations is not enough. This is because of a set of distribution of start and goal locations, there can still be many different combinations of which agent going to which goal and such difference will also lead to dramatic



(a) (b)

Figure 4.5: Encoding an instance map with (a) start and goal locations which are marked using blue and green and (b) single-agent shortest paths which are marked in red.

change of the hardness for the MAPF instance. For example, the map shown in Figure 4.4a and two possible permutations of agents and goals in Figure 4.4b and Figure 4.4c. Figure 4.4b represents an easy instance where all the agents in the lower part of the map need to move to the top part. It takes CBS algorithm 0.02 s to solve it and BCP takes 0.03 s. With a different mapping for start and goal locations as shown in Figure 4.4c, where two groups of agents need to swap sides, CBS doesn't finish within the five-minute time limit, but BCP successfully solves the instance in only 0.33 s. Any model trained on the binary encoding of agents and goals will not be able to differentiate between these two very different instances in Figure 4.4b and Figure 4.4c.

4.4.1 Single-Agent Shortest Path Encoding

We propose a new way of encoding MAPF instances that encodes a mapping between agents and their goals. Our key idea is that we want to provide the additional information that will help the algorithm selector to better classify different instances. More specifically, in addition to marking the start and goal locations, we include another marking in our input which encodes *single-agent shortest paths* from each agent to its goal. A single-agent shortest path is an optimal path for an agent without considering collisions with other agents. For every agent, we add only a single shortest path, despite the fact there may be many

Table 4.3: The encoding value of map cells and their corresponding descriptions.

Cell Value	Description
$[0, 0, 0]$	Cell contains an obstacle
$[1, 0, 0]$	Cell lies on a shortest path from any agent to their goal
$[0, 1, 0]$	Cell is the starting location of an agent
$[0, 0, 1]$	Cell is the goal location of an agent
$[0, 1, 1]$	Cell is both a start and goal location
$[1, 1, 1]$	Cell is empty

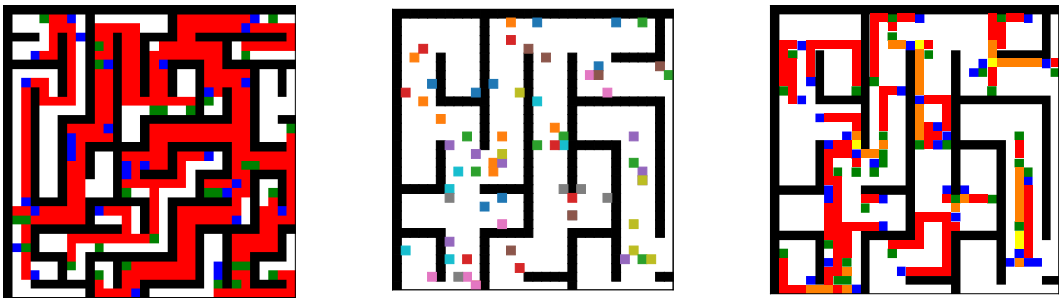
distinct shortest paths for every agent to its goal. We encode the shortest paths on our map where each cell is marked if it lies on a shortest single-agent path, an example is shown in Figure 4.5b.

There are many benefits of using the single-agent shortest path encoding. Firstly, it helps to distinguish between the MAPF instance where the agents have the same distribution of start and goal locations but distinctively different permutations of which agents go to which goal. Moreover, this encoding is very easy to calculate since it doesn't consider the collisions between different agents and thus won't lead to too much computation cost.

We initially encoded our MAPF instance into a tensor with four layers. Each layer encoded a binary feature for each map cell: obstacle, agent, goal, and shortest path. Models trained on this encoding performed relatively poorly in all metrics, perhaps requiring more training data than we generated. We then encoded our features in a different manner into a tensor with three layers by annotating every cell based on different conditions as shown in Table 4.3.

Note that since every cell with a start/goal location is guaranteed to lie on a shortest path, we only mark that these cells contain a start/goal, we do not mark that they also lie on a shortest path since it is already implied by the presence of a start/goal. This is the encoding we use for our algorithm selector models.

Limitation of Single-Agent Shortest Path Encoding As shown in Figure 4.6a, the basic version of single-agent shortest path encoding may have limited performances on maps with a large number of



(a) (b) (c)

Figure 4.6: (a) The single-agent shortest path encoding on maps with a high density of agents. (b) and (c) illustrate two alternative approaches for encoding MAPF instances: (b) using different colors for different agents, and (c). using different colors based on the frequency of a map cell being visited by the single-agent shortest path.

agents. In these scenarios, the single-agent shortest paths often overlap, covering most of the free space and making it difficult to differentiate different instances. There are some straightforward methods to tackle this issue. One improvement could be combining features like the frequency of a map cell being visited by shortest paths into the encoding.

4.4.2 Alternative Encoding Methods

There are several alternative ways of encoding the MAPF instances:

Color-Based Encoding One approach is to encode agents using distinct colors as shown in Figure 4.6b. However, our tests show this encoding performs poorly. It is challenging to determine a reasonable color scheme for different agents. As the color differences do not provide additional information, it confuses the algorithm selection model and reduces the prediction performance.

Frequency-Based Encoding Another approach is to enhance the single-agent shortest path by coloring the map cells based on how frequently they are visited by the single-agent shortest paths. This encoding didn't work well in our current model or a similar model [4]. However, we believe this approach could

outperform the basic single-agent shortest path encoding with more advanced deep learning models and a larger training dataset, as it encapsulates additional information.

4.5 Models

Similar to Sigurdson et al. [142], we treat the MAPF algorithm selection problem as an image classification problem as well. Our model is called Multi-Agent Path Finding Algorithm SelecTor (MAPFAST), where given a MAPF instance, MAPFAST attempts to automatically predict the fastest algorithm from a predefined algorithm portfolio. MAPFAST is based on the convolutional neural network (CNN) [82].

We introduce the following algorithm selectors in this section: (1) $\text{MAPFAST}_{\text{cl}}$, which treats algorithm selection as a classification task, (2) $\text{MAPFAST}_{\text{aux}}$, an augmented version of $\text{MAPFAST}_{\text{cl}}$ which adds two additional loss function, and (3) G2V, a graph-embedding based model that offers insights into what information is required to perform algorithm selection on MAPF instances.

4.5.1 $\text{MAPFAST}_{\text{cl}}$

In this work, we model algorithm selection as a classification problem. Our model takes as input an encoding of a MAPF instance and returns a prediction for the fastest algorithm in the portfolio.

4.5.1.1 Inception Module

Inception modules are used to improve training speed and allow for a much deeper network compared to architectures like VGGNet [160]. Moreover, since the inception module contains multiple sizes of convolution kernels (shown in Figure 4.7), there is no need to decide the exact kernel size for each layer as the network learns which kernel to use.

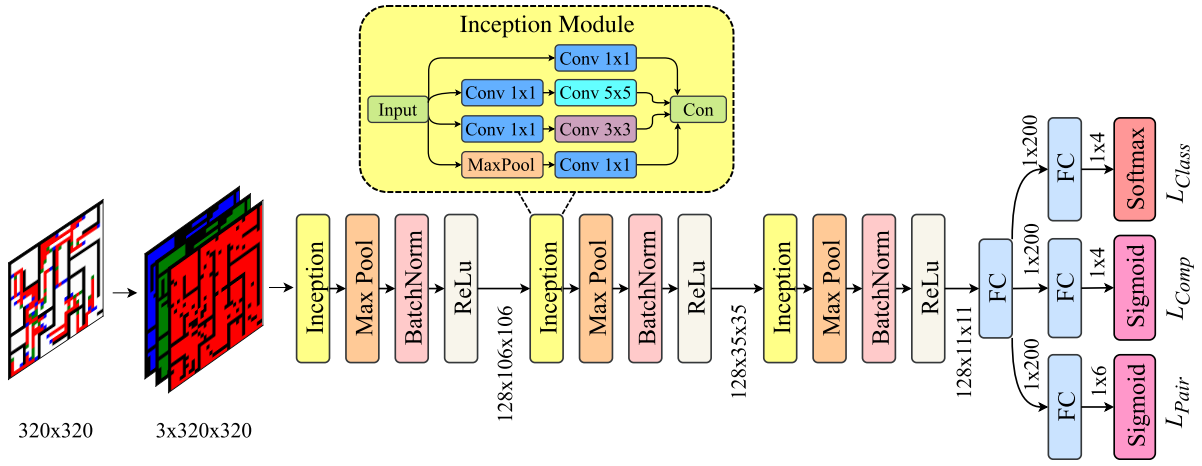


Figure 4.7: The CNN architecture of MAPFAST. $\text{MAPFAST}_{\text{cl}}$ is only using the classification loss L_{Class} and $\text{MAPFAST}_{\text{aux}}$ is using all three of the loss functions.

4.5.1.2 Model Structure

Figure 4.7 shows the model structure of MAPFAST. The input to the model is a $320 \times 320 \times 3$ tensor of the encoding described in Section 4.4.1. The input is passed through three inception modules in our CNN. Each of the inception modules is followed by a max-pooling layer (kernel size 3×3 with stride 3), a batch normalization layer, and a rectified linear unit (ReLu) activation layer. Since the pre-trained inception network [160] uses the image size of 224×224 and our map size is 320×320, we cannot use the pre-trained weights, thus we train from scratch with the Adam optimizer [73].

After the three inception modules, the network outputs a feature vector of size 15488. This is connected to a fully connected layer which outputs 200 learned features. This learned representation is then fed through a fully connected layer with a softmax activation function, which is the output of our model. The output is a prediction of which algorithm in the portfolio solves the input instance the fastest. We compute our classification loss L_{Class} using categorical cross-entropy between our predictions and the ground truth fastest algorithm. This model is referred to as $\text{CNN}_{\text{Class}}$, as it is only trained with L_{Class}.

4.5.2 MAPFAST$_{\text{aux}}$

To explore the possibility of further improving the quality of prediction, we further augment our model with two additional supplemental loss functions:

- The first additional output layer is a four-neuron layer with a sigmoid activation function to predict, for every algorithm, whether it will finish within the time limit or not. We compute our completion loss L_{Comp} using cross-entropy loss between our second output layer and the ground truth algorithm completions.
- The second additional output layer predicts the pairwise relative performance of algorithms on an input instance. This is done using six output neurons. The first three output neurons predict whether BCP will be faster than CBS, CBSH, and SAT. The following two neurons predict whether CBS will be faster than CBSH and SAT. The final neuron predicts whether CBSH will be faster than SAT. Again, we compute our pairwise loss L_{Pair} using cross-entropy.

We train a model with the total loss $L_{Tot} = L_{Class} + L_{Comp} + L_{Pair}$ and refer to this model using auxiliary output as MAPFAST$_{\text{aux}}$. Figure 4.7 shows the structure of MAPFAST.

4.5.3 G2V

For the previous models, we encode our MAPF instance as a tensor that contains information about the map as well as single-agent shortest paths. We now demonstrate that the single-agent paths alone, regardless of map topology, contain enough information to outperform any individual algorithm in our portfolio. We utilize graph embedding techniques to convert the single-agent shortest paths for an instance into an embedding and train a model to make an algorithm prediction from this embedding.

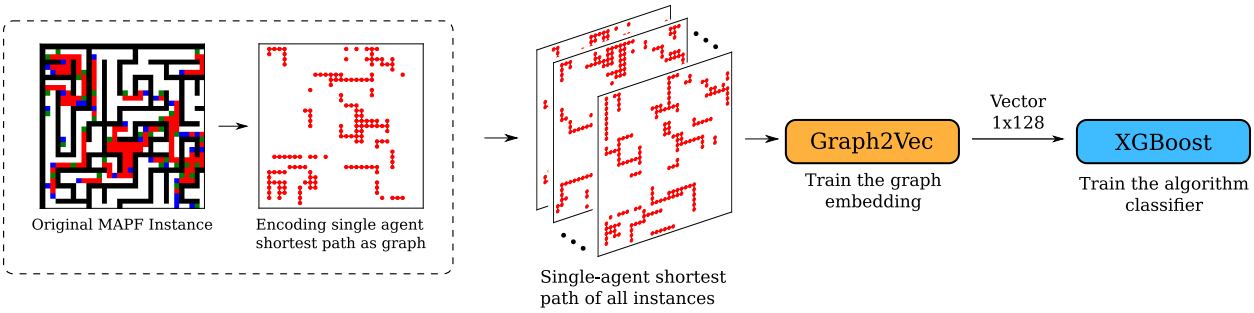


Figure 4.8: Model structure of G2V. Unlike MAPFAST, G2V only takes the single-agent shortest path as input rather than the complete map.

4.5.3.1 Encoding Method for G2V

For a MAPF instance, we construct our graph encoding as follows. First, we compute a single-agent shortest path for every agent. We then construct a graph consisting of only nodes which lie on these shortest paths. We add edges to all nodes which are adjacent in our MAPF instance. This (possibly disjoint) graph serves as the encoding of the MAPF instance. The size of our graph encoding is often significantly smaller than the size of the original instance map, as our graph encoding only contains nodes that lie along the shortest paths.

4.5.3.2 Model Structure of G2V

After encoding our MAPF instance as a graph, we use the unsupervised graph embedding algorithm *Graph2Vec* [114] to embed our graph into a vector. Graph2Vec takes as input a set of graphs and outputs a fixed-size vector representation for each graph. We feed Graph2Vec the graph representation of every instance in our dataset and it produces a vector of size 128 for each instance. While Graph2Vec requires access to every graph in our dataset (including the test set) apriori, it does not have access to information on algorithm performances while generating embeddings and can be seen as a data preprocessing step. We train an XGBoost [18] classifier on the embeddings generated from maps in our training set to predict the fastest algorithm in our portfolio. This model is referred to as G2V in our results and is using only L_{class}, the cross-entropy loss to optimize the classification accuracy.

4.5.3.3 Limitation of G2V

The drawback of this approach is that Graph2Vec needs all graphs *before* any embedding is calculated. This means that, once trained, the model cannot be used to create a new graph embedding for an unseen instance. Therefore, it is not a deployable algorithm selector in any reasonable sense. However, we believe the results from this model are very informative. Our G2V model performs well despite only having access to nodes on the shortest paths, potentially a very small fraction of the total number of nodes in the instance. In the shortest paths, there is no explicit information on the map type, obstacle density, or map size, heuristics which have previously been used to select algorithms [159, 67]. Despite this lack of map information, our G2V model performs quite well, better than any single algorithm in our portfolio, and close in performance to our CNN-based models, which have access to much more information. This suggests that just information on single-agent shortest paths may be enough to distinguish the performances of our portfolio algorithms.

4.6 Evaluation of Algorithm Selection Models

4.6.1 Simulation Setup

We evaluate the three models presented in the previous section: $\text{MAPFAST}_{\text{cl}}$, $\text{MAPFAST}_{\text{aux}}$, and G2V. Of the 24967 instances generated from the MAPF Benchmarks [155] for our evaluation, we used 80% for training, 10% for validation and 10% for testing. Before data collection, we built wrappers for the algorithms in our portfolio so that all the algorithms use a common method to read the input instances. The code can be found online [*]. We train our model for 5 epochs with a learning rate of .001, both determined by measuring performance on the validation set. All metrics reported in the following sections for algorithms and models are reported for the test set. Training takes 40 minutes using an RTX 2070 GPU.

[*] https://github.com/USC-ACTLab/MAPFAST

Table 4.4: Simulation results of algorithm portfolio and different algorithm selection models.

Algorithm	Accuracy	Coverage	Runtime
CBS	0.1888	0.41	7,714
CBSH	0.2810	0.90	**2,211**
BCP	**0.5294**	**0.91**	2,256
SAT	0.0008	0.38	8,548
CNN_{agents}	0.5865	0.86	2,577
XGBoost Cl	0.6711	0.95	1,694
G2V	0.7130	0.95	1,548
$MAPFAST_{aux}$	0.7210	0.95	1,519
$MAPFAST_{cl}$	**0.7375**	**0.95**	**1,497**
Oracle	1.0	1.0	917

We use the classification outputs of the models to select the fastest algorithm. To select an algorithm we take the argmax of the classification output, and select the corresponding algorithm (if the chosen algorithm fails to solve the instance, another algorithm is not selected).

4.6.2 Performance Metrics

As mentioned in Section 4.3, we use accuracy, coverage, and runtime to evaluate the performance of portfolio algorithms. These metrics can also be used to analyze the performance of algorithm selectors but with slightly different definitions.

The *Accuracy* metric gives the proportion of instances that the algorithm selector correctly selects the fastest algorithm. *Coverage* is the proportion of instances where the algorithm selector selects an algorithm that solves the instance within the time limit. *Runtime* is the overall time taken for the selected algorithms, in minutes, to solve all the problems in the test set. A default value of 5 minutes was added to runtime when the algorithm didn't solve the input instance within the time limit.

4.6.3 Results and Analysis

Table 4.4 shows the results from evaluating our models on the 2484 MAPF instances in the test set. In the first four rows, we report results for using a single algorithm on all input instances. Our experiments

show that BCP and CBSH were successful in solving 90% of the input instances. However, BCP, the best individual algorithm in accuracy and coverage, is the fastest for only 53% of instances and takes more than twice as long as selecting the fastest algorithm for each instance (shown as Oracle in Table 4.4).

The second part of the table shows the comparison of a state-of-the-art MAPF algorithm selector, XGBoost Cl [67], and our approach. To generate these results, we train XGBoost Cl with our algorithm portfolio and dataset. MAPFAST$_{\text{cl}}$ successfully selects the fastest algorithm for 73.75% of the input instances and had coverage of 95%, outperforming XGBoost Cl, which had 67% accuracy and 95% coverage. Our Model G2V had a performance comparable to our MAPFAST$_{\text{aux}}$ model, with 71.30% accuracy and 95% coverage, also outperforming XGBoost Cl.

4.6.3.1 Model Insights and Observations

Although using additional output layers, the performance of MAPFAST$_{\text{aux}}$ is slightly worse than the base model in accuracy, MAPFAST$_{\text{cl}}$ (72.10% v.s. 73.75%). This could be due to the need for better-tuned parameters for the loss functions of the different output layers, rather than assigning them equal weights.

The total runtime for the algorithms chosen by our models is significantly less than using the same algorithm for every instance. On average, it takes 1 second to annotate one input instance with single-agent shortest paths and 0.01 second for the trained model to select the fastest algorithm, which is negligible to the average runtime of the best portfolio algorithm (i.e., CBSH has an average runtime of 53 seconds). Our models also have a remarkable improvement in accuracy compared to all of the individual algorithms, further justifying our approach.

4.6.3.2 Coverage Analysis

We use the following method to further analyze the performance of the four output neurons in MAPFAST that use L_{Comp} loss to predict if an algorithm solves a given input instance or not. Let $\mathcal{T}$ be the set of all

Table 4.5: Actual and predicted coverage for MAPFAST$_{\text{aux}}$.

	CBS	CBSH	BCP	SAT
Actual Coverage	0.41	0.90	0.91	0.38
Predicted Coverage	0.45	0.95	0.93	0.43
Recall	0.91	0.98	0.98	0.92
Correctness	0.89	0.92	0.92	0.89

test instances. For a particular algorithm, let $\mathcal{S}$ be the set of test instances that it can solve within the time limit, and $\mathcal{Q}$ be the set of test instances it cannot solve within the time limit such that $\{\mathcal{S}, \mathcal{Q}\}$ is a partition of $\mathcal{T}$. Let $\tilde{\mathcal{S}}$ be the set of test instances our model predicts as solvable by the algorithm within the given time limit, and $\tilde{\mathcal{Q}}$ be the instances that our model predicts as not solvable by the algorithm within the given time limit. $\{\tilde{\mathcal{S}}, \tilde{\mathcal{Q}}\}$ is another partition of $\mathcal{T}$. The first row of Table 4.5 shows the actual coverage of the algorithms in the portfolio, i.e. $\frac{|\mathcal{S}|}{|\mathcal{T}|}$. The second row shows the predicted coverage of our model for each algorithm, i.e. $\frac{|\tilde{\mathcal{S}}|}{|\mathcal{T}|}$. The third row lists the recall of our model, which is the fraction of solvable instances that our model predicts as solvable: $\frac{|\mathcal{S}\cap\tilde{\mathcal{S}}|}{|\mathcal{S}|}$. The final row lists the correctness of our model, which is the fraction of correct outputs: $\frac{|(\mathcal{S}\cap\tilde{\mathcal{S}})\cup(\mathcal{Q}\cap\tilde{\mathcal{Q}})|}{|\mathcal{T}|}$. Our model predicts whether an algorithm solves an instance or not with at least 91% correctness for each algorithm. This suggests that the neural network learns the inherent behavior of algorithms for the given MAPF instances.

4.6.3.3 Custom Scoring

We also use an additional metric, the *speed award* [161], which provides more information about relative performance among different algorithms, to further analyze our models. This metric gives a greater reward for solving tasks that not every algorithm solves and a smaller reward to fast algorithms when every algorithm takes around the same amount of time. For different algorithm selectors, it gives greater rewards for the models that correctly choose the fastest algorithm when other models fail to do so. It therefore provides more information on the relative performance of algorithms and algorithm selectors than the accuracy and the cumulative runtime.

Speed Factor To calculate the speed award, we first compute the *speed factor*:

$$speedFactor(p, a_i) = \frac{300}{1 + timeUsed(p, a_i)},$$

where $timeUsed(p, a_i)$ is the time taken by the algorithm a_i to solve instance p and 300 is the time limit for each instance. The speed factor shows how fast an algorithm can solve an instance. The faster an algorithm is, the higher the speed factor will be. If algorithm a_i fails to solve the instance p, the speed factor is set to 0.

Speed Award Once we have the speed factor of all algorithms for a problem instance p, we compute the speed award for each algorithm a_i to solve instance p as follows:

$$speedAward(p, a_i) = \frac{speedFactor(p, a_i)}{\sum_{a_j \in \texttt{algorithms}} speedFactor(p, a_j)}.$$

Here, $\texttt{algorithms} = \{$BCP, CBS, CBSH, SAT, XGBoost Cl, G2V, $\text{CNN}_{\text{Agents}}$, $\text{MAPFAST}_{\text{cl}}$, $\text{MAPFAST}_{\text{aux}}$, Oracle$\}$. The speed award for an algorithm has a higher value if the algorithm solves the instance faster than other algorithms.

Custom Score The final score for an algorithm on a set of problem instances is given by:

$$score(a_i) = \sum_{\forall p} speedAward(p, a_i). \tag{4.1}$$

The custom score metric provides more information about the relative performance between different algorithm selectors than just using the runtime metric. In particular, if the algorithms selected by different algorithm selectors have very similar runtime, then similar scores will be granted to these algorithm selectors instead of giving all the credits to the fastest algorithm.

Table 4.6: Custom score results.

Algorithm	SAT	CBS	CBSH	BCP	CNN_{Agents}	XGBoost Cl	G2V	$MAPFAST_{aux}$	$MAPFAST_{cl}$	Oracle
Custom Score	28.22	79.05	232.11	274.45s	286.95	290.21	302.01	300.55	**304.20**	386.23

The calculated custom scores are shown in Table 4.6. Oracle is the model that always selects the fastest algorithm. The best model should have the highest custom score. Based on the results in Table 4.6, all of the algorithm selectors outperform the portfolio algorithms. Moreover, all of our models outperform the state-of-the-art model, XGBoost Cl. $MAPFAST_{cl}$ is ranked as the best algorithm selector by *speedAward*.

4.7 Dataset Analysis

In order to gain a deeper understanding of our MAPF algorithm portfolio, we analyze the performance of each algorithm for all the MAPF instances we generated for training and testing the algorithm selector. We aim to provide more insight on when a specific algorithm might work well for a certain scenario.

4.7.1 Algorithm Performance and SpaceRatio Analysis

Since the input for MAPFAST contains the single-agent shortest paths, there may be some corresponding patterns of these paths for the test cases that have the same fastest algorithms. Here we define *SpaceRatio*, which is equal to the number of map cells that are on the single-agent shortest paths divided by the number of the map cells that have no obstacles in it. The SpaceRatio not only represents how much free map space is used by the single-agent shortest paths, but also how spread the start and goal locations are in a map. We present the scatter plots for the average length of single-agent shortest paths with respect to SpaceRatio of two different maps in Figure 4.9a and 4.9c. Each data point is colored by the algorithm that solves the corresponding instance fastest. The distributions of the average single-agent shortest path lengths and SpaceRatio with respect to each algorithm are also shown on the top and right sides of the figures. We see that CBS tends to perform better than CBSH and BCP for the test cases having a shorter average length of

single-agent shortest path and smaller SpaceRatios. CBSH and BCP have similar performance on different average single-agent shortest path lengths. However, CBSH performs better than BCP on the test cases with higher SpaceRatios.

Since the SpaceRatio is also affected by the total number of agents, we further present the scatter plots for the number of agents with respect to SpaceRatios in Figure 4.9b and Figure 4.9d. When there are fewer agents in the map, CBS works better for smaller SpaceRatios while BCP and CBSH dominate the test cases with larger SpaceRatios.

4.7.2 Algorithm Performance and Heatmap Analysis

In Figure 4.10, we show the heat maps of the single-agent shortest paths for all of the test cases where a certain algorithm is ranked as the fastest one. The more a map cell is used by a single-agent shortest path, the brighter it is. The heat maps for CBS (Figure 4.10a and 4.10b) have lots of scattered shortest paths compared to BCP and CBSH. This is because the solving speed of CBS is determined by the number of conflicts found during the search phase. The test cases with longer single-agent shortest paths tends to result in more potential conflicts, thus making CBS slower. On the other hand, the heat maps for CBSH and BCP are mostly dominated by the longer paths. Although it seems that the heat maps for CBS have occupied more map space than CBSH and BCP, it has no correlation with SpaceRatio since the heat maps contain paths from different test cases. The notable differences of CBS's heat maps with other algorithms also demonstrate our motivation of adding single-agent shortest paths as an input tensor for MAPFAST. The differences in the heat maps are so readily apparent that a human can manually decide whether or not to use CBS without the help of an algorithm selector. However, the heat maps alone do not lead to any obvious suggestion about when to use BCP or CBSH. These two algorithms have very similar performance on test cases with different numbers of agents and SpaceRatios. Although the test cases in Figure 4.9

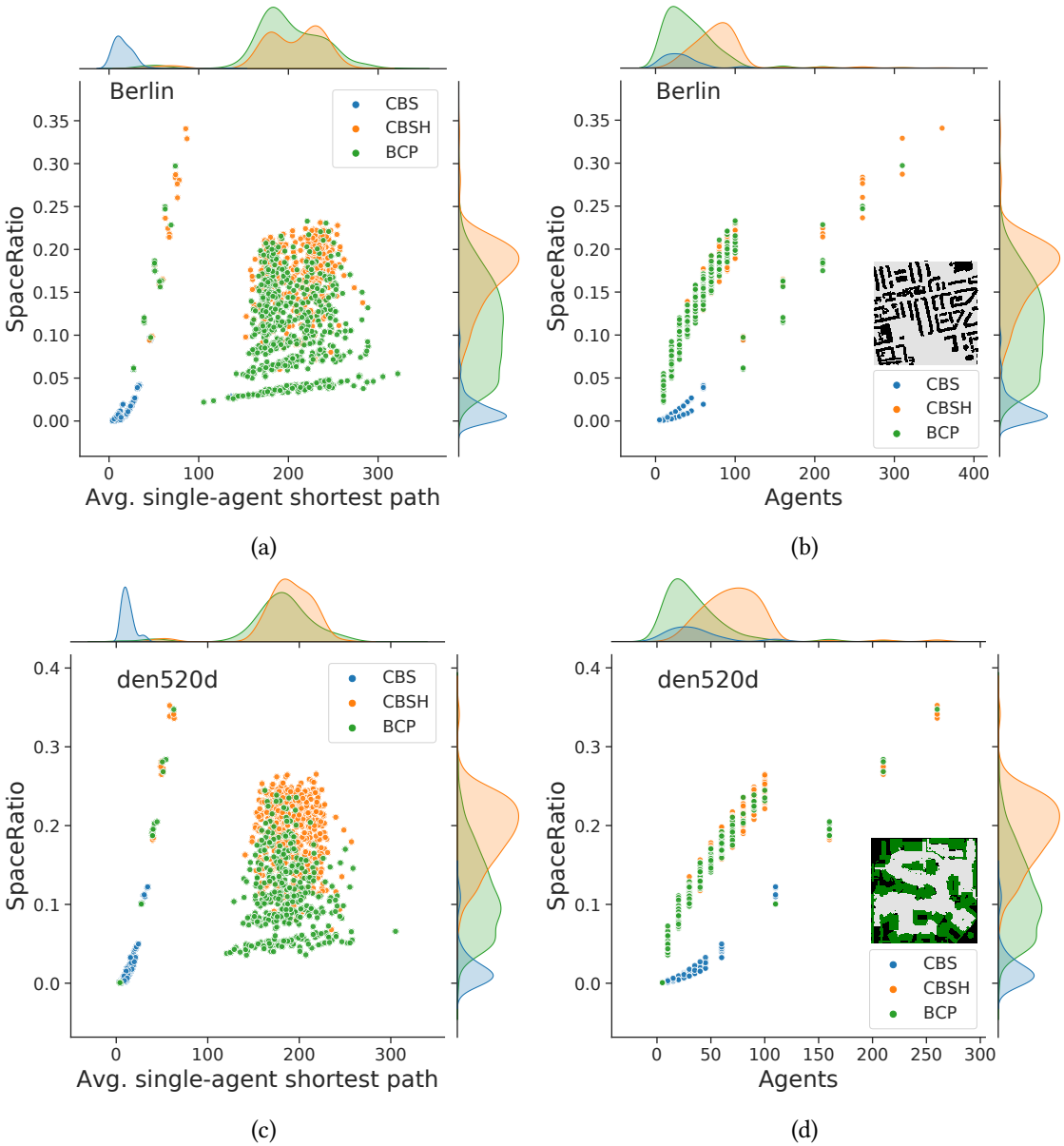
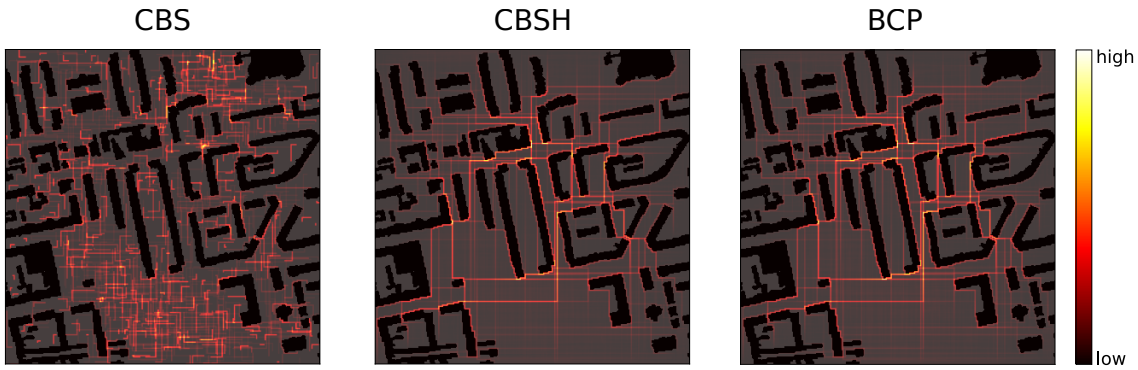
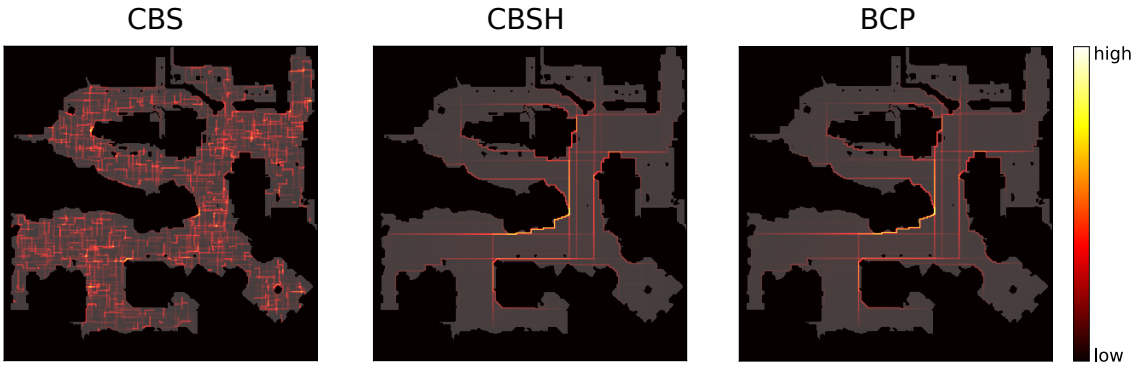


Figure 4.9: (a)-(d) The scatter plots for average single-agent shortest path length and number of agents with respect to SpaceRatio.



(a) Berlin



(b) den520d

Figure 4.10: Heat maps of the single-agent shortest paths with respect to different algorithms for (a) Berlin and (b) den520d.

indicate that CBSH works better for higher SpaceRatios, we have observed similar results for BCP in other maps which are shown in the Appendix (Figure A.6 and A.10).

Based on the dataset analysis, one interesting future work is to develop *hybrid* MAPF algorithms that combine the strengths of different algorithms. For instance, one can use the number of agents or SpaceRatio as an additional heuristic to help decide whether to use the basic version of an algorithm such as CBS or an improved version such as CBSH.

4.8 Impacts and Extensions of MAPFAST

There are several follow-up works on MAPFAST. Alkazzi et al. [4] enhanced MAPFAST's performance and training speed by incorporating an autoencoder model. They also fixed a bug in our code. Inspired by the G2V model used in MAPFAST, Shabalin et al. [136] proposed a new instance encoding method that combines graph-based embedding with hand-crafted features. Chen et al. explored the performance differences of using various deep learning models for sub-optimal algorithm selection [19]. Although MAPFAST was not originally designed for sub-optimal algorithms, it still shows comparable performance with other models. Notably, all these follow-up works used variations of single-agent shortest path encoding, demonstrating the broad impact of our contribution.

Adding New Algorithms to MAPFAST The current design of MAPFAST requires modifications to the output layer and retraining whenever new algorithms are added to the portfolio. This is because the number of neurons in MAPFAST's output layer corresponds to each of the individual algorithms in the portfolio. A potential future direction could involve developing algorithm selectors that can update the algorithm portfolio without requiring structural changes to the neural networks.

Impacts of New MAPF Algorithms Another concern of algorithm selection research is that maybe there will be a powerful algorithm in the future that outperforms all the existing algorithms so that there is no need to use algorithm selection. For optimal MAPF algorithms, this scenario is highly unlikely. Advanced optimal MAPF algorithms often depend on specific heuristics or tie-breaking techniques, such as CBSH2-RTC [92]. While these techniques enhance runtime performance for certain instances, they may lead to slower performance in others. As demonstrated in Chapter 3, Figure 3.2, we presented a corner case illustrating this trade-off.

Algorithm Selection for Sub-optimal Algorithms MAPFAST can be applied to sub-optimal MAPF algorithms as well. We can add several versions of a sub-optimal algorithm, each parameterized with a different optimality bound, as individual algorithms to the portfolio. The loss function in this context would represent a combined objective, balancing the trade-off between runtime and solution quality. Chen et al. [19] demonstrated that MAPFAST achieved decent prediction performance on sub-optimal algorithms.

It would also be interesting to explore how randomization techniques, such as random restarts during the search process, influence the performance of optimal algorithm selectors.

4.9 Summary

In this chapter, we have demonstrated how algorithm selection techniques can be applied to predict the best algorithm for a given MAPF instance without solving it. We have shown that the single-agent shortest path encoding significantly improves the prediction performance. Additionally, we present a graph-based algorithm selection model called G2V that relies solely on single-agent shortest path encoding. G2V can achieve comparable performance to MAPFAST without the information of map topology.

Chapter 5

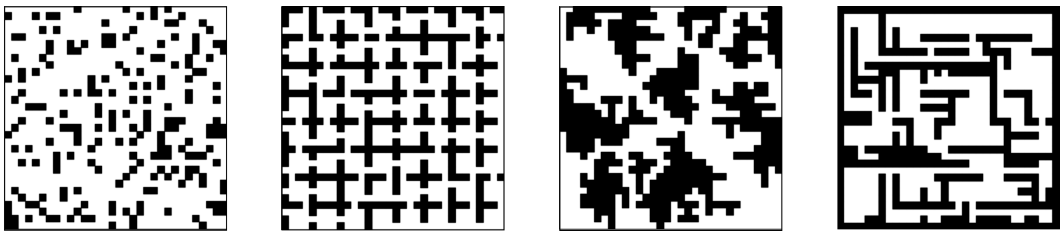
Map Connectivity and the Empirical Hardness of MAPF Problem

In this chapter, we explore the correlation between map connectivity and the empirical hardness of MAPF instances. We validate our hypothesis that map connectivity plays a key role in determining the empirical hardness of MAPF instances. Additionally, we demonstrate how the Quality Diversity method can be used to generate maps with varying levels of connectivity. This allows us to generate more challenging instances by controlling the map connectivity. This chapter addresses two fundamental questions: "What makes MAPF instances hard?" and "How to generate hard instances?"

5.1 Introduction

There are two major components of a MAPF instance: a map, which is normally represented as a 2D environment, and a certain distribution for a set of agents with their start and goal locations. While many optimal MAPF algorithms can solve some instances with hundreds of agents in a map, they can also struggle with only a small number of agents on a different map [127, 33]. In Figure 5.1 we present a more detailed example. Given the four different maps, we used uniform random sampling to generate the start and goal locations of the agents. Albeit there are more agents in `random-32-32-20` than `maze-32-32-5`, the average runtime of solving 100 instances using CBSH2-RTC [92] algorithm is much

This chapter follows closely to [126].



	random-32-32-20	room-32-32-4	frac-32-32-4	maze-32-32-5
average runtime (s)	0.003	0.01	0.04	77.18
num of agents	18	15	13	14

Figure 5.1: Runtime for different maps. The algorithm used is CBSH2-RTC [92]. All the instances have the same ratio of the number of agents to free space.

faster for `random-32-32-20` (0.03 seconds) than `maze-32-32-5` (77.18 seconds). This indicates that the map topology can have a major impact on the empirical hardness of MAPF instances. In this chapter, we seek to understand what features of the map environment make the MAPF instance hard to be solved optimally.

Since it is non-trivial to study the map topology and different distributions of the agents' locations together, we will focus on studying the map topology first while assuming that the start and goal locations of the agents are being uniformly randomly sampled.

We are interested in understanding what features of MAPF instances make them hard to be solved optimally. We are also interested in finding an effective way to compare the hardness of different maps when randomly generating MAPF instances on them. For example, when using uniform random sampling to generate agents and goals on two given maps, we seek to predict which map will have harder instances on average. This area of research is known as *empirical hardness*, which focuses on identifying features that determine how hard individual instances will be for particular algorithms to solve [86]. More background information can be found in Chapter 3. In this chapter, we study the correlation between map connectivity and the empirical hardness of the MAPF problem.

Contribution

We conducted an in-depth study investigating the relationship between map connectivity and the empirical hardness of MAPF instances. Our findings provide both theoretical and empirical evidence that map connectivity significantly influences instance hardness. Another key contribution is the development of a Quality Diversity (QD) map generator, which creates maps with varied connectivity or a user-defined range. This generator offers an effective method of generating challenging MAPF instances which can be further used to evaluate existing MAPF algorithms and inspire new heuristics. It also enables the creation of diverse MAPF benchmark datasets with maps of varying connectivity.

5.2 Related Research

5.2.1 Betweenness Centrality

Betweenness centrality measures how often a map cell lies on the shortest path between all pairs of map cells. It has been widely used to study the structure of social networks [12, 45]. Betweenness centrality can also be applied to study the relationship between map structures and the empirical hardness of the MAPF problem. Understanding the relationship between map features and empirical hardness is an under-explored topic. Ewing et al. [33] * conducted one of the earliest research on betweenness centrality and MAPF. They showed that maps with high betweenness centrality are more likely to generate inter-agent conflicts, resulting in MAPF instances that are significantly harder to solve optimally using current algorithms. While calculating betweenness centrality for large maps is computationally intensive, the process can be significantly accelerated using GPU and parallel computing techniques [105]. Although the calculation speed of betweenness centrality is slow, it only needs to be computed once and can be reused across different instances on the map.

*I'm a co-author of this paper. My contribution is the analysis of the correlation between single-agent shortest path and instance hardness.

Figure 5.2: A 2D grid map and its graph representation: black cell represents an obstacle, while white cells denote free space.

5.2.2 Spectral Graph Theory

Spectral graph theory is a powerful tool for analyzing and understanding graphs' complex structure. It studies the properties of graphs based on the eigenvalue and eigenvectors of the matrices associated with the graph [151, 21]. Some of the most commonly used matrices are adjacency matrix, Laplacian matrix, and normalized Laplacian matrix. The study of spectral graph theory over the past years has brought many useful algorithms such as graph partitioning [29, 106, 11], random walk [99, 151], community detection [141, 116], graph visualization [75, 39], parallel computing [51, 115], and combinatorial optimization [28, 109]. The idea of using eigenvalues to analyze graph properties also inspired other research domains including graph neural networks [7, 162] and robotics, particularly in formation control [24, 134, 174] and Simultaneous Localization and Mapping (SLAM) [121, 13].

In the following sections, we demonstrate how to use spectral graph theory to analyze and compare the connectivity of different maps.

5.3 Preliminary

Here, we introduce the preliminary knowledge of graph connectivity, focusing on 2D grid-based MAPF problems. The 2D grid map is modeled as a 4-connected undirected graph $G(V, E)$. Each free cell of the 2D grid map corresponds to a vertex $v_i \in V$ in $G(V, E)$. If a vertex v_i is connected to neighboring vertex v_j, an edge $e_{ij} \in E$ exists between them. Figure 5.2 shows a 2 by 3 grid map and the corresponding graph.

Definition 5.1. The degree of a vertex v_i is the number of vertices in $G(V, E)$ that are adjacent to v_i. If v_i and v_j is adjacent then there is an edge $e_{ij} \in E$.

Definition 5.2 (Degree Matrix). The degree matrix D of a graph $G(V, E)$ with n vertices is defined as a $\mathbb{R}^{n \times n}$ diagonal matrix where $D_{i,i} = \text{degree}(v_i)$ and other off the diagonal elements are 0.

Definition 5.3 (Adjacency Matrix). The adjacency matrix A of a graph $G(V, E)$ with n vertices is defined as a $\mathbb{R}^{n \times n}$ matrix. If a vertex v_i is connected with another vertex v_j, then $A_{i,j} = 1$, otherwise 0.

Definition 5.4 (Normalized Laplacian). In spectral graph theory, the normalized Laplacian matrix $\bar{L}$ of a graph is defined by:

$$\begin{aligned} L &= D - A \\ \bar{L} &= D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2} \end{aligned} \tag{5.1}$$

where the D is the diagonal degree matrix and A is the adjacency matrix. More specifically, we have:

$$\bar{L}_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{d(i)} & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

For example, the degree and adjacency matrix of the graph in Figure 5.2 are:

$$\mathbf{D} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{5.4}$$

$$\bar{\mathbf{L}} = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -\frac{1}{3} & 1 & -\frac{1}{3} & -\frac{1}{3} \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \tag{5.5}$$

The normalized Laplacian matrix $\bar{L}$ encapsulates both the structural and spectral information which are very important to help understand the graph connectivity and other graph features. The eigenvalues of the normalized Laplacian matrix also contain important information about the graph.

Property 5.1. The smallest eigenvalue of the normalized Laplacian matrix for a grid graph is always 0.

First, we show that $\bar{L}$ is positive semi-definite. Let x be any non-zero vector of size $\mathbb{R}^n$, we have:

$$\begin{aligned} x^T \bar{L} x &= x^T x - x^T D^{-1/2} A D^{-1/2} x \\ &= \sum_i x_i^2 - \sum_{i,j} \frac{x_i}{\sqrt{d_i}} A_{ij} \frac{x_j}{\sqrt{d_j}} \end{aligned}$$

Recall that $A_{ij} = \{1 \text{ when } e_{ij} \in E \text{ and } 0 \text{ otherwise }\}$:

$$\begin{aligned} x^T \bar{L} x &= \sum_i x_i^2 - \sum_{e_{ij} \in E} 2 \frac{x_i}{\sqrt{d_i}} \frac{x_j}{\sqrt{d_j}} \\ &= \sum_{e_{ij} \in E} (\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}})^2 \end{aligned}$$

This indicates $x^T \bar{L} x \geq 0$ and $\bar{L}$ is positive semi-definite and all eigenvalues are non-negative. Let $v = [1, 1, 1, \ldots, 1]$ we have $\bar{L}v = 0$ which indicates 0 is an eigenvalue of $\bar{L}$. Given that all eigenvalues of $\bar{L}$ are non-negative, thus the smallest eigenvalue of $\bar{L}$ is 0. □

Property 5.2. The number of connected components in a graph is equal to the multiplicity of 0 of the eigenvalue of the normalized Laplacian matrix.

The disconnected sub-graphs of a graph can be viewed as many individually connected graphs. According to corollary 5.1, each of the sub-graph has its smallest eigenvalue equals to 0. Thus the number of 0s of eigenvalue for normalized Laplacian matrix equals to the number of connected components. □

Property 5.3. The largest eigenvalue of the normalized Laplacian matrix for a grid graph is always 2.

Here we use the property of a bipartite graph where the largest eigenvalue of the normalized Laplacian matrix is 2 [150]. The grid map can be viewed as a bipartite graph and the same property still holds. □

5.3.1 Conductance and Cheeger's Inequality

More specifically, for a connected graph, the second smallest eigenvalue (referred as λ_2) of the normalized Laplacian $\bar{L}$ defines the algebraic connectivity of the graph, describing how well the graph is connected [44]. To better understand why λ_2 is related to the connectivity of graphs, we will further introduce the concept of *boundary* and the *conductance* of a graph.

Definition 5.5 (Boundary). The *boundary* for a set of vertices $S \subset V$ of an undirected graph $G(V, E)$ is defined as:

$$\partial S = \{e_{ij} \in E : v_i \in S, v_j \notin S\}. \tag{5.6}$$

Given a set of vertices $S \subset V$, the boundary represents the set of edges where one endpoint is in S and the other endpoint is outside of S.

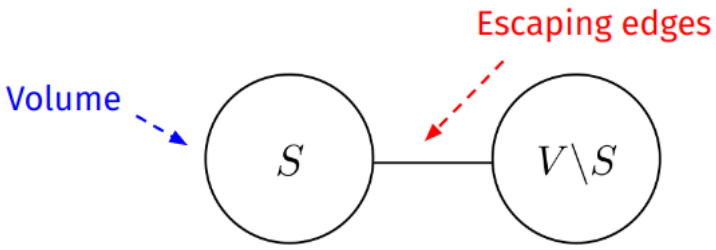


Figure 5.3: An example of a poorly connected graph.

Definition 5.6 (Conductance). The *conductance* of S is defined as:

$$\phi(S) = \frac{|\partial S|}{\min(d(S), d(V \backslash S))} \tag{5.7}$$

where $|\partial S|$ is the number of edges on the boundary and $d(S)$ denotes the number of edges with both endpoints (nodes) within S. $\phi(S)$ represents the ratio of the number of edges on the boundary of set S to the minimum of its internal and external edges.

Definition 5.7 (Conductance of a graph). The *conductance* of a graph $G(V, E)$ is subsequently defined as the smallest conductance over all partitions of vertices:

$$\phi(G) = \min_{\emptyset \subsetneq S \subsetneq V} \phi(S) \tag{5.8}$$

The conductance represents how well-connected a graph is. Using Figure 5.3 as an example, the boundary of a vertex partition S can be viewed as the number of edges escaping from S to the other vertex partition $V \backslash S$. The $d(S)$ and $d(V \backslash S)$ in Equation 5.7 can be viewed as the volume of a vertex partition, since it represents the number of edges in the partition. When the number of escaping edges is small and the values of $d(S)$ and $d(V \backslash S)$ are roughly equal to each other, the conductance of the graph will be small, thus indicating a weakly connected part in the graph.

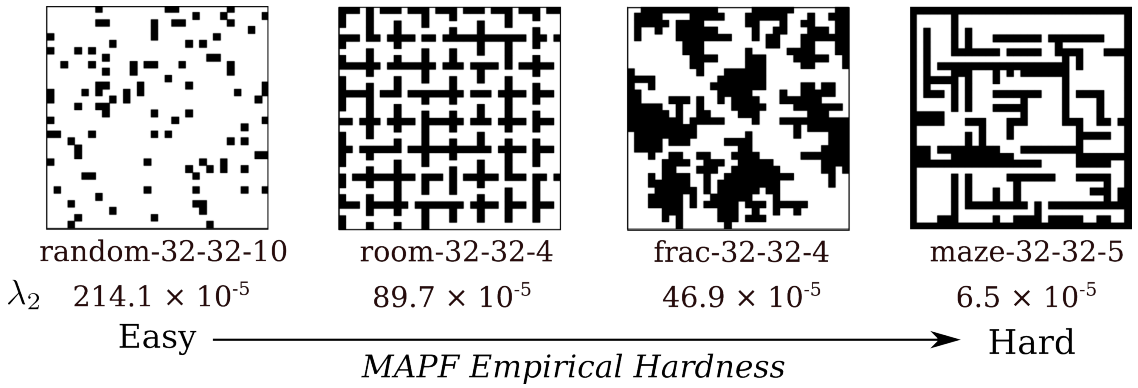


Figure 5.4: Different maps and their corresponding λ_2 values.

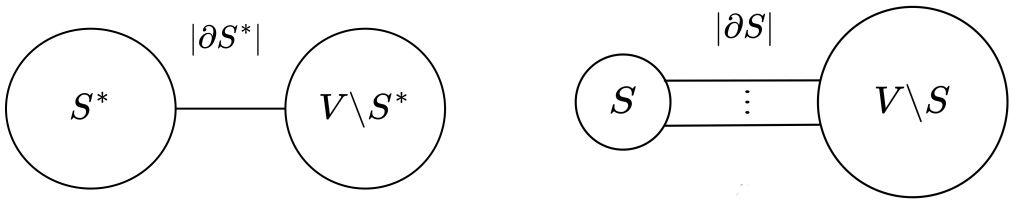
Theorem 5.1 (Cheeger's Inequality). Let λ_2 be the second smallest eigenvalue of the normalized Laplacian $\bar{L}$ of undirected graph $G(V, E)$, then:

$$\frac{\lambda_2}{2} \leq \phi(G) \leq \sqrt{2\lambda_2}. \tag{5.9}$$

Cheeger's inequality brings the graph connectivity and λ_2 together. This implies that λ_2 can be used as a quantitative method for characterizing the impacts of a map's features, such as narrow corridors, on the overall map connectivity. Generally, a relatively small λ_2 indicates the graph is poorly connected, whereas a large λ_2 implies strong connectivity [44]. Figure 5.4 shows the λ_2 values of different maps. The difference in λ_2 between the well-connected, easy map `random-32-32-10` on the far left and the less connected, hard map `maze-32-32-5` on the far right is significant.

5.4 Conductance and MAPF Conflicts

Here we present an intuitive proof of how the maps with smaller conductance are more likely to generate more conflicts for MAPF instances. Consider a simple dumbbell graph G_d shown in Figure **??**, where two partitions are only connected with a single edge. G_d is derived from a 4-connected map, thus the number of edges is roughly proportional to the number of vertices in its subgraphs. The size of the circle indicates



(a) (b)

Figure 5.5: Two different partitions of a graph. The size of each circle represents how many edges are in the partition (also known as volume). (a) represents a more balanced partition, while (b) depicts a less balanced one.

a different number of edges within the partition. Let's also assume this partition S^* has the smallest conductance of G_d, thus we have $\phi(G_d) = \phi(S^*)$. Next, consider another partition S of G_d shown in Figure 5.5b, where the two partitions are connected by more edges; thus, $|\partial(S)| > 1$ and $\phi(S^*) < \phi(S)$. Another observation is that S^* is a more balanced partition than S in terms of the number of edges within the partition, and we further have:

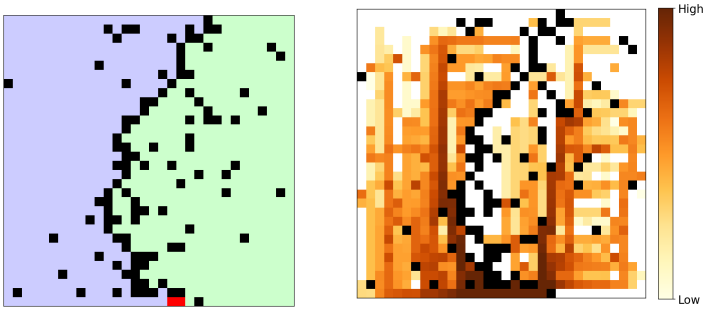
$$d(S^*)d(V \backslash S^*) > d(S)d(V \backslash S). \tag{5.10}$$

When uniformly and randomly sampling the start and goal locations on G_d, the shortest path will traverse a boundary edge only if the start and goal locations are on different sides of the boundary. The probability of the shortest path visiting a boundary edge of partition S is:

$$P(\partial S) = \frac{1}{|\partial S|} \frac{2d(S)d(V \backslash S)}{d(V)^2} \tag{5.11}$$

Given Equation 5.10, we have:

$$\frac{2d(S^*)d(V \backslash S^*)}{d(V)^2} > \frac{2d(S)d(V \backslash S)}{d(V)^2} > \frac{1}{|\partial S|} \frac{2d(S)d(V \backslash S)}{d(V)^2}.$$



(a) Conductance cut (b) Heatmap of conflicts

Figure 5.6: (a) Conductance cut of a map. The purple and green regions represent different vertex partitions. Red map cells indicate where the cut is (i.e., escaping edges). (b) Heatmap of the conflicts when using the CBSH2-RTC algorithm.

The left-hand side is $P(\partial S^*)$ since $|\partial S^*| = 1$ and the right-hand side is $P(\partial S)$. This indicates $P(\partial S^*) > P(\partial S)$. This inequality implies a higher likelihood of agents visiting the boundary edges of poorly connected cuts within the same graph, leading to increased potential conflicts, particularly in scenarios with more agents. Intuitively, one can think of these boundary edges as choke points that need to be traversed to get from one partition to the other. Relating this to the definition of $\phi(G)$ and Cheeger's inequality, we can loosely demonstrate that $P(\partial S^*) \propto \frac{1}{\phi(G)}$. This suggests that maps with smaller $\phi(G)$ or λ_2 may tend to exhibit more conflicts; thus MAPF instances on those maps are more likely to be challenging.

Figure 5.6a shows the conductance cut of a poorly connected map. The purple and green regions represent different sub-graphs, with the red cells indicating the escaping edge. This map resembles the dumbbell graph in Figure 5.5a where two sub-graphs are only connected by a single edge. Figure 5.6b shows the heat map of conflicts when using the CBSH2-RTC [92] algorithm. We can easily see that more conflicts occur in the weakly connected region, which is also the location of the conductance cut. This further illustrates why poorly connected maps tend to have more challenging MAPF instances.

5.5 Environment Generation

Generating diverse and challenging environments is important for studying the empirical hardness of MAPF instances. We introduce two map generation methods: Cellular Automata and Diffusion-Limited Aggregation [†]. These methods can generate maps with distinctive styles and layouts.

5.5.1 Cellular Automata

Cellular Automata (CA) [66] is commonly used for procedure environment generation of computer games. Ewing et al. [33] presents a CA-based map generator to generate cave-like environments with smooth walls and open chambers connected by narrow corridors. The map is initialized with a preset obstacle density by uniformly generating obstacles at random. A cellular automata rule is then applied to all map cells based on how many obstacles are around their adjacent cells. For example, if a map cell has five adjacent obstacles, it becomes an obstacle, otherwise it changes to free space. To further smooth the map, the cellular automata rule is normally applied multiple times. The controllable parameters for CA-based generators are the obstacle density and the number of iterations for applying the CA rules.

5.5.2 Diffusion-Limited Aggregation

Diffusion-Limited Aggregation (DLA) [165] uses random walks to simulate the aggregation of particles due to Brownian motion in physics. The second map generator uses the diffusion-limited aggregation method to generate fractal tree-like environments with lots of choker points [33]. The map is initialized with a random number of obstacles as seeds. A random walk is then performed at a random free cell until it reaches a cell that is adjacent to an obstacle. The random walk continues from this obstacle until a desired obstacle density is satisfied.

[†]There is extensive research on environment generation, particularly within the gaming industry [52]. Here, we mainly focus on methods specific to the MAPF research domain.

Algorithm 1: Maze Generation Using Diffusion-Limited Aggregation

Input: dim_x, dim_y, $step\_size$, $skip\_rate$
Output: $maze$

```
/* Initialize the maze                                                    */
```
$maze \leftarrow$ array of zeros of size $dim\_x \times dim\_y$ with 1-cell width bounding box set to 1;

```
/* Generate anchor points                                                 */
```
$anchor \leftarrow$ empty list;
for $i \leftarrow 1$ **to** $dim\_x - 2$ **do**
  for $j \leftarrow 1$ **to** $dim\_y - 2$ **do**
    if $i \bmod step\_size = 0$ ***and*** $j \bmod step\_size = 0$ **then**
      if *random(0, 1)* $> skip\_rate$ **then**
        Append $[i, j]$ to $anchor$;
        $maze[i, j] \leftarrow 1$; `/* Add anchor point */`

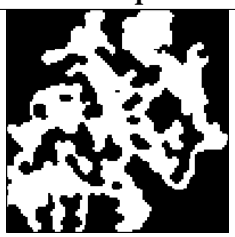
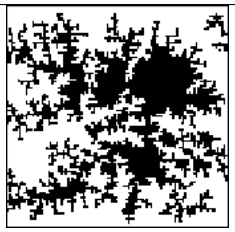
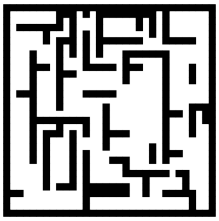
```
/* Shuffle the anchor points                                              */
```
Randomly shuffle $anchor$;
$directions \leftarrow \{[0, 1],\ [0, -1],\ [1, 0],\ [-1, 0]\}$; `/* Up, down, right, left */`

```
/* Generate the maze                                                      */
```
foreach $anchor\_i$ *in* $anchor$ **do**
  $curr\_pos \leftarrow anchor\_i$;
  $curr\_dir \leftarrow$ random choice from $directions$;
  while *True* **do**
    $new\_pos \leftarrow curr\_pos + curr\_dir$;
    if $maze[new\_pos[0],\ new\_pos[1]] == 1$ **then**
      break;
    else
      $maze[new\_pos[0],\ new\_pos[1]] \leftarrow 1$;
      $curr\_pos \leftarrow new\_pos$;

return $maze$;

The diffusion-limited aggregation method can also be used to generate maze-like environments. By using a different step size and skipping rate, we can generate maze-like environments with lots of straight walls and narrow corridors. An example of such a maze generator is presented in Algorithm 1. Table 5.1 presents a comparison of different map generation methods.

Table 5.1: Comparison of different environment generation methods for MAPF

Method	Example	Style	Parameter
Cellular Automata [33]		Cave	Obstacle density, Number of iterations
Fractal [33]		Tree	Obstacle density
Diffusion-Limited Aggregation [126]		Maze	Skipping rate, Step size

5.6 Experiments

In this section, we present extensive experimental results to reveal the relationship between map connectivity and the empirical hardness of MAPF instances.

5.6.1 Experiment Setup

To thoroughly investigate the relationship between map connectivity and empirical hardness of MAPF, we have selected four different optimal MAPF algorithms which are proven to be quite powerful according to various benchmark analysis [33, 139]: **LazyCBS** [41], **BCP** [78], **CBSH2-RTC** [92] and **CBSH2-RTC-CHBP** [140].

To ensure the diversity of our test dataset, we included maps generated using different techniques. Firstly, we have included all 32×32 maps (5 in total) from the MAPF benchmark dataset [155]. Additionally, we generated 31 fully connected 32×32 maps using the Cellular Automata and Diffusion-Limited Aggregation methods described in Section 5.5. Details of these generated maps are shown in Figure 5.7.

When generating MAPF instances, we ensured that all the maps have the same agent-to-freespace ratio, where $r = \frac{\#agents}{\#free\ cells}$. This value is chosen based on our test such that the instances are neither excessively challenging nor overly easy so that we can still effectively compare the performance across different maps. For each map, we generated 100 instances using uniform random sampling to determine the start and goal locations of agents. The feasibility of the generated instances was validated by using a sub-optimal MAPF algorithm ECBS with a relaxed bound ($w = 1.6$) [10]. Simulations were conducted on a PC with Ryzen 3950x CPU and 64GB RAM, with the runtime limit set to 300 seconds.

Experiment 1: Average Runtime and Number of Agents

The number of agents for a MAPF instance is often used as the metric to understand the theoretical hardness of the MAPF problem. As the number of agents in a MAPF problem increases, finding optimal solutions in the worst-case scenario becomes computationally intractable [171]. However, the number of agents is not an effective metric to compare the empirical hardness of different instances. An optimal MAPF algorithm can solve instances with up to hundreds of agents, yet struggle with instances having a small number of agents. Figure 5.8 shows the relationship between the number of agents and the average runtime for different maps when using different algorithms. The results show that having more agents does not necessarily make the instances harder. Maps with more obstacles and poorly connected regions can result in much harder instances, even with a small number of agents. In contrast, well-connected maps with a larger number of agents may still be easier to solve.

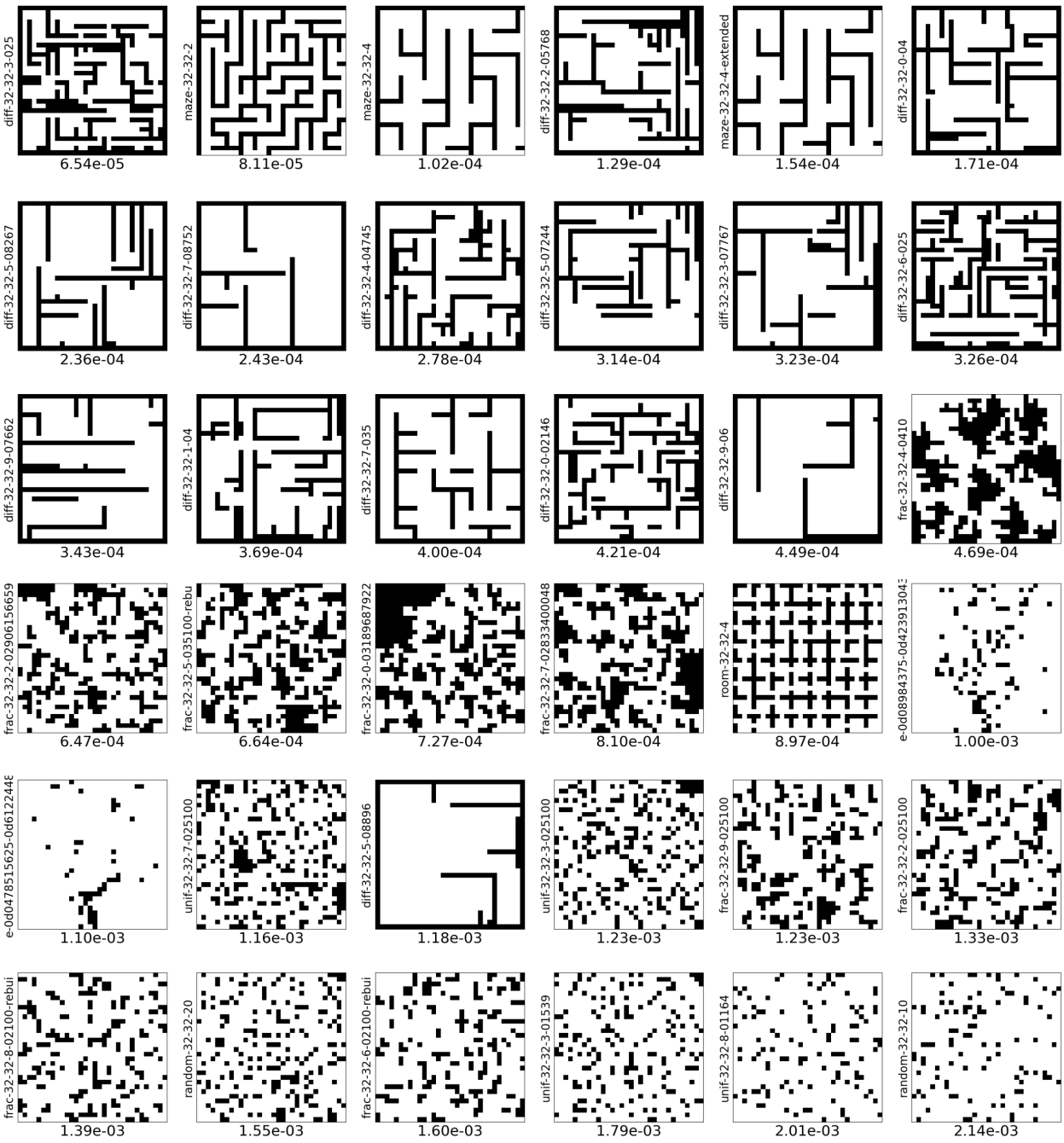


Figure 5.7: Details of the maps used in our experiments. The λ_2 value is annotated on x-axis.

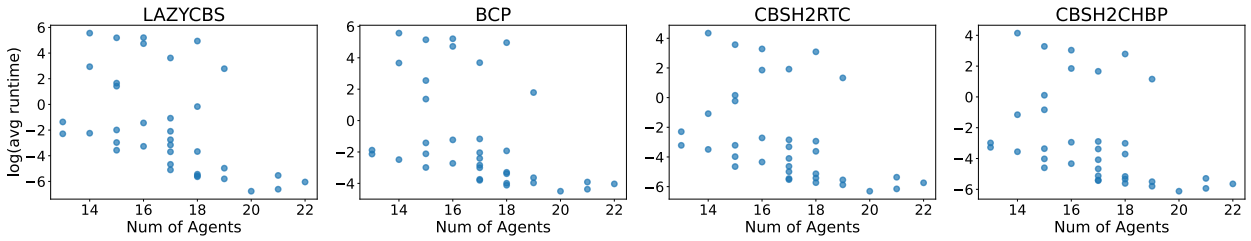
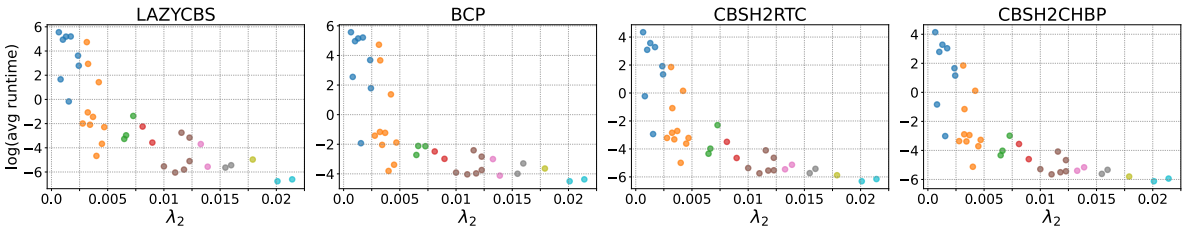
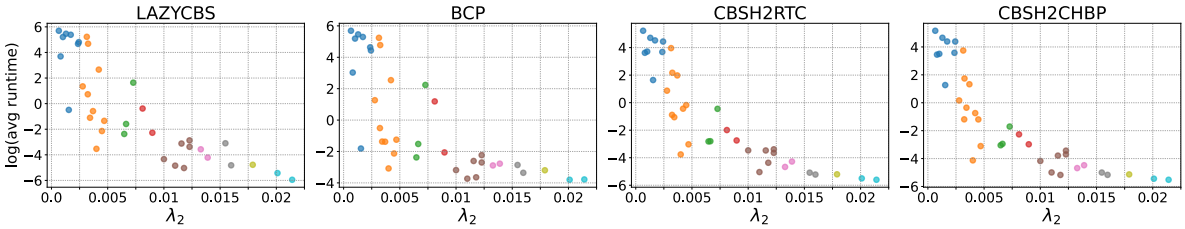


Figure 5.8: The logarithm of average runtime and number of agents for different maps. Agent-to-freespace ratio is 2.25×10^{-2}.



(a) Agent-to-freespace ratio: 2.25×10^{-2}. The number of agents ranges from 13 to 22.



(b) Agent-to-freespace ratio: 3.00×10^{-2}. The number of agents ranges from 17 to 29.

Figure 5.9: Simulation results of the runtime for different algorithms on the maps with different λ_2. Different colors are used to represent λ_2 values within different ranges.

Experiment 2: Average Runtime and λ_2

As an initial proof of concept to show that the λ_2 value of a map has some correlation with the empirical hardness, or runtime, of MAPF instances on that map, we randomly generated MAPF instances on 36 maps with varying λ_2 values and compared runtimes. The simulation results in Figure 5.9 illustrate the relationship between the logarithm of average runtime and λ_2 of different maps. We have made several interesting observations on the results.

First, hard instances often appear on maps with smaller λ_2 (top left corner), whereas maps with larger λ_2 can be considerably easy (bottom right corner). This pattern remains consistent across different algorithms and r settings. Additionally, it is noteworthy that CBSH2-based algorithms generally exhibit faster

runtime than LazyCBS and BCP (notice the different scales on the y-axis). Despite differences in absolute runtime, our results indicate within each algorithm, challenging instances happen more frequently on maps with smaller λ_2.

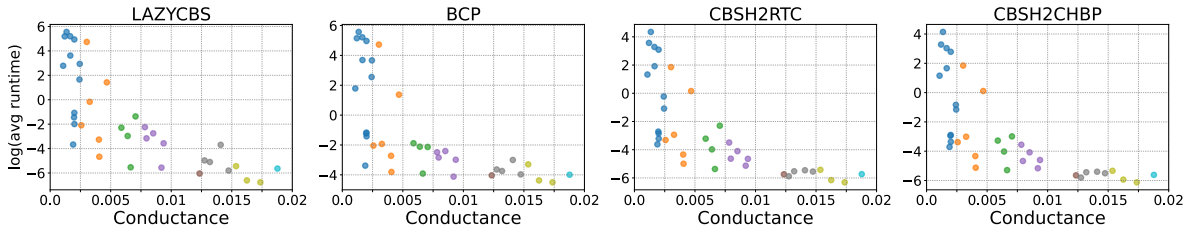
Second, maps with smaller λ_2 could still have relatively easy instances. Given that λ_2 is not the only factor influencing empirical hardness, we are not surprised to see that the average runtime and λ_2 do not exhibit a strict monotonic correlation. One possible reason might be the effect of narrow corridors on a 2D grid map, for instance increasing the width of a narrow corridor from 1-cell to 2-cell width will only change λ_2 slightly, but the wider corridors are less likely to create enough contested regions, thus the empirical hardness could drastically shift from hard to easy. We further explore this in Experiment 4.

Experiment 3: Average runtime and Conductance

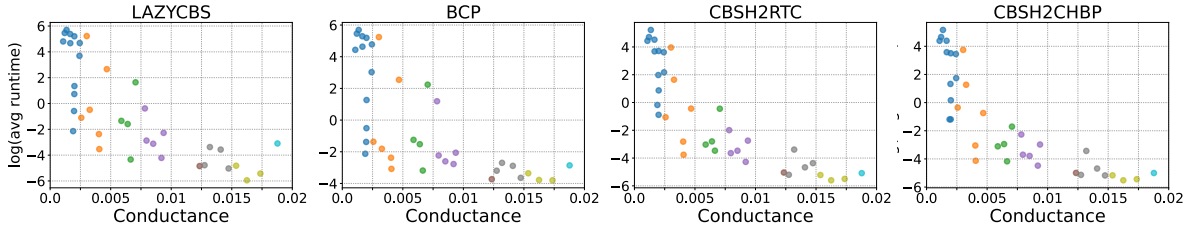
Figure 5.10 presents the relationship between conductance and average runtime of each map. It exhibits a similar trend to λ_2. When the conductance of a map is small, instances generated using uniform random sampling tend to be harder. This supports the claim that λ_2 can be used as an effective approximation of conductance. Additionally, λ_2 is significantly easier to calculate compared to conductance. This makes λ_2 a practical metric for approximating the average empirical hardness across different maps.

Experiment 4: Average Number of Constraint Tree (CT) Expansions and λ_2.

Next, we illustrate the relationship between the average number of Constraint Tree (CT) expansions and λ_2 for all instances on a map. For the CBSH2-based algorithms, the number of CT expansions is related to how many conflicts have been resolved during the searching process and reflects the instance hardness [48]. From Figure 5.11, the trend for the number of CT expansions is similar to the runtime trend in Figure 5.9. This correlation is believed to be caused by the poorly connected map regions where conflicts are more likely to happen.

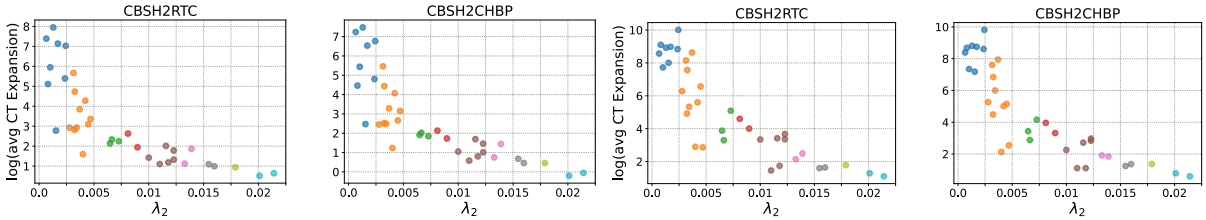


(a) Agent-to-freespace ratio: 2.25×10^{-2}. The number of agents ranges from 13 to 22.



(b) Agent-to-freespace ratio: 3.00×10^{-2}. The number of agents ranges from 17 to 29.

Figure 5.10: Simulation results of the runtime for different algorithms on the maps with different conductance.



(a) Agent-to-freespace ratio: 2.25×10^{-2}.

(b) Agent-to-freespace ratio: 3.00×10^{-2}.

Figure 5.11: The logarithm of average number of CT expansions and λ_2 for different maps.

Experiment 5: Expand the Width of Narrow Corridors.

In Experiment 2, we mentioned that there are still many easy instances on maps with low λ_2. To investigate this phenomenon, we manually changed the connectivity of maps without affecting the number of obstacles. More specifically, we expand some of the narrow corridors (red boxes in Figure 5.12 in a `maze` map from 1-cell to 2-cell width and observed that λ_2 changed from 10.1×10^{-5} to 15.4×10^{-5}. Although the change in λ_2 is small, there is a significant change in empirical hardness, where instances on expanded version `maze-e` (shown with dashed lines) are much easier. This demonstrates that even maps with small λ_2 can have easy instances. Figure 5.13 shows the conductance cut and heatmap of `maze` before and after

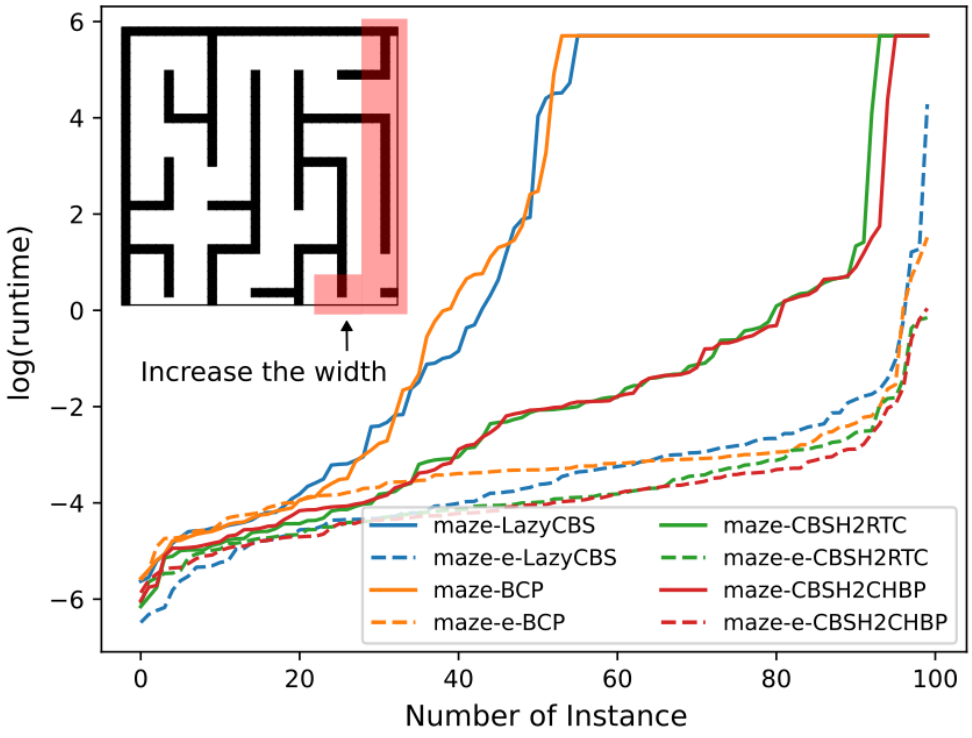


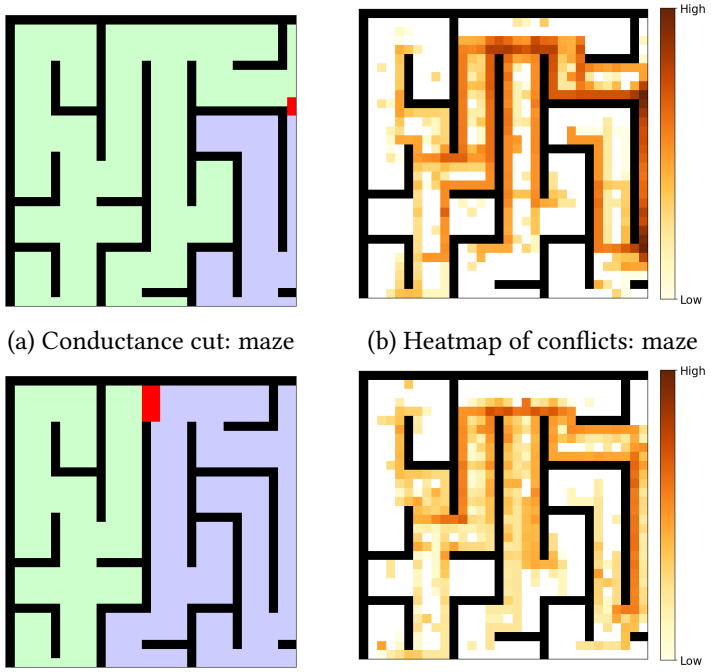
Figure 5.12: Sorted logarithm of runtime for `maze` and its expanded version `maze-e` by increasing the width of the narrow corridors in red boxes from 1-cell to 2-cell.

the change. It suggests that a 1-cell-width corridor is more likely to create contested regions and cause conflicts between agents, thus slowing down the algorithms (especially for conflict-based algorithms). These contested regions are significantly mitigated when increasing the corridor width, making the instances easier; in the meantime λ_2 exhibits minor change. We intend to develop a hybrid reasoning model on both λ_2 and corridor width in future research.

The complete results of the conductance cut and conflict heatmap for all maps in our dataset are provided in the Appendix, as shown in Figures B.1 and B.2.

5.6.2 Computation Cost of λ_2

Table 5.2 shows the time cost to calculate λ_2 of maps with different sizes. The time cost for λ_2 on maps smaller than 64×64 is fairly low (e.g., less than 1 second). For maps larger than 64×64, it is impractical



(a) Conductance cut: maze (b) Heatmap of conflicts: maze

(c) Conductance cut: maze-e (d) Heatmap of conflicts: maze-e

Figure 5.13: The conductance cut and heatmap of conflicts for `maze` and `maze-e`.

to calculate the λ_2 directly but approximation techniques such as Power Method [131, 123] and Lanczos algorithm [80, 22] can be used to speed up the process.

5.6.3 Remarks

In summary, our experiments demonstrate that λ_2 is an effective metric for comparing map connectivity and shows a clear correlation with the average empirical hardness of MAPF instances. Although λ_2 does not exhibit a strict monotonic correlation with empirical hardness, it still shows notable effectiveness, especially for very challenging instances associated with small λ_2. Considering the simplicity and ease of comparing λ_2 across different maps, we believe it is a reasonably effective metric and a great starting point for future research on MAPF empirical hardness.

Map	Size	# of Nodes	Runtime (sec)
empty-8-8	8×8	64	0.042
empty-16-16	16×16	256	0.048
maze-32-32-4	32×32	790	0.078
random-64-64-10	64×64	3687	0.933
maze-128-128-10	128×128	14818	159.296

Table 5.2: Time cost to calculate the λ_2 for different maps.

5.7 Generating Hard MAPF Instances

In the previous sections, we demonstrated that map connectivity is a key factor affecting the empirical hardness of MAPF instances. With this knowledge, we can generate challenging MAPF instances by deliberately reducing map connectivity. Moreover, by generating maps with diverse connectivity, we can create better benchmark datasets that cover varying levels of empirical hardness. In this section, we present a map generator that can create maps with tunable connectivity. Using these generated maps, we further validate the correlation between instance hardness and map connectivity.

5.7.1 Generating Maps With Tunable Connectivity

To further investigate the relationship between λ_2 and the empirical hardness of maps, we developed a map generator that can produce maps with a given λ_2 value. The maps generated should provide as much diversity as possible in measures other than λ_2 to try and isolate the relationship between λ_2 and hardness. We used a Quality Diversity (QD) method based on the algorithm MAP-Elites [113]. MAP-Elites is a search space illumination algorithm that seeks to find high-quality solutions that are diverse along certain prescribed features. It maintains a container, which contains a set of potential solutions. New potential solutions are inserted into the container if they are the best solution found so far in a specific bin of feature space with respect to some objective function. MAP-Elites typically utilize evolutionary algorithms to alter existing solutions in the container and produce new potential solutions.

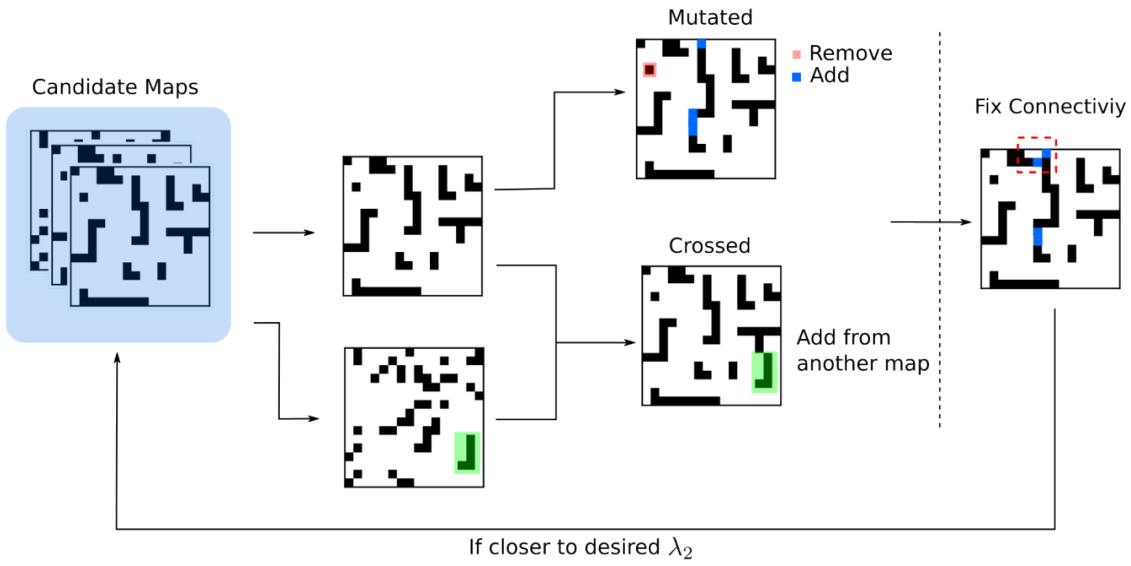


Figure 5.14: Structure of the QD map generator.

For our purposes, we sought to produce maps with specific λ_2 values that had diverse obstacle properties. We set the objective function to be the distance from a desired λ_2 value. We used features on the percentage of obstacles and the density of those obstacles (how many obstacles were adjacent to other obstacles). For each iteration, we took an existing map from the container, and with equal probability we either "mutated" the map or "crossed" the map with another random map from the container. A mutation consisted of adding or removing up to five obstacles on the map uniformly at random. Crossing two maps involved randomly selecting regions of one map to add to the other map. We then checked for connectedness and added back the minimum number of vertices to reconnect any disconnected components. The new maps, either with randomly added or removed obstacles or the cross between existing maps, were then evaluated on closeness to the desired λ_2 value. If they were closer than any other map with similar features, they were kept and the other map in the container with those feature values was removed. Figure 5.14 shows the structure of the QD map generator.

Previous work has explored generating MAPF maps with Quality Diversity algorithms to generate maps suitable for high-throughput online MAPF [180]. Zhang et al. trained a surrogate model DSAGE [14] that could help repair instances to meet constraints (e.g., number of shelves and connectivity) and predict

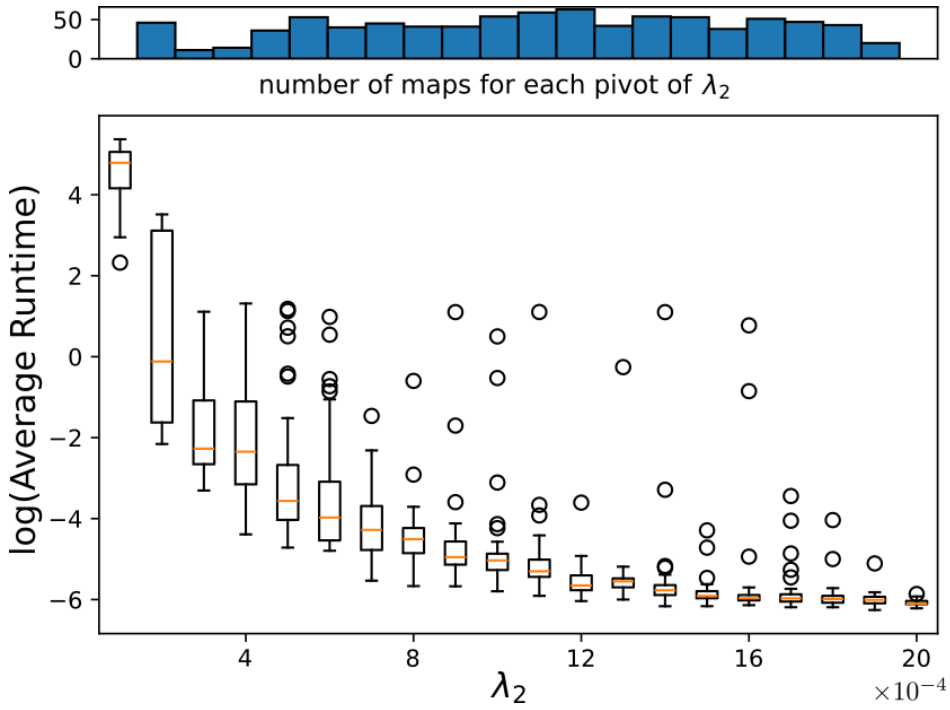


Figure 5.15: Boxplot for λ_2 and the average runtime of maps created by the QD generator.

the throughput of an instance. Our problem requires less sophisticated repair since we have no hard constraint on the number of obstacles in an instance. Additionally, our objective function is relatively easy to compute, requiring only λ_2 for a generated map, and does not require any additional MAPF simulations. Our map generator is designed to create instances with a wide range of connectivity to use in the evaluation and benchmarking of MAPF algorithms. A more detailed introduction to related research is in Section 3.4.4.

5.7.2 Additional Tests of QD Map Generator

We present additional tests on the runtime using CBSH2-RTC for maps generated by our QD map generator. We have generated 851 maps with a step size of 10^{-4} for λ_2. For the generated instances, the $r = 2.25 \times 10^{-2}$ and number of agents ranges in $[19, 23]$. Different from the cellular automata and diffusion-limited aggregation map generators, which lack precise control over map connectivity, the QD map generator can generate maps with a well-distributed range of λ_2. This nice feature makes it a great choice for creating benchmark dataset that covers a wider spectrum of map connectivity.

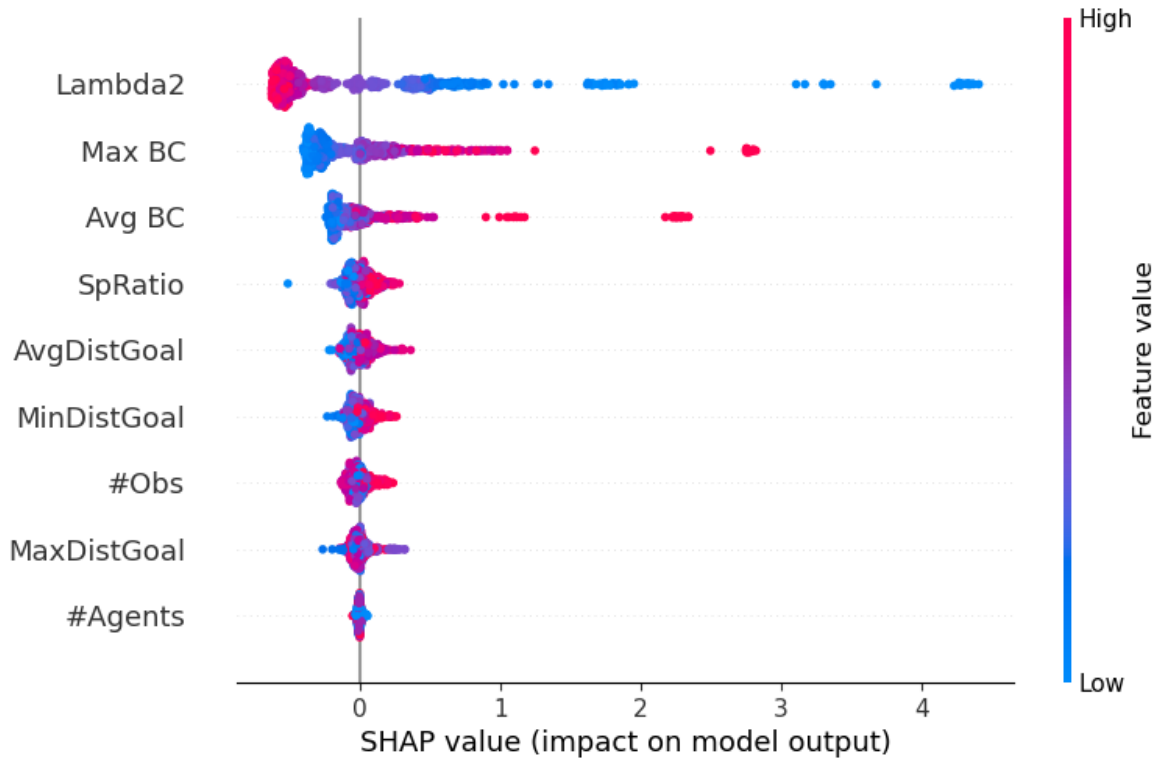


Figure 5.16: SHAP value of different features on the impact of the XGBoost regression model output when predicting the logarithm of average runtime for different maps.

The results are shown in Figure 5.15. Despite the fact that there are still many outliers in Figure 5.15, the relationship between λ_2 and empirical hardness still holds. This indicates that hard instances tend to happen more frequently around small λ_2, while large λ_2 generally result in easier instances. Moreover, the top part of Figure 5.15 shows the number of maps for each pivot of λ_2. It shows the capability of the QD generator to create a diverse benchmark dataset with maps well distributed for different connectivity.

5.7.3 Feature Importance of λ_2

The λ_2 value can be used as a major feature to predict the average empirical hardness of different maps. We combined the instance features proposed by [142, 33] and trained an XGBoost [18] regression model to predict the logarithm of average runtime for maps with varying connectivity. This model was trained on over 3200 maps generated by our QD generator. The λ_2 values range from 9×10^{-5} to 251×10^{-5}.

To better understand the impact of different instance features on the model's output, we performed a SHapley Additive exPlanation (SHAP) analysis [98]. The features analyzed include λ_2, maximum and average betweenness centrality, SpRatio (agent-to-freespace ratio), minimum/maximum/average distance to goal for all agents, the number of agents, and the number of obstacles. The results are shown in Figure 5.16. Each data point in Figure 5.16 represents a different map. The color of each data point represents the SHAP value of different features. For different features, the higher the SHAP value is, the more impact it has on the model output (logarithm of average runtime). The importance of different features is ranked from top to bottom. The results clearly show that λ_2 has a major impact on the average runtime of different maps with varying connectivity. When λ_2 is smaller, it tends to make the average runtime higher and indicates more hard instances. Maximum and average betweenness centrality also play a significant role, as they are also closely related to the map connectivity.

5.8 Summary

In this chapter, we first showed both theoretically and experimentally that the empirical hardness of MAPF is closely correlated with map connectivity. We showed the second smallest eigenvalue (λ_2) of the normalized Laplacian matrix is an effective approximation of map connectivity and can be used to roughly compare the average empirical hardness of different maps. Although λ_2 does not exhibit a strict monotonic correlation with empirical hardness, it still shows notable effectiveness, especially for very challenging instances associated with small λ_2. Given its simplicity and ease of comparison across maps, we believe λ_2 is an effective starting point for future research on MAPF empirical hardness.

Additionally, we demonstrated how to use the quality diversity method to generate maps with a specific range of connectivity (λ_2 values). The QD generator enables the creation of more diverse benchmark datasets by generating maps with varying levels of connectivity. This helps researchers to systematically evaluate MAPF algorithms across different map types, from well-connected environments to more

challenging, bottleneck-heavy layouts. It also provides a reliable method for creating challenging MAPF instances, as maps with smaller λ_2 values typically have poor connectivity and produce more hard instances. These challenging instances can help identify the limitations of existing algorithms and inspire the development of new heuristics and algorithms.

Chapter 6

Conclusions and Future Work

In this dissertation, we have made several foundational contributions to understanding the empirical hardness of the Multi-Agent Path Finding (MAPF) problem. We formalized the concept of empirical hardness in MAPF and provided the first comprehensive taxonomy of this field. This helps other researchers understand the significance of studying the empirical hardness of MAPF. Next, we demonstrated how to use algorithm selection models to effectively predict the MAPF empirical hardness without solving the instances. Then, we showed both theoretically and empirically that map connectivity plays a critical role in determining the empirical hardness of MAPF instances. We demonstrated that the second smallest eigenvalue of the normalized Laplacian matrix is an effective measure of map connectivity. Lastly, we present techniques to generate challenging MAPF instances. We introduced our MAPF instance generator which is capable of creating maps with controllable connectivity. This also provides a valuable tool for creating MAPF benchmark datasets with diverse connectivity and varying hardness levels.

This dissertation introduces and establishes a new research direction in MAPF focused on empirical hardness. The techniques and insights presented will ultimately improve real-world multi-agent systems. It also enables many future research directions such as:

- **Better Encoding Techniques For MAPF Algorithm Selection:** In Chapter 4, we demonstrated that using single-agent shortest path encoding significantly improved algorithm selection performance for MAPF instances. However, there are many potential improvements and extensions to explore. The basic single-agent shortest path encoding may struggle with instances containing a large number of agents, as the entire map can become saturated with paths. One possible enhancement is to encode each map cell based on how often it appears in the single-agent shortest path. Another promising direction is to explore new encoding methods beyond shortest paths, such as encoding cells based on how frequently conflicts occur when using specific algorithms. Additionally, as shown in [19], combining multiple encodings as different input layers can provide richer information and further enhance prediction performance.

  For models that do not rely on CNN-based approaches, it is essential to consider how to extract map topology as numerical parameters, such as using λ_2 to represent connectivity. Another promising direction, suggested by [136], is to develop models that combine hand-crafted features with graph-embedding features. This hybrid approach combines both traditional and graph-based methods to better capture the underlying structure of MAPF instances.

- **Tunable Empirical Hardness Instance Generators:** In Section 5.7, we presented a map generator capable of generating maps with desired connectivity values. Future MAPF instance generators should be capable of creating instances that span a broad spectrum of empirical hardness. This will lead to more diverse benchmark datasets and allow for in-depth analysis and fair comparison of current MAPF algorithms. In a recent study, Qian et al. [125] presented a MAPF instance generator that can generate maps with varying hardness. This work is a great example of how such instance generators should be designed. A promising follow-up would be developing instance generators that can directly control the empirical hardness of *individual* MAPF instances rather than just the average hardness. Existing MAPF instance generators often make general assumptions about the

distribution of start and goal location, such as uniform random sampling. Future research could focus on directly controlling empirical hardness by strategically adjusting the placements of agents' start and goal locations. For example, a reinforcement learning model could be used to reward the start and goal locations that lead to more conflicts, thus making instances harder.

- **Phase Transition of MAPF:** Phase transition is fundamental to empirical hardness research and essential to help understand the instance hardness. In MAPF, studying phase transitions helps identify the specific points where problem instances shift from easily solvable to exceptionally hard. Understanding the MAPF phase transition has broad impacts on various research domains. It will guide the development of more powerful and efficient instance generators as we know where the really hard instances are. Studying the features and structures of the hard instances near the phase transition boundary can also inspire the development of new heuristics and algorithms.

  In Chapter 3, we demonstrated that studying MAPF phase transition is significantly challenging due to their complex instance features. Future research could initially focus on the sharp increase in hardness as a preliminary step (such as [33, 126]) before exploring the actual phase transition phenomenon. It would also be interesting to explore the correlation between the percentage of backbone and backdoors in MAPF instances and phase transitions.

Based on the feedback from my committee during my defense, there are several additional promising research directions:

- **Generating Instances with Smooth Hardness Transitions:** Given two MAPFAST instances with varying hardness, an interesting research direction would be to develop methods for generating new instances that exhibit smooth and continuous empirical hardness transitions within the bounds defined by the two original instances. This could be achieved by using reinforcement learning to train an instance generator to gradually increase the empirical hardness with a fixed step size.

- **Generating Maps with Controlled Style and Connectivity:** In Chapter 5, we demonstrated how to use the Quality Diversity method to generate maps with controlled connectivity. A possible follow-up direction involves generating maps with varying connectivity while maintaining the map style. For instance, one could generate well-connected maze maps or poorly-connected city maps. The Quality Diversity method is particularly suited for this task, as it allows setting connectivity as an objective function while preserving diverse map styles, such as by leveraging KL-divergence of map topology [125].

Bibliography

[1] Dimitris Achlioptas and Ehud Friedgut. “A sharp threshold for k-colorability”. In: *Random Structures & Algorithms* 14.1 (1999), pp. 63–70.

[2] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. “Generating satisfiable problem instances”. In: *AAAI/IAAI* 2000 (2000), pp. 256–261.

[3] Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

[4] Jean-Marc Alkazzi, Anthony Rizk, Michel Salomon, and Abdallah Makhoul. “Mapfaster: A faster and simpler take on multi-agent path finding algorithm selection”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 10088–10093.

[5] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. “Unsatisfiability-based optimization in clasp”. In: *Technical Communications of the 28th International Conference on Logic Programming (ICLP’12)(2012)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2012.

[6] Olivier Bailleux and Yacine Boufkhad. “Efficient CNF encoding of boolean cardinality constraints”. In: *International conference on principles and practice of constraint programming*. Springer. 2003, pp. 108–122.

[7] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. “Breaking the limits of message passing graph neural networks”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 599–608.

[8] Adrian Balint and Uwe Schöning. “Choosing probability distributions for stochastic local search and the role of make versus break”. In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2012, pp. 16–29.

[9] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. “Intractability of Time-Optimal Multirobot Path Planning on 2d Grid Graphs with Holes”. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 1941–1947.

[10] Max Barer, Guni Sharon, Roni Stern, and Ariel 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. 1. 2014, pp. 19–27.

[11] Earl R Barnes. “An algorithm for partitioning the nodes of a graph”. In: *SIAM Journal on Algebraic Discrete Methods* 3.4 (1982), pp. 541–550.

[12] Marc Barthelemy. “Betweenness centrality in large complex networks”. In: *The European physical journal B* 38.2 (2004), pp. 163–168.

[13] Lukas Bernreiter, Shehryar Khattak, Lionel Ott, Roland Siegwart, Marco Hutter, and Cesar Cadena. “Collaborative robot mapping using spectral graph analysis”. In: *2022 international conference on robotics and automation (ICRA)*. IEEE. 2022, pp. 3662–3668.

[14] Varun Bhatt, Bryon Tjanaka, Matthew Fontaine, and Stefanos Nikolaidis. “Deep surrogate assisted generation of environments”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37762–37777.

[15] Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. “Solving constraint satisfaction problems with SAT modulo theories”. In: *Constraints* 17 (2012), pp. 273–303.

[16] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and Eyal Shimony. “ICBS: The Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding”. In: *Eighth Annual Symposium on Combinatorial Search*. Citeseer. 2015.

[17] Peter C Cheeseman, Bob Kanefsky, William M Taylor, et al. “Where the really hard problems are.” In: *Ijcai*. Vol. 91. 1991, pp. 331–337.

[18] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proc. of the 22nd ACM SIGKDD Intl Conf on Knowledge Discovery and Data Mining*. 2016, pp. 785–794.

[19] Weizhe Chen, Zhihan Wang, Jiaoyang Li, Sven Koenig, and Bistra Dilkina. “No Panacea in Planning: Algorithm Selection for Suboptimal Multi-Agent Path Finding”. In: *arXiv preprint arXiv:2404.03554* (2024).

[20] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. “Integrated task assignment and path planning for capacitated multi-agent pickup and delivery”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5816–5823.

[21] Fan RK Chung. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

[22] Jane K Cullum and Ralph A Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations: Vol. I: Theory*. SIAM, 2002.

[23] Martin Davis and Hilary Putnam. “A computing procedure for quantification theory”. In: *Journal of the ACM (JACM)* 7.3 (1960), pp. 201–215.

[24] Jaydev P Desai. “A graph theoretic approach for modeling mobile robot team formations”. In: *Journal of Robotic Systems* 19.11 (2002), pp. 511–525.

[25] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*. Vol. 5. Springer Science & Business Media, 2006.

[26] Jacques Desrosiers and Marco E Lübbecke. "Branch-price-and-cut algorithms". In: *Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Chichester* (2011), pp. 109–131.

[27] Bistra Dilkina, Carla P Gomes, Yuri Malitsky, Ashish Sabharwal, and Meinolf Sellmann. "Backdoors to combinatorial optimization: Feasibility and optimality". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings 6*. Springer. 2009, pp. 56–70.

[28] Chris Ding, Tao Li, and Michael I Jordan. "Nonnegative matrix factorization for combinatorial optimization: Spectral clustering, graph matching, and clique finding". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 183–192.

[29] William E Donath and Alan J Hoffman. "Lower bounds for the partitioning of graphs". In: *IBM Journal of Research and Development* 17.5 (1973), pp. 420–425.

[30] Daniel S Drew. "Multi-agent systems for search and rescue applications". In: *Current Robotics Reports* 2 (2021), pp. 189–200.

[31] Olivier Dubois and Gilles Dequen. "A backbone-search heuristic for efficient solving of hard 3-SAT formulae". In: *IJCAI*. Vol. 1. 2001, pp. 248–253.

[32] Esra Erdem, Doga Kisa, Umut Oztok, and Peter Schüller. "A general formal framework for pathfinding problems with multiple agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013, pp. 290–296.

[33] Eric Ewing, Jingyao Ren, Dhvani Kansara, Vikraman Sathiyanarayanan, and Nora Ayanian. "Betweenness centrality in multi-agent path finding". In: *International Conference on Autonomous Agents and Multiagent Systems*. 2022.

[34] Ariel Felner, Meir Goldenberg, Guni Sharon, Roni Stern, Tal Beja, Nathan Sturtevant, Jonathan Schaeffer, and Robert Holte. "Partial-expansion A* with selective node generation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 1. 2012, pp. 471–477.

[35] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. "Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding". In: *Proc. of the Intl Conf on Automated Planning and Scheduling*. Vol. 28. 1. 2018, pp. 83–87.

[36] Matteo Fischetti and Michele Monaci. "Backdoor branching". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2011, pp. 183–191.

[37] Matthew Fontaine and Stefanos Nikolaidis. "Covariance matrix adaptation map-annealing". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2023, pp. 456–465.

[38] Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. “Covariance matrix adaptation for the rapid illumination of behavior space”. In: *Proceedings of the 2020 genetic and evolutionary computation conference*. 2020, pp. 94–102.

[39] Joshua J Forman, Paul A Clemons, Stuart L Schreiber, and Stephen J Haggarty. “SpectralNET–an application for spectral graph analysis and visualization”. In: *BMC bioinformatics* 6 (2005), pp. 1–13.

[40] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. “SAT competition 2020”. In: *Artificial Intelligence* 301 (2021), p. 103572.

[41] Graeme Gange, Daniel Harabor, and Peter J Stuckey. “Lazy CBS: implicit conflict-based search using lazy clause generation”. In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 29. 2019, pp. 155–162.

[42] Ian P Gent, Ewan MacIntyre, Patrick Prosser, Toby Walsh, et al. “The constrainedness of search”. In: *AAAI/IAAI, Vol. 1*. 1996, pp. 246–252.

[43] Ian P Gent and Toby Walsh. “The TSP phase transition”. In: *Artificial Intelligence* 88.1-2 (1996), pp. 349–358.

[44] Chris Godsil and Gordon F Royle. *Algebraic graph theory*. Vol. 207. Springer Science & Business Media, 2001.

[45] K-I Goh, Eulsik Oh, Byungnam Kahng, and Doochul Kim. “Betweenness centrality correlation in social networks”. In: *Physical Review E* 67.1 (2003), p. 017101.

[46] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. “Enhanced Partial Expansion A”. In: *Journal of Artificial Intelligence Research* 50 (2014), pp. 141–187.

[47] Meir Goldenberg, Ariel Felner, Nathan Sturtevant, Robert C Holte, and Jonathan Schaeffer. “Optimal-generation variants of EPEA”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 4. 1. 2013, pp. 89–97.

[48] Ofir Gordon, Yuval Filmus, and Oren Salzman. “Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 12. 1. 2021, pp. 64–72.

[49] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. “ManySAT: a parallel SAT solver”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 6.4 (2010), pp. 245–262.

[50] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[51] Bruce Hendrickson and Robert Leland. “An improved spectral graph partitioning algorithm for mapping parallel computations”. In: *SIAM Journal on Scientific Computing* 16.2 (1995), pp. 452–469.

[52] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. “Procedural content generation for games: A survey”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.1 (2013), pp. 1–22.

[53] Florence Ho, Rúben Geraldes, Artur Gonçalves, Bastien Rigault, Benjamin Sportich, Daisuke Kubo, Marc Cavazza, and Helmut Prendinger. “Decentralized multi-agent path finding for UAV traffic management”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.2 (2020), pp. 997–1008.

[54] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W Durham, and Nora Ayanian. “Persistent and robust execution of MAPF schedules in warehouses”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1125–1131.

[55] Wolfgang Hönig, James A Preiss, TK Satish Kumar, Gaurav S Sukhatme, and Nora Ayanian. “Trajectory planning for quadrotor swarms”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 856–869.

[56] Sunan Huang, Rodney Swee Huat Teo, and Kok Kiong Tan. “Collision avoidance of multi unmanned aerial vehicles: A review”. In: *Annual Reviews in Control* 48 (2019), pp. 147–164.

[57] Frank Hutter, Domagoj Babic, Holger H Hoos, and Alan J Hu. “Boosting verification by automatic tuning of decision procedures”. In: *Formal Methods in Computer Aided Design (FMCAD’07)*. IEEE. 2007, pp. 27–34.

[58] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. “Automated configuration of mixed integer programming solvers”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7*. Springer. 2010, pp. 186–202.

[59] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. “Sequential model-based optimization for general algorithm configuration”. In: *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer. 2011, pp. 507–523.

[60] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. “ParamILS: an automatic algorithm configuration framework”. In: *Journal of artificial intelligence research* 36 (2009), pp. 267–306.

[61] Frank Hutter, Holger H Hoos, and Thomas Stützle. “Automatic algorithm configuration based on local search”. In: *Aaai*. Vol. 7. 2007, pp. 1152–1157.

[62] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. “Algorithm runtime prediction: Methods & evaluation”. In: *Artificial Intelligence* 206 (2014), pp. 79–111.

[63] Yannet Interian. “Backdoor sets for random 3-SAT”. In: (2003).

[64] James S Jennings, Greg Whelan, and William F Evans. “Cooperative search and rescue with a team of mobile robots”. In: *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR’97*. IEEE. 1997, pp. 193–200.

[65] He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li. “Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 17. 2024, pp. 234–242.

[66] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. “Cellular automata for real-time generation of infinite cave levels”. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 2010, pp. 1–4.

[67] Omri Kaduri, Eli Boyarski, and Roni Stern. “Algorithm Selection for Optimal Multi-Agent Pathfinding”. In: *Proc. of the Intl Conf on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 161–165.

[68] Omri Kaduri, Eli Boyarski, and Roni Stern. “Experimental evaluation of classical multi agent path finding algorithms”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 12. 1. 2021, pp. 126–130.

[69] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H Hoos, and Heike Trautmann. “Leveraging TSP Solver Complementarity Through Machine Learning”. In: *Evolutionary Computation* 26.4 (2018), pp. 597–620.

[70] Ashiqur R KhudaBukhsh, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. “SATenstein: Automatically building local search SAT solvers from components”. In: *Artificial Intelligence* 232 (2016), pp. 20–42.

[71] Philip Kilby, John Slaney, Sylvie Thiébaux, Toby Walsh, et al. “Backbones and backdoors in satisfiability”. In: *AAAI*. Vol. 5. 2005, pp. 1368–1373.

[72] Diederik P Kingma. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[73] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[74] Scott Kirkpatrick and Gérard Toulouse. “Configuration space analysis of traveling salesman problems”. In: *Journal de physique* 46.8 (1985), pp. 1277–1292.

[75] Yehuda Koren. “On spectral graph drawing”. In: *International Computing and Combinatorics Conference*. Springer. 2003, pp. 496–508.

[76] Lars Kotthoff. “Algorithm Selection for Combinatorial Search Problems: A Survey”. In: *Data Mining and Constraint Programming*. Springer, 2016, pp. 149–190.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[78] Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J Stuckey. “Branch-and-cut-and-price for multi-agent path finding”. In: *Computers & Operations Research* 144 (2022), p. 105809.

[79] Edward Lam, Pierre Le Bodic, Daniel Damir Harabor, and Peter J Stuckey. "Branch-and-Cut-and-Price for Multi-Agent Pathfinding". In: *Proc. of the Intl Joint Conf on Artificial Intelligence*. 2019, pp. 1289–1296.

[80] Cornelius Lanczos. "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators". In: (1950).

[81] League of Robot Runners. *League of Robot Runners*. `https://www.leagueofrobotrunners.org/`. Accessed: 2024-09-14. 2024.

[82] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[83] Wonjong Lee, Joonyeol Sim, and Changjoo Nam. "Puzzle Heuristics: Efficient Lifelong Multi-Agent Pathfinding Algorithm for Large-scale Challenging Environments". In: *The Journal of Korea Robotics Society* 19.3 (2024), pp. 281–286.

[84] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. "Boosting as a metaphor for algorithm design". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2003, pp. 899–903.

[85] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. "Empirical hardness models for combinatorial auctions". In: *Combinatorial auctions* (2006), pp. 479–504.

[86] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. "Empirical hardness models: Methodology and a case study on combinatorial auctions". In: *Journal of the ACM (JACM)* 56.4 (2009), pp. 1–52.

[87] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. "Learning the empirical hardness of optimization problems: The case of combinatorial auctions". In: *Principles and Practice of Constraint Programming-CP 2002: 8th International Conference, CP 2002 Ithaca, NY, USA, September 9–13, 2002 Proceedings 8*. Springer. 2002, pp. 556–572.

[88] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven 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.

[89] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig. "Scalable rail planning and replanning: Winning the 2020 flatland challenge". In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 31. 2021, pp. 477–485.

[90] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. "Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search." In: *Proc. of the Intl Joint Conf on Artificial Intelligence*. 2019, pp. 442–449.

[91] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig. "New techniques for pairwise symmetry breaking in multi-agent path finding". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 193–201.

[92] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Hang Ma, Graeme Gange, and Sven Koenig. "Pairwise symmetry reasoning for multi-agent path finding search". In: *Artificial Intelligence* 301 (2021), p. 103574.

[93] Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Hang Ma, and Sven Koenig. "Symmetry-breaking constraints for grid-based multi-agent path finding". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 6087–6095.

[94] Jiaoyang Li, Kexuan Sun, Hang Ma, Ariel Felner, TK Kumar, and Sven Koenig. "Moving agents in formation in congested environments". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 11. 1. 2020, pp. 131–132.

[95] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. "Lifelong multi-agent path finding in large-scale warehouses". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13. 2021, pp. 11272–11281.

[96] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. "Task and path planning for multi-agent pickup and delivery". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019.

[97] Marco E Lübbecke and Jacques Desrosiers. "Selected topics in column generation". In: *Operations research* 53.6 (2005), pp. 1007–1023.

[98] Scott Lundberg. "A unified approach to interpreting model predictions". In: *arXiv preprint arXiv:1705.07874* (2017).

[99] Russell Lyons and Shayan Oveis Gharan. "Sharp bounds on random walk eigenvalues via spectral embedding". In: *International Mathematics Research Notices* 2018.24 (2018), pp. 7555–7605.

[100] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. "Lifelong path planning with kinematic constraints for multi-agent pickup and delivery". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7651–7658.

[101] Hang Ma, Jingxing Yang, Liron Cohen, TK Kumar, and Sven 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. 1. 2017, pp. 270–272.

[102] Ángel Madridano, Abdulla Al-Kaff, David Martín, and Arturo De La Escalera. "Trajectory planning for multi-robot systems: Methods and applications". In: *Expert Systems with Applications* 173 (2021), p. 114660.

[103] Sebastian Mai and Sanaz Mostaghim. "Modeling pathfinding for swarm robotics". In: *International Conference on Swarm Intelligence*. Springer. 2020, pp. 190–202.

[104] Joao Marques-Silva and Inês Lynce. “Towards robust CNF encodings of cardinality constraints”. In: *Principles and Practice of Constraint Programming–CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings 13*. Springer. 2007, pp. 483–497.

[105] Adam McLaughlin and David A Bader. “Scalable and high performance betweenness centrality on the GPU”. In: *SC’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2014, pp. 572–583.

[106] Frank McSherry. “Spectral partitioning of random graphs”. In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE. 2001, pp. 529–537.

[107] David Mitchell, Bart Selman, Hector Levesque, et al. “Hard and easy distributions of SAT problems”. In: *Aaai*. Vol. 92. 1992, pp. 459–465.

[108] Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, et al. “Flatland-rl: Multi-agent reinforcement learning on trains”. In: *arXiv preprint arXiv:2012.05893* (2020).

[109] Bojan Mohar and Svatopluk Poljak. “Eigenvalues in combinatorial optimization”. In: *Combinatorial and graph-theoretical problems in linear algebra*. Springer, 1993, pp. 107–151.

[110] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. “Determining computational complexity from characteristic ‘phase transitions’”. In: *Nature* 400.6740 (1999), pp. 133–137.

[111] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. “Growing neural cellular automata”. In: *Distill* 5.2 (2020), e23.

[112] Robert Morris, Corina S Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. “Planning, scheduling and monitoring for airport surface operations”. In: *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[113] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. In: *arXiv preprint arXiv:1504.04909* (2015).

[114] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. “Graph2Vec: Learning Distributed Representations of Graphs”. In: *arXiv preprint arXiv:1707.05005* (2017).

[115] Maxim Naumov and Timothy Moon. “Parallel spectral graph partitioning”. In: *NVIDIA, Santa Clara, CA, USA, Tech. Rep., NVR-2016-001* (2016).

[116] Mark EJ Newman. “Spectral methods for community detection and graph partitioning”. In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 88.4 (2013), p. 042822.

[117] Eugene Nudelman, Kevin Leyton-Brown, Alex Devkar, Yoav Shoham, and Holger Hoos. “Satzilla: An algorithm portfolio for SAT”. In: *Solver description, SAT competition* 2004 (2004).

[118] Eugene Nudelman, Kevin Leyton-Brown, Holger H Hoos, Alex Devkar, and Yoav Shoham. "Understanding random SAT: Beyond the clauses-to-variables ratio". In: *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*. Springer. 2004, pp. 438–452.

[119] Olga Ohrimenko, Peter J Stuckey, and Michael Codish. "Propagation via lazy clause generation". In: *Constraints* 14 (2009), pp. 357–391.

[120] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. "Priority inheritance with backtracking for iterative multi-agent path finding". In: *Artificial Intelligence* 310 (2022), p. 103752.

[121] Edwin Olson, Matthew R Walter, Seth J Teller, and John J Leonard. "Single-Cluster Spectral Graph Partitioning for Robotics Applications." In: *Robotics: Science and Systems*. 2005, pp. 265–272.

[122] Andrew J Parkes. "Clustering at the phase transition". In: *AAAI/IAAI*. Citeseer. 1997, pp. 340–345.

[123] Beresford N Parlett. *The symmetric eigenvalue problem*. SIAM, 1998.

[124] Patrick Prosser. "Binary constraint satisfaction problems: Some are harder than others". In: *Proceedings of the 11th European Conference on Artificial Intelligence*. 1994, pp. 95–99.

[125] Cheng Qian, Yulun Zhang, and Jiaoyang Li. "A Quality Diversity Approach to Automatically Generate Multi-Agent Path Finding Benchmark Maps". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 17. 2024, pp. 279–280.

[126] Jingyao Ren, Eric Ewing, TK Satish Kumar, Sven Koenig, and Nora Ayanian. "Map Connectivity and Empirical Hardness of Grid-based Multi-Agent Pathfinding Problem". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 34. 2024, pp. 484–488.

[127] Jingyao Ren, Vikraman Sathiyanarayanan, Eric Ewing, Baskin Senbaslar, and Nora Ayanian. "MAPFAST: A deep algorithm selector for multi agent path finding using shortest path embeddings". In: *arXiv preprint arXiv:2102.12461* (2021).

[128] John R Rice. "The Algorithm Selection Problem". In: *Advances in Computers*. Vol. 15. Elsevier, 1976, pp. 65–118.

[129] Yongshao Ruan, Henry A Kautz, and Eric Horvitz. "The backdoor key: A path to understanding problem hardness". In: *AAAI*. Vol. 4. 2004, pp. 118–123.

[130] Malcolm Ryan. "Constraint-based multi-robot path planning". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 922–928.

[131] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

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

[133] Johannes Schneider, Christine Froschhammer, Ingo Morgenstern, Thomas Husslein, and Johannes Maria Singer. “Searching for backbones—an efficient parallel algorithm for the traveling salesman problem”. In: *Computer Physics Communications* 96.2-3 (1996), pp. 173–188.

[134] Michael Schuresko and Jorge Cortés. “Distributed motion constraints for algebraic connectivity of robotic networks”. In: *Journal of Intelligent and Robotic Systems* 56 (2009), pp. 99–126.

[135] Bart Selman, David G Mitchell, and Hector J Levesque. “Generating hard satisfiability problems”. In: *Artificial intelligence* 81.1-2 (1996), pp. 17–29.

[136] Carmel Shabalin, Omri Kaduri, and Roni Stern. “Algorithm Selection for Optimal Multi-Agent Path Finding via Graph Embedding”. In: *arXiv preprint arXiv:2406.10827* (2024).

[137] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. “Conflict-Based Search for Optimal Multi-Agent Pathfinding”. In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[138] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. “The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding”. In: *Artificial Intelligence* 195 (2013), pp. 470–495.

[139] Bojie Shen, Zhe Chen, Muhammad Aamir Cheema, Daniel D Harabor, and Peter J Stuckey. “Tracking progress in multi-agent path finding”. In: *arXiv preprint arXiv:2305.08446* (2023).

[140] Bojie Shen, Zhe Chen, Jiaoyang Li, Muhammad Aamir Cheema, Daniel D Harabor, and Peter J Stuckey. “Beyond Pairwise Reasoning in Multi-Agent Path Finding”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2023, pp. 338–347.

[141] Hua-Wei Shen and Xue-Qi Cheng. “Spectral methods for the detection of network community structure: a comparative analysis”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2010.10 (2010), P10020.

[142] Devon Sigurdson, Vadim Bulitko, Sven Koenig, Carlos Hernandez, and William Yeoh. “Automatic Algorithm Selection in Multi-Agent Pathfinding”. In: *arXiv preprint arXiv:1906.03992* (2019).

[143] Devon Sigurdson, Vadim Bulitko, William Yeoh, Carlos Hernández, and Sven Koenig. “Multi-agent pathfinding with real-time heuristic search”. In: *2018 IEEE conference on computational intelligence and games (CIG)*. IEEE. 2018, pp. 1–8.

[144] Laurent Simon, Daniel Le Berre, and Edward A Hirsch. “The SAT2002 competition”. In: *Annals of Mathematics and Artificial Intelligence* 43 (2005), pp. 307–342.

[145] John Slaney, Toby Walsh, et al. “Backbones in optimization and approximation”. In: *IJCAI*. 2001, pp. 254–259.

[146] Barbara M Smith. “The phase transition and the mushy region in constraint satisfaction problems”. In: *Proceedings of the 11th European Conference on Artificial Intelligence*. 1994, pp. 100–104.

[147] Kate Smith-Miles, Jano Van Hemert, and Xin Yu Lim. "Understanding TSP difficulty by learning from evolved instances". In: *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers 4*. Springer. 2010, pp. 266–280.

[148] Kate A Smith-Miles. "Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection". In: *ACM Computing Surveys (CSUR)* 41.1 (2009), pp. 1–25.

[149] Ricard V Solé. *Phase transitions*. Vol. 3. Princeton University Press, 2011.

[150] Daniel Spielman. "Spectral and algebraic graph theory". In: *Yale lecture notes, draft of December* 4 (2019), p. 47.

[151] Daniel Spielman. "Spectral graph theory". In: *Combinatorial scientific computing* 18 (2012), p. 18.

[152] Arvind Srinivasan, Timothy Ham, Sharad Malik, and Robert K Brayton. "Algorithms for discrete function manipulation". In: *1990 IEEE international conference on computer-aided design*. IEEE Computer Society. 1990, pp. 92–93.

[153] Trevor Standley. "Finding optimal solutions to cooperative pathfinding problems". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 24. 1. 2010, pp. 173–178.

[154] Roni Stern. "Multi-agent path finding–an overview". In: *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures* (2019), pp. 96–115.

[155] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". In: *Symposium on Combinatorial Search (SoCS)* (2019), pp. 151–158.

[156] Pavel Surynek. "Towards optimal cooperative path planning in hard setups through satisfiability solving". In: *Pacific Rim international conference on artificial intelligence*. Springer. 2012, pp. 564–576.

[157] Pavel Surynek. "Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 10. 1. 2019, pp. 202–203.

[158] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. "Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective". In: *Proc. of the European Conf on Artificial Intelligence*. 2016, pp. 810–818.

[159] Jiri Svancara and Roman Bartak. "Combining Strengths of Optimal Multi-Agent Path Finding Algorithms". In: *Proc. of the Intl Conf on Agents and Artificial Intelligence - Vol. 1*. 2019, pp. 226–231.

[160] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions". In: *Proc. of the IEEE Conf on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[161] Allen Van Gelder, Daniel Le Berre, Armin Biere, Oliver Kullmann, and Laurent Simon. "Purse-Based Scoring for Comparison of Exponential-time Programs". In: *Eighth Intl Conf on Theory and Applications of Satisfiability Testing* (2005).

[162] Xiyuan Wang and Muhan Zhang. "How powerful are spectral graph neural networks". In: *International conference on machine learning*. PMLR. 2022, pp. 23341–23362.

[163] Ryan Williams, Carla Gomes, and Bart Selman. "On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search". In: *structure* 23.4 (2003).

[164] Ryan Williams, Carla P Gomes, and Bart Selman. "Backdoors to typical case complexity". In: *IJCAI*. Vol. 3. 2003, pp. 1173–1178.

[165] Thomas A Witten and Leonard M Sander. "Diffusion-limited aggregation". In: *Physical review B* 27.9 (1983), p. 5686.

[166] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), pp. 9–9.

[167] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2012, pp. 228–241.

[168] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming". In: *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*. 2011, pp. 16–30.

[169] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "SATzilla: Portfolio-Based Algorithm Selection for SAT". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 565–606.

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

[171] Jingjin Yu. "Intractability of optimal multirobot path planning on planar graphs". In: *IEEE Robotics and Automation Letters* 1.1 (2015), pp. 33–40.

[172] Jingjin Yu and Steven M LaValle. "Planning optimal paths for multiple robots on graphs". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3612–3617.

[173] Jingjin Yu and Steven M LaValle. "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs". In: *Twenty-Seventh AAAI Conf on Artificial Intelligence*. 2013, pp. 1444–1449.

[174] Michael M Zavlanos, Magnus B Egerstedt, and George J Pappas. "Graph-theoretic connectivity control of mobile robot networks". In: *Proceedings of the IEEE* 99.9 (2011), pp. 1525–1540.

[175] Lenka Zdeborová and Florent Krząkała. "Phase transitions in the coloring of random graphs". In: *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 76.3 (2007), p. 031131.

[176] Han Zhang, Jiaoyang Li, Pavel Surynek, TK Satish Kumar, and Sven Koenig. “Multi-agent path finding with mutex propagation”. In: *Artificial Intelligence* 311 (2022), p. 103766.

[177] Weixiong Zhang. *State-space search: Algorithms, complexity, extensions, and applications*. Springer Science & Business Media, 1999.

[178] Yulun Zhang, Matthew Fontaine, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. “Arbitrarily scalable environment generators via neural cellular automata”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[179] Yulun Zhang, Matthew C Fontaine, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. “Multi-robot coordination and layout design for automated warehousing”. In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 17. 2024, pp. 305–306.

[180] Yulun Zhang, Matthew C. Fontaine, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. “Multi-Robot Coordination and Layout Design for Automated Warehousing”. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. Ed. by Edith Elkind. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2023, pp. 5503–5511. DOI: 10.24963/ijcai.2023/611.

Appendix

A Supplementary Figures for Chapter 4

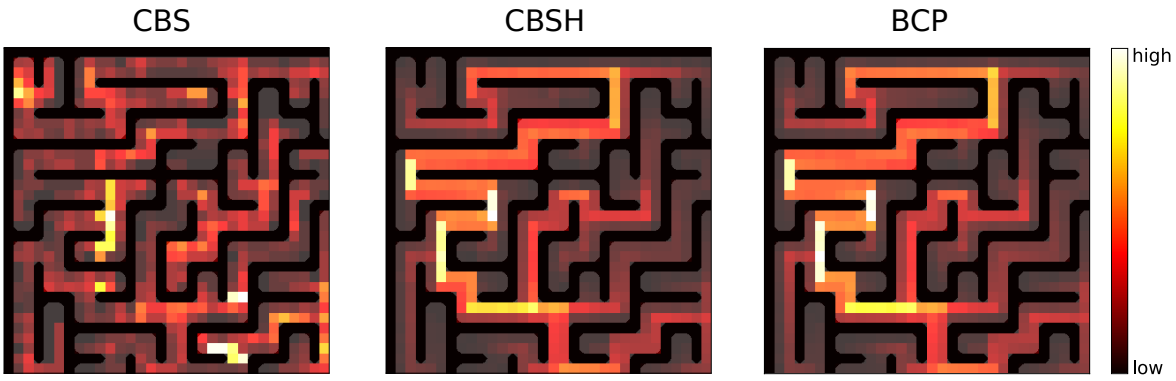


Figure A.1: Heat maps of the single-agent shortest paths for different algorithms for maze-32-32-2.

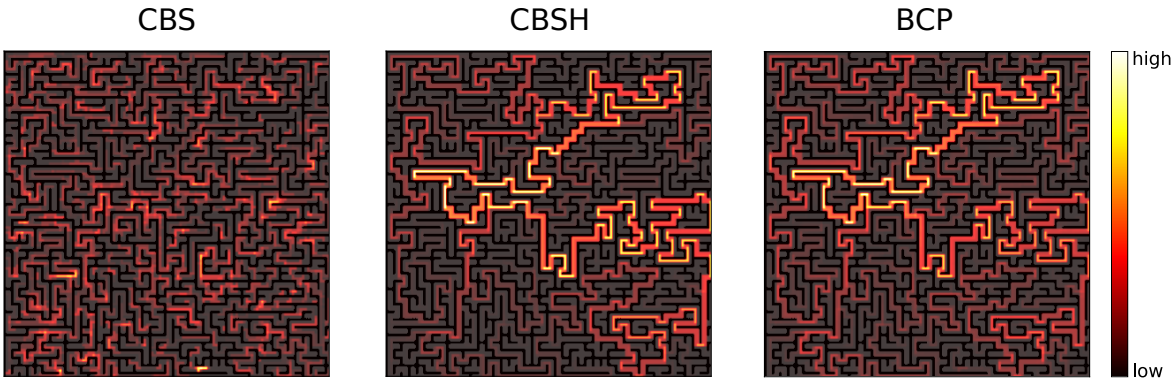


Figure A.2: Heat maps of the single-agent shortest paths for different algorithms for maze-128-128-2.

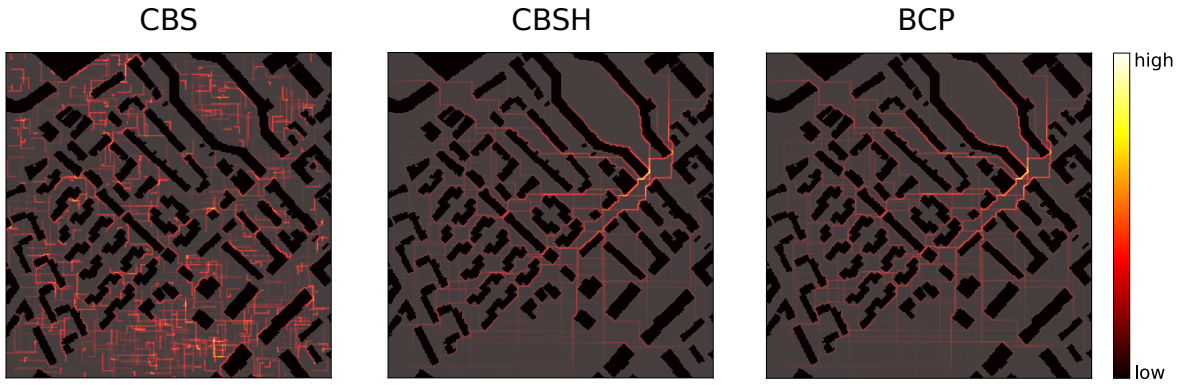


Figure A.3: Heat maps of the single-agent shortest paths for different algorithms for Boston.

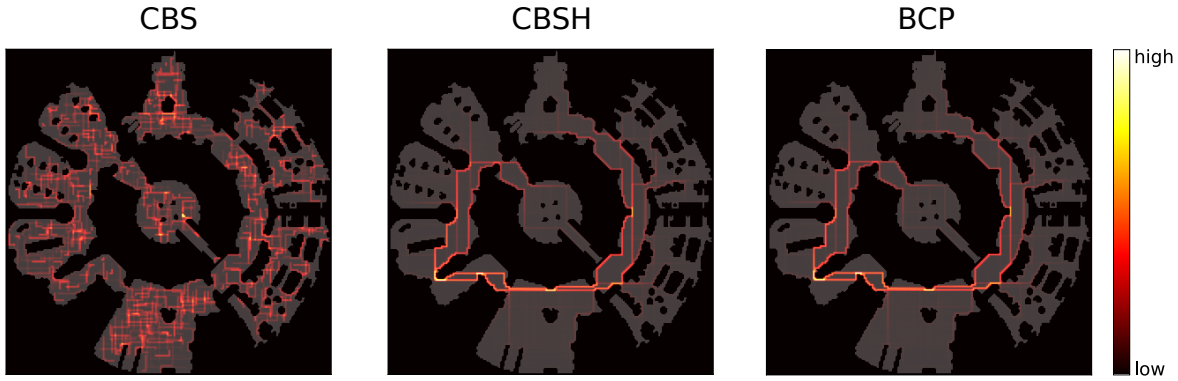


Figure A.4: Heat maps of the single-agent shortest paths for different algorithms for lak303d

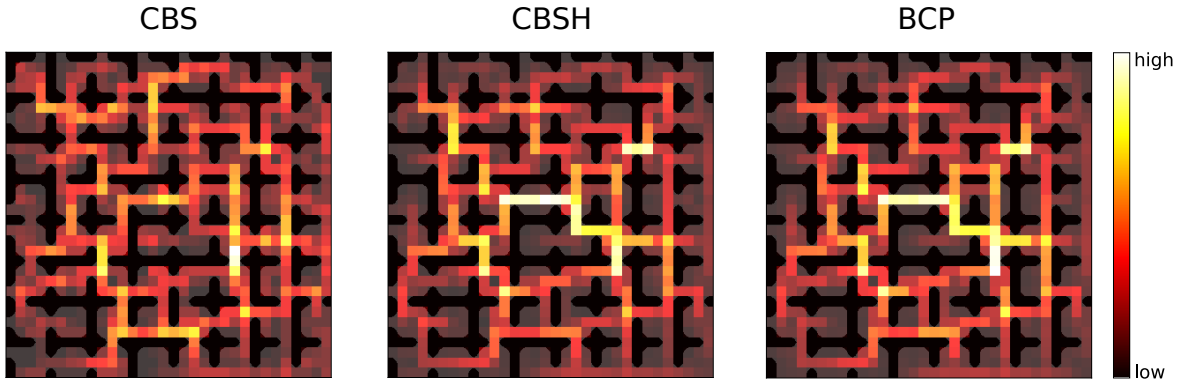
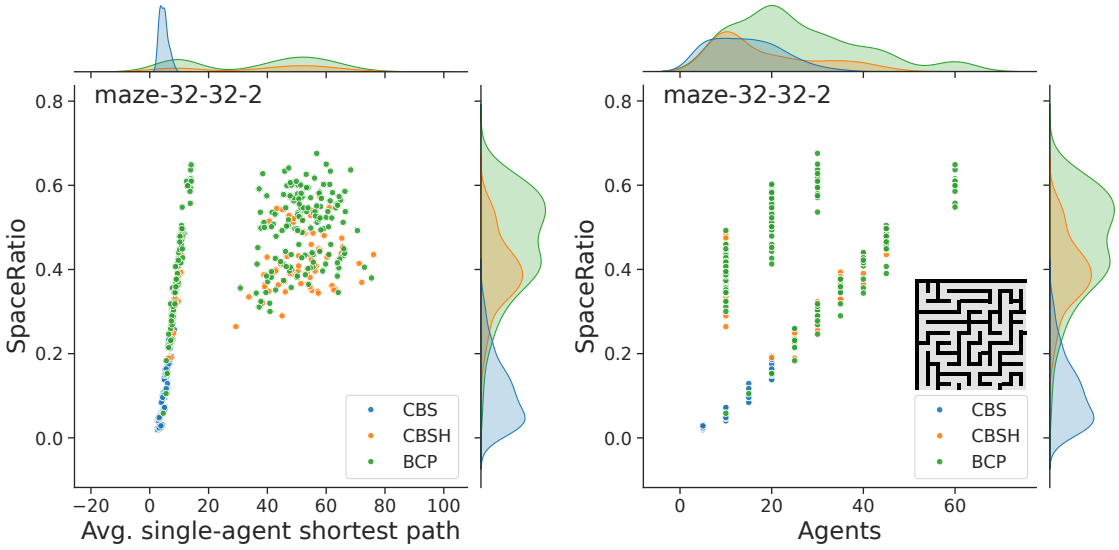
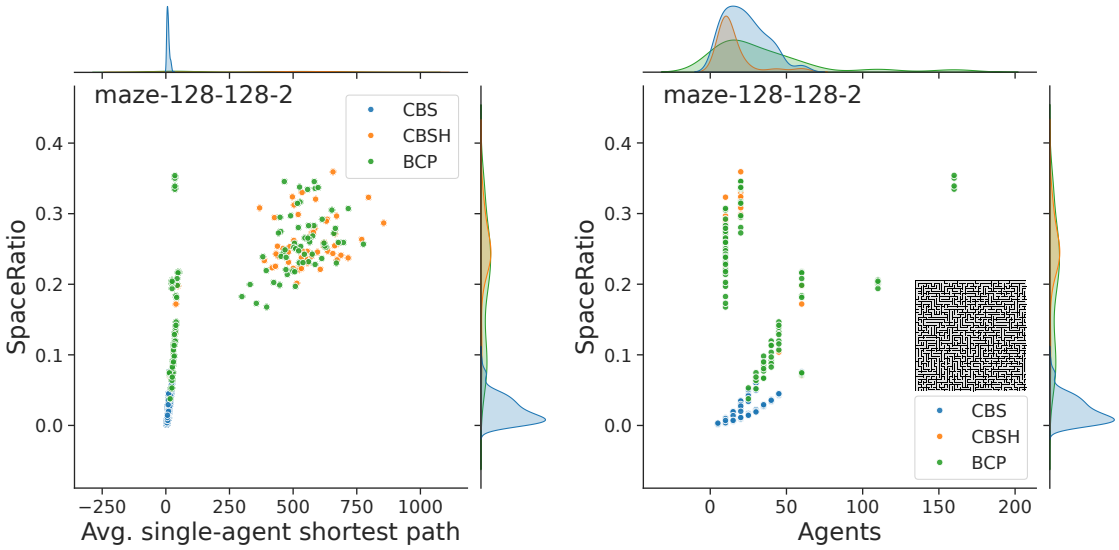


Figure A.5: Heat maps of the single-agent shortest paths for different algorithms for room-32-32-4.



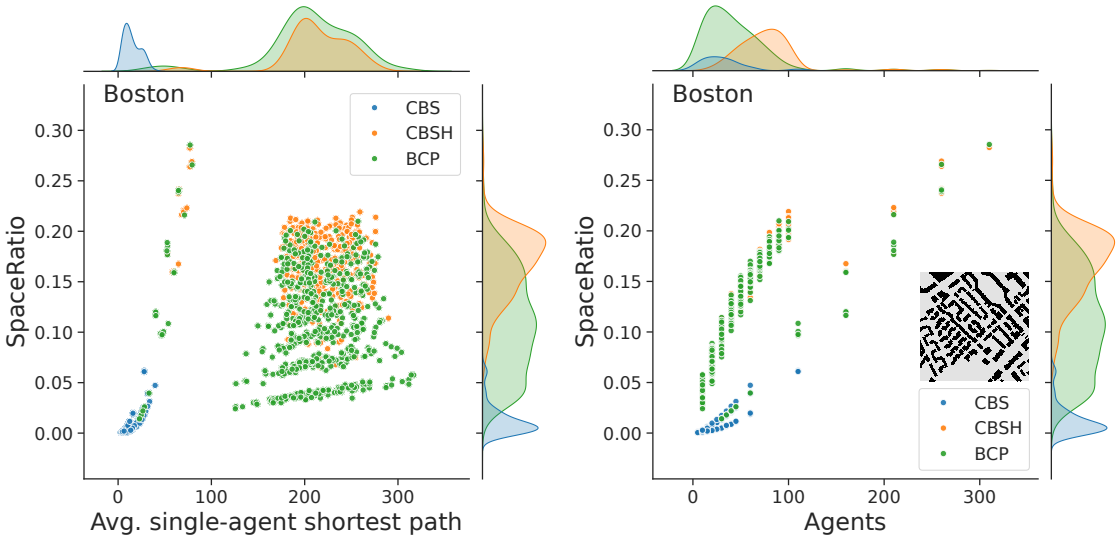
(a) (b)

Figure A.6: The scatter plots of maze-32-32-2 for average single-agent shortest path length and number of agents to SpaceRatio.



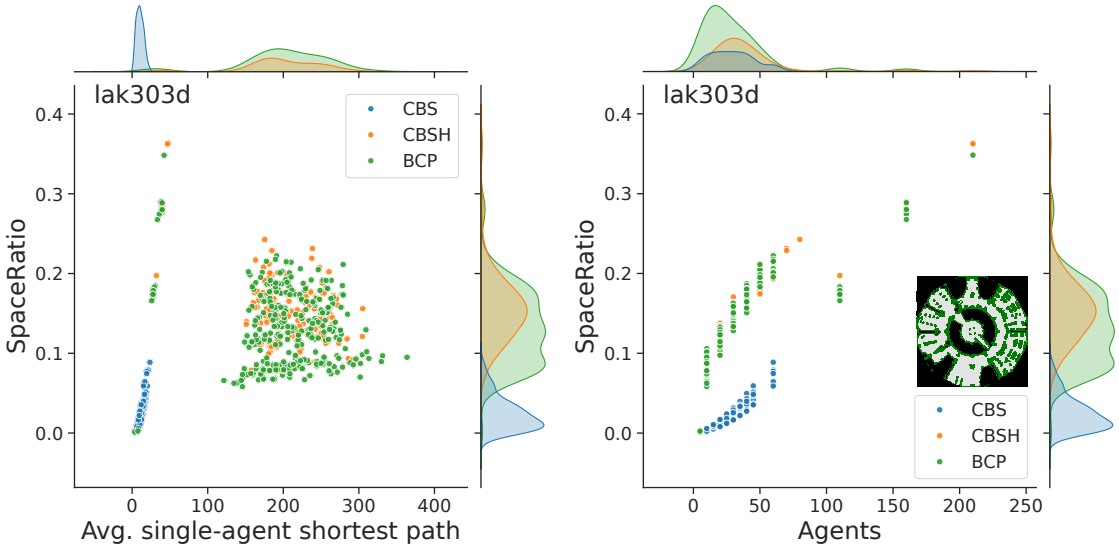
(a) (b)

Figure A.7: The scatter plots of maze-128-128-2 for average single-agent shortest path length and number of agents to SpaceRatio.



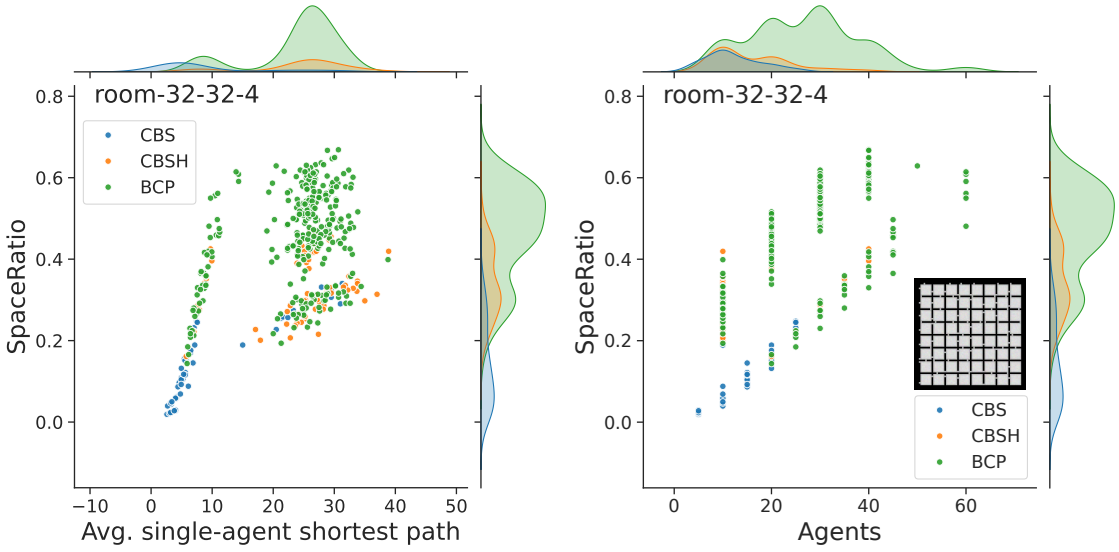
(a) (b)

Figure A.8: The scatter plots of Boston for average single-agent shortest path length and number of agents to SpaceRatio.



(a) (b)

Figure A.9: The scatter plots of lak303d for average single-agent shortest path length and number of agents to SpaceRatio.



(a) (b)

Figure A.10: The scatter plots of room-32-32-4 for average single-agent shortest path length and number of agents to SpaceRatio.

B Supplementary Figures for Chapter 5

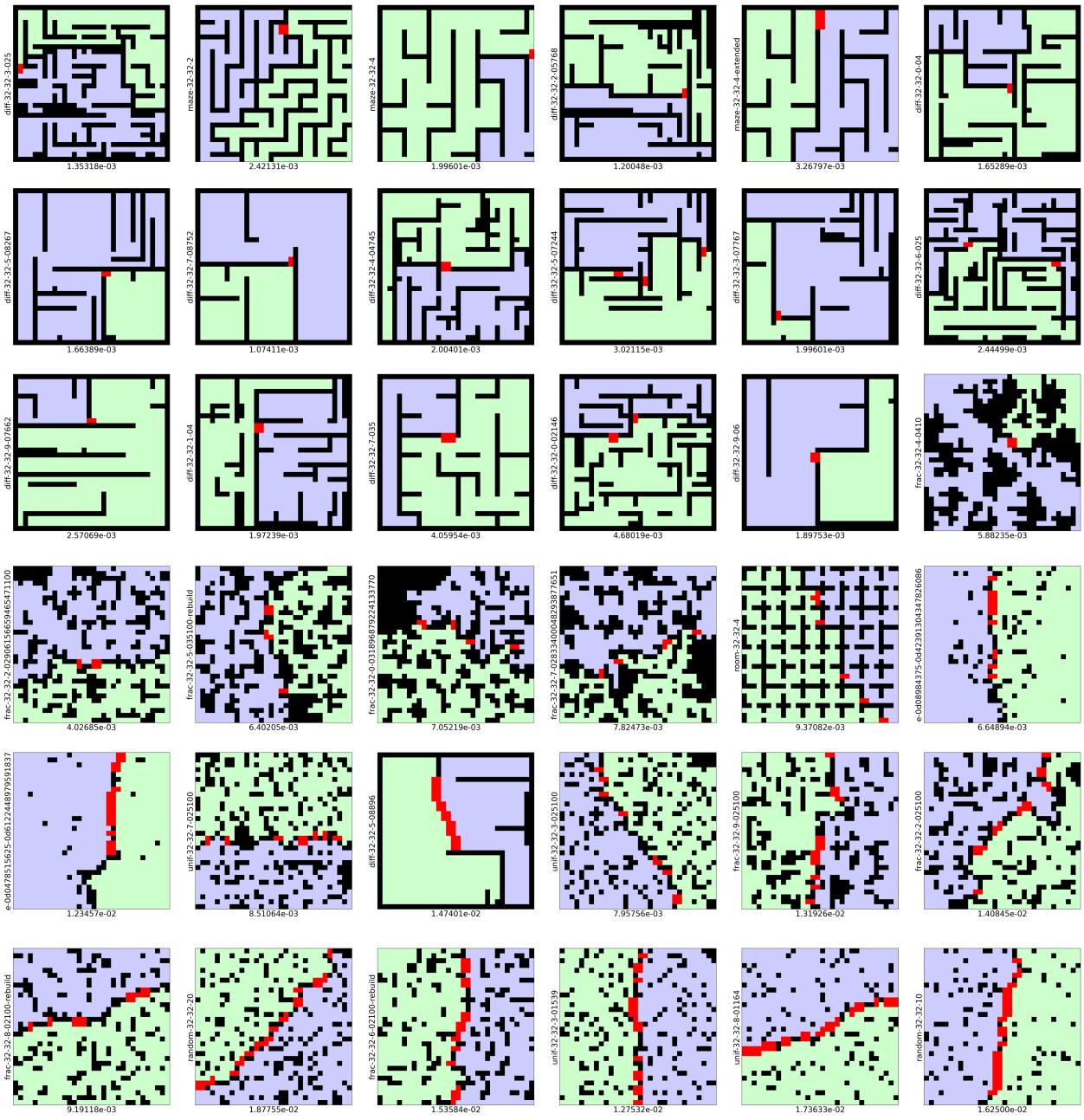


Figure B.1: Conductance cut of all maps in Section 5.5.

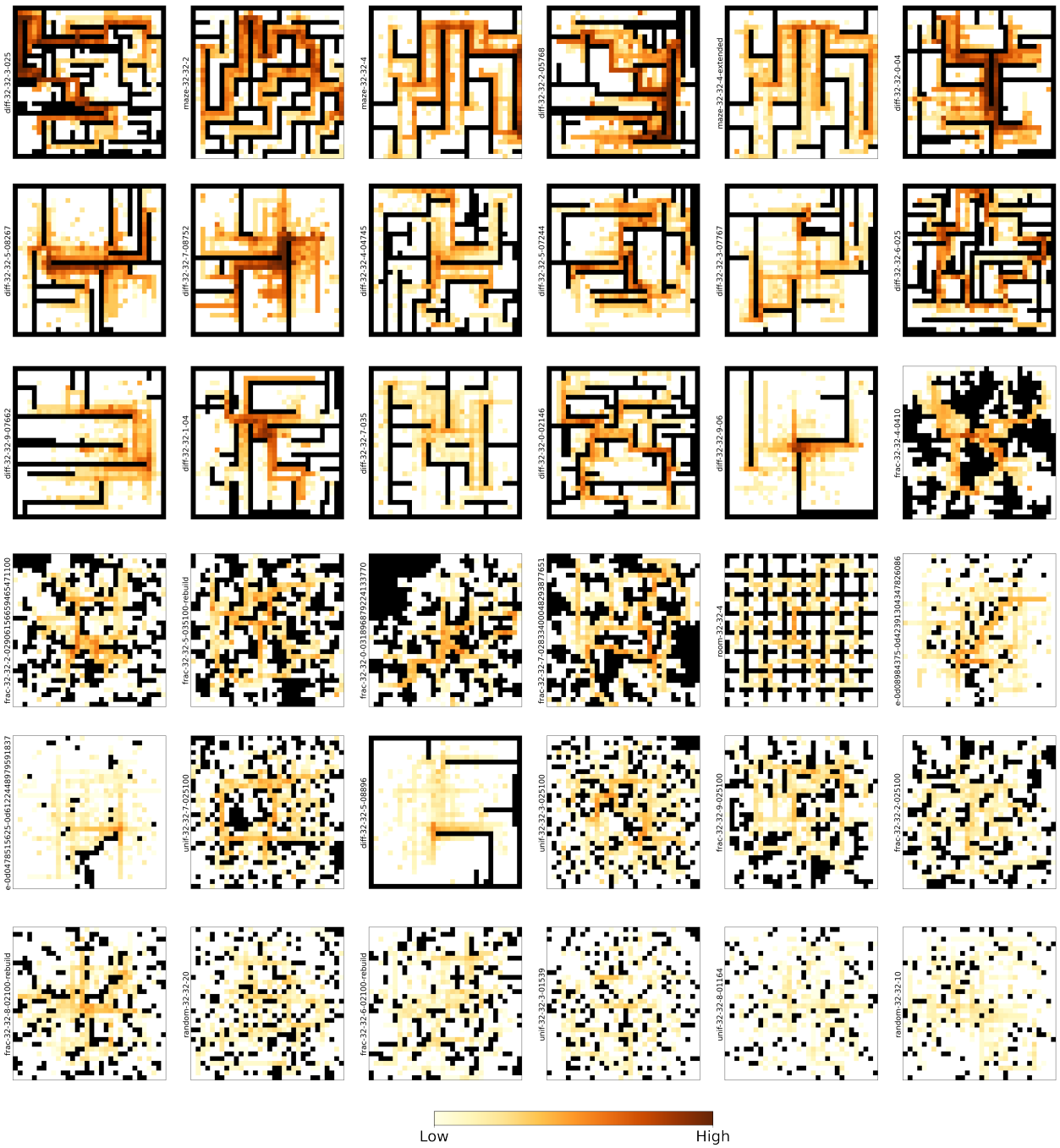


Figure B.2: Heatmap of conflicts for all maps in Section 5.5 when using the CBSH2-RTC algorithm.