Scalable Imitation Learning for Lifelong Multi-Agent Path Finding

He Jiang

June 30, 2025
CMU-RI-TR-25-67

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:

Prof. Jiaoyang Li, *chair*	The Robotics Institute, Carnegie Mellon University
Prof. Maxim Likhachev	The Robotics Institute, Carnegie Mellon University
Rishi Veerapaneni	The Robotics Institute, Carnegie Mellon University

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

With the advancement of artificial intelligence (AI) and robotics, autonomous agents are becoming an essential part of our daily lives. Future large-scale multi-agent systems—such as those envisioned in smart cities—may involve tens of thousands of ground vehicles, aerial drones, and diverse robots, all requiring highly efficient coordination.

This thesis focuses on a fundamental coordination problem underlying many such systems: Lifelong Multi-Agent Path Finding (LMAPF). LMAPF is the task of constantly planning efficient, collision-free paths for multiple agents, which get new goals every time they reach their current ones. Despite decades of research in related areas, the rapid increase in the number of agents in modern systems demands increasingly scalable algorithms—capable, for example, of coordinating thousands of mobile robots in automated warehouses operated by companies such as Amazon and Ocado.

Therefore, the main contribution of this thesis is to introduce Scalable Imitation Learning for LMAPF (SILLM), which can effectively learn a neural policy shared by up to ten thousand agents to generate high-quality single-step decisions based on their local observations. One of the key ingredients in our learning recipe is the scalable expert, a windowed anytime search-based solver named Windowed PIBT-LNS (WPL), originally proposed as a part of our winning solution in the 2023 League of Robot Runners (LoRR) competition. To emulate WPL's decision-making logic, we carefully design a neural policy representation that integrates a novel Spatially Sensitive Communication (SSC) module that allows for precise spatial reasoning, with other heuristic search progresses on single-step collision resolution and global heuristic guidance.

Our algorithm substantially outperforms state-of-the-art learning-based solvers in both inference speed and solution quality. When planning time is limited, it also surpasses existing search-based methods, including our competition-winning solution.

Acknowledgements

Research in this thesis was supported by the National Science Foundation (NSF) under Grant #2328671, a gift from Amazon, and an Amazon Research Award.

This thesis draws heavily from two of my previously published works: WPPL [1], which I first-authored, and SILLM [2], which I co-first-authored. In WPPL, I devised, implemented, and evaluated the core algorithm. The discussion of existing challenges and future directions in LMAPF was a collaborative effort among all co-authors. In SILLM, I proposed the Spatially Sensitive Communication module, explored multiple forms of global heuristic guidance, and designed the overall training pipeline. The idea of applying CS-PIBT to learning-based methods for single-step safeguarding was originally proposed by Rishi Veerapaneni [3], first integrated into reinforcement learning by Yutong Wang, and later extended by me to both reinforcement and imitation learning within SILLM. Yutong and I conducted simulation experiments, while Tanishq Duhan developed the real-world demonstration.

Lastly, I would like to express my heartfelt gratitude to Rishi Veerapaneni, Yulun Zhang, Yutong Wang, Tanishq Duhan, Professor Jiaoyang Li, Professor Guillaume Sartoretti, and Professor Maxim Likhachev for their invaluable support and guidance throughout this work.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, we have witnessed remarkable advances in the real-world deployment of AI and robotics. Digital assistants powered by large language models, such as ChatGPT and Claude, have become invaluable partners in our daily lives. Meanwhile, autonomous vehicles—exemplified by Waymo's fully driverless taxis, operating in cities like Phoenix and San Francisco—are transforming urban mobility. It can be expected that in the near future, we will live in a world populated with autonomous agents. For example, smart cities may involve millions of ground vehicles, aerial drones, and diverse robots. A key challenge will then be how to coordinate these agents efficiently, especially given the limited resources.

In this thesis, we focus on a fundamental coordination problem known as Lifelong Multi-Agent Path Finding (LMAPF) [4], which underlies many real-world multi-agent systems such as automated warehouses, traffic systems, and virtual games. LMAPF extends the classic Multi-Agent Path Finding (MAPF) [5] problem by requiring continuous planning of efficient, collision-free paths for multiple agents that are assigned new goals each time they reach their current ones. The primary objective of LMAPF is to maximize throughput, defined as the average number of goals reached by all agents per timestep.

In recent years, with rapid growth of agent numbers, more and more scalable search-based solvers have been developed for LMAPF [1, 6–8], with state-of-the-art ones [1, 8] capable of scaling to thousands of agents. Since learning-based solvers are expected to be more decentralized and scalable than search-based ones, numerous studies on learning have also been conducted following the pioneering work PRIMAL [9] since 2019.[1]

However, most solvers have only been tested on small-scale instances only involving tens to hundreds of agents so far [9–13]. Additionally, most learning papers emphasize the scalability of their solvers compared to optimal or bounded-suboptimal search-based solvers [14, 15], often showing positive results. This is mainly because these search-based solvers struggle with computational complexity as solving MAPF optimally is NP-hard [16]. However, when compared to scalable suboptimal search-based solvers like PIBT [7, 17] (first introduced in 2019), most state-of-the-art learning-based solvers, such as PICO [10] or SCRIMP [11], would actually fail to beat them in terms of both planning speed and solution quality. Only very recently has a

[1]Since solvers for MAPF can be easily adapted for LMAPF, we do not differentiate between MAPF and LMAPF in the rest of this paper, unless necessary.

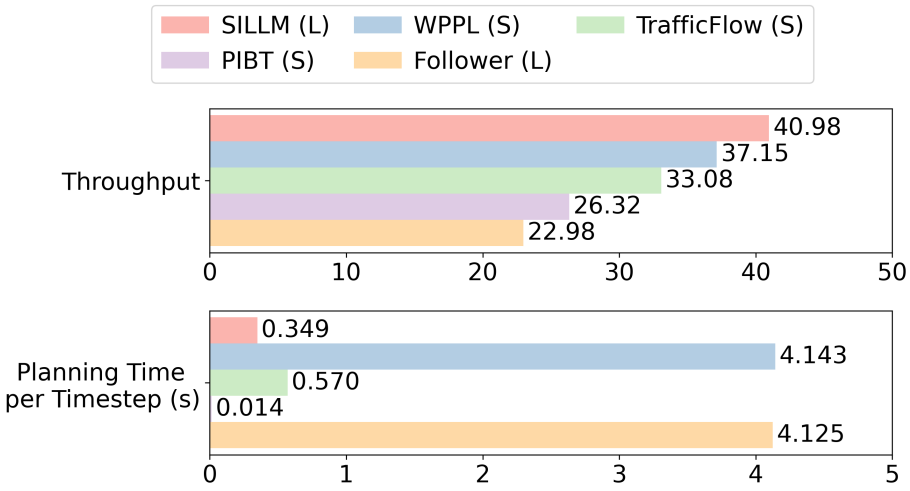


Figure 1.1: Comparison of mean throughput and planning time per step between our solver, SILLM, with other state-of-the-art search- and learning-based solvers on 6 maps with 10,000 agents. The (L) and (S) in the legend denote learning-based and search-based solvers. Details are given in Table 5.2.

learning method, Follower [13], shown to outperform PIBT in throughput on small maps with up to 192 agents. Indeed, the recent state-of-the-art search-based solver, TrafficFlow [8], built on top of PIBT, can act as an even stronger baseline.

In this thesis, we introduce Scalable Imitation Learning for LMAPF (SILLM) [2], a learning-based solver that can manage up to $10,000$ agents after training on down-scaled scenarios with hundreds of agents. Unlike prior works, we choose to directly imitate a scalable expert, a windowed anytime search-based solver named Windowed PIBT-LNS (WPL), the core part of our winning solution, WPPL [1], in the 2023 League of Robot Runners (LoRR) LMAPF competition [18]. To emulate WPL's decision-making, we design a novel communication architecture, the Spatially Sensitive Communication (SSC) module that emphasizes precise spatial reasoning to better imitate highly cooperative actions seen in efficient LMAPF solutions. Furthermore, we systematically integrate recent advancements in heuristic search, specifically single-step collision safeguarding [3] and global heuristic guidance [1, 8].

Our experimental results show that SILLM outperforms state-of-the-art learning- and search-based solvers on six large maps from common benchmarks with up to 10,000 agents and varying obstacle structures, achieving average throughput improvements of 137.7% and 16.0%, respectively. We further demonstrate that SILLM,

equipped with dynamic global guidance described in Section 4.3.3, is even able to outperform WPPL. Notably, with GPU acceleration, SILLM requires less than one second per timestep to plan for 10,000 agents. Figure 1.1 presents a comparison of state-of-the-art search-based and learning-based solvers in terms of average throughput and planning time.

This work highlights the effectiveness of learning-based methods for large-scale LMAPF instances and offers a comprehensive framework to unlock the full power of learning in future research. The central concepts—like anytime local optimization, hierarchical system decomposition, and combining search with learning—are also broadly applicable to other multi-agent systems that feature high-dimensional joint action spaces and require long-horizon planning.

2 Problem Formulation

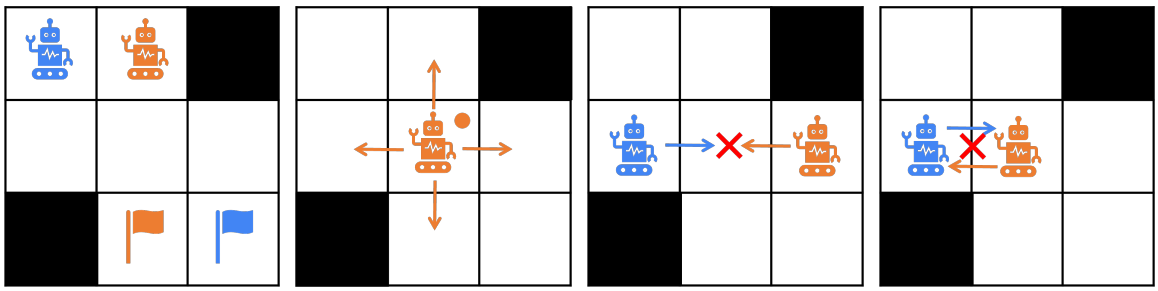
In this thesis, we study an extended problem of the Multi-Agent Path Finding (Definition 1), named Lifelong Multi-Agent Path Finding (Definition 2).

Definition 1 (MAPF)**.** Multi-Agent Path Finding (MAPF) takes in a directed graph $G = (V, E)$, where V are vertices and E are edges, and a set A of n agents with their start and goal vertices. Time is discretized into timesteps where at each timestep, an agent can take an action to move along an edge to an adjacent vertex or wait at its current vertex. Vertex collision occurs when two agents move to the same vertex at a timestep, and edge collision occurs when two agents swap their vertices at a timestep. Our target is to find collision-free paths for all agents to their goals while minimizing the total path costs, namely the total number of actions taken by all the agents.

Definition 2 (LMAPF)**.** Lifelong MAPF (LMAPF) extends MAPF by constantly assigning new goals to agents via a task assigner. The target is to find collision-free paths for all agents while maximizing throughput, namely the average number of goals reached by all agents per step.

We specifically study four-neighbor grid maps. Each empty location on the grid corresponds to a vertex in V, and two adjacent empty locations are connected by two edges of opposite directions in E. Agents are supposed to avoid the remaining locations that are static obstacles, in addition to the vertex and edge collisions with other agents. At each step, an agent can take five actions: move east, south, west, and north, or wait at the current vertex. We provide a simple example in Figure 2.1. In this thesis, we primarily focus on pathfinding and simply assume that each agent reaching its current goal will be assigned a new goal uniformly sampled from V.

Since directly optimizing throughput in LMAPF is challenging, we instead attempt to solve a series of MAPF problems repeatedly: given the agents' current locations and goals, minimize the total path cost of all agents.



(a) State Space (b) Action Space (c) Vertex Collision (d) Edge Collision

Figure 2.1: A simple MAPF example. (a) At each step, the state of the system is composed of agents' current locations (robots) and goals (flags). The black grids are the obstacles. (b) At each step, an agent could move in one of four directions (arrows) or wait at its current location (dot), and collision with obstacles is disallowed. (c) Vertex collision occurs when two agents move into the same location, and is disallowed. (d) Edge collision occurs when two agents swap their locations, and is also disallowed.

3 Related Work

3.1 Search-Based Solvers

For LMAPF, RHCR [6] is one of the most widely used baselines, which introduces the planning window to reduce the computation. However, its planning in each time window still relies on some existing MAPF solvers that are relatively computationally heavy, such as ECBS [19] and PBS [20]. Therefore, even though it has a high solution quality, it is still relatively slow and can scale up to hundreds of agents. On the other hand, PIBT [7] is a greedy, single-step planner that is very fast and scalable. It can coordinate thousands of agents but has a relatively low solution quality. Therefore, TrafficFlow [8] incorporates traffic information to help PIBT avoid congestion and improve solution quality. WPPL [1], instead, exploits PIBT to generate an initial windowed plan and then refines the plan by windowed MAPF-LNS [21], an anytime algorithm that iteratively selects a small group of agents and tries to improve their paths.

Anytime algorithms provide a flexible trade-off between solution quality and runtime, making them especially well-suited for large-scale problems with limited planning time. Among these, MAPF-LNS stands out due to its superior scalability and faster convergence compared to other anytime approaches for MAPF, which either scale poorly [22] or improve solution quality much more slowly [23, 24]. Various enhancements and extensions to MAPF-LNS exist [25–28], and a recent comprehensive study [29] offers a unified benchmark and in-depth analysis of these variants. While these developments are valuable, they are not central to the scope of this thesis and can be incorporated when necessary.

3.2 Learning-Based Solvers

PRIMAL [30] is the pioneering work that adopts learning to solve the MAPF problem. It leverages the homogeneity of agents to train a decentralized policy with shared neural network weights. Subsequent research has focused on improving it from four main directions: enabling communication between agents, incorporating global guidance into agents' field of view (FoV), enhancing collision resolution, and imitating search-based MAPF algorithms.

Communication enables agents to observe beyond their FoV and is critical to agents' local coordination. Most of the earlier works either use attention-based graph

neural networks [31–33] or simple multi-head attention structures [11, 34–36], where the spatial relationships between agents are supposed to be implicitly captured. In this thesis, we instead emphasize the importance of explicitly modeling the relative locations between agents and propose our Spatially Sensitive Communication module.

Limited by the local view, agents need to rely on global guidance to move toward their goals. Previous work mainly chose to encode agents' shortest paths to their goals in their FoV [9, 11, 32–37], but these individual shortest paths could easily cause congestion and harm the throughput in LMAPF. Only recently, Follower [13] began incorporating local congestion into guide path computation. In this thesis, we further argue for the importance of alleviating global congestion and thoroughly examine two congestion-aware global guidance methods beyond the general single-agent shortest path heuristic, which we denote as "backward Dijkstra" heuristic (as it can be computed via a backward Dijkstra).

Because actions from the agents' policies are sampled independently, collisions may still occur. Most prior works opt to freeze the movements of agents that may collide [9, 13, 32, 34–37], a strategy that is clearly inefficient. SCRIMP [11] and EPH [33] address collisions by reselecting agents' actions according to their priorities and learned value functions. However, both focus on only a single pair of colliding agents and lack an efficient recursive collision resolution mechanism for a chain of collisions. Therefore, their approaches become very time-consuming in congested scenarios. In this thesis, we showcase the importance of incorporating well-founded search-based solvers. Specifically, we adopt the idea of CS-PIBT [3], which utilizes PIBT [17], the greedy single-step search algorithm mentioned in Section 3.1, to shield agents from collisions. Concurrent work SILL [38] also proves the effectiveness of this one-step collision resolution method in learning.

Most previous methods apply reinforcement learning to train neural policies directly [13, 32–37]. However, it is challenging to design cooperative reward functions and to explore the huge joint action space. Since MAPF has a perfect dynamic model of the system, state-of-the-art search algorithms, such as MAPF-LNS [21] and LaCAM* [39], typically provide significantly better solution quality, albeit at the cost of longer planning times. Thus, it is reasonable to apply imitation learning from search-based solvers in MAPF. However, previous methods trained by imitation learning or a mixture of imitation and reinforcement learning [9, 11, 31] all choose to emulate the behaviors of some optimal or bounded suboptimal algorithms, such as ODrM* [15]

or ECBS [19]. Due to the limited scalability of these expert search algorithms, the resulting policies are trained only on small instances and consequently perform significantly worse on larger ones. Therefore, in this thesis, we choose to directly imitate a scalable anytime search-based solver, which inherently balances the tradeoff between solution quality and planning time, as discussed in Section 3.1.

In Table 3.1, we summarize these learning-based solvers proposed in recent years that are commonly used as baselines. There is also another interesting concurrent work named MAPF-GPT [40], which tries to build a foundation model with modern Transformer networks and tokenization techniques. This method relies more on the general modeling ability of Transformer and less on the domain-specific design for MAPF; thus, it is not included in the comparison. We believe the ideas discussed in this thesis will also improve its performance.

Table 3.1: Overview of learning-based solvers. The "Max #Agents" is based on experiments reported in the papers. Some methods allow communication between agents: "ABC" denotes attention-based communication, and "SSC" denotes our spatially sensitive communication. All methods use agents' FoV as inputs, while some also incorporate global guidance, such as a path calculated by A* (Path) or binary features indicating whether a movement brings the agent closer to its goal (Movements). Our method, SILLM, uses three types of global guidance: Backward Dijkstra (BD), Static Guidance (SG), and Dynamic Guidance (DG), which are explained in Section 4.3.3. At each timestep, collisions are resolved by either keeping agents in their previous locations (Freeze), allowing them to reselect new actions (Reselect), or applying Collision Shield PIBT (CS-PIBT) [3]. All methods are trained via imitation learning (IL), reinforcement learning (RL), or both (Mixed). The imitation objective is the search-based solver adopted for imitation. ODrM* [15] and ECBS [19] are bounded suboptimal search algorithms. Our imitation objective, WPL, is detailed in Section 4.1.

	PRIMAL 2019 [30]	MAPPER 2020 [37]	PRIMAL2 2021 [9]	MAGAT 2021 [31]	DHC 2021 [34]	DCC 2021 [35]
Max #Agents	1,024	150	2,048	1,000	64	128
Communication	None	None	None	ABC	ABC	ABC
Global Guidance	None	Path	Path	None	Movements	Movements
Collision Resolution	Freeze	Freeze	Freeze	Freeze	Freeze	Freeze
Training Approach	Mixed	RL	Mixed	IL	RL	RL
Imitation Objective	ODrM*	None	ODrM*	ECBS	None	None
	SACHA 2023 [32]	RDE 2023 [36]	SCRIMP 2023 [11]	EPH 2024 [33]	FOLLOWER 2024 [13]	SILLM (Ours) 2025 [2]
Max #Agents	64	70	128	64	256	10,000
Communication	ABC	ABC	ABC	ABC	None	SSC
Global Guidance	Movements	Movements	Movements	Movements	Path	BD,SG,DG
Collision Resolution	Freeze	Freeze	Reselect	Reselect	Freeze	CS-PIBT
Training Approach	RL	RL	Mixed	RL	RL	IL
Imitation Objective	None	None	ODrM*	None	None	WPL

4 Methods

In this chapter, we first introduce our expert search-based solver, Windowed PIBT-LNS (WPL) in Section 4.1. Then, we show how to adapt its pipeline for scalable imitation learning in Section 4.2. Finally, we discuss our design of neural policy representation with search-based single-step safeguarding, spatially sensitive communication, and congestion-aware global guidance in each subsection of Section 4.3.

4.1 A Scalable Search-Based Solver: Windowed PIBT-LNS

Unlike MAPF, agents in LMAPF are repeatedly assigned new goals when they complete their current ones. Therefore, frequent replanning is required. To balance execution and planning, the planning time for each execution step is usually limited. In the 2023 League of Robot Runners (LoRR) Competition [18], planners are required to plan single-step actions for up to $10,000$ agents within 1 second. Such a requirement is very challenging. Therefore, we propose a windowed and anytime search-based solver, named Windowed PIBT-LNS (WPL), building on top of some existing state-of-the-art work in (L)MAPF: Rolling-Horizon Collision Resolution (RHCR) [6], Priority Inheritance with Backtracking (PIBT) [7, 17], and Large Neighborhood Search (LNS) [21].

4.1.1 Windowed Planning

With large agent numbers and map sizes, even planning the full shortest paths individually for all agents is computationally challenging within 1 second for a centralized planner. Therefore, we adopt the idea of windowed planning in RHCR [6]. Specifically, we replan for all agents at each timestep with a planning window of length w, and we only execute the first timestep of the plan.

Within the planning window, we apply some MAPF algorithms to resolve collisions between agents and compute the estimated path costs. The path costs beyond the window are approximated by the costs of agents' individual shortest paths to their current goals. The overall objective at timestep T is described by Equation (4.1).

$$Objective = \sum_{i=1}^{n}(\sum_{t=T}^{T+w-1} cost(v_t^i, v_{t+1}^i) + h(v_{T+w}, g^i)), \tag{4.1}$$

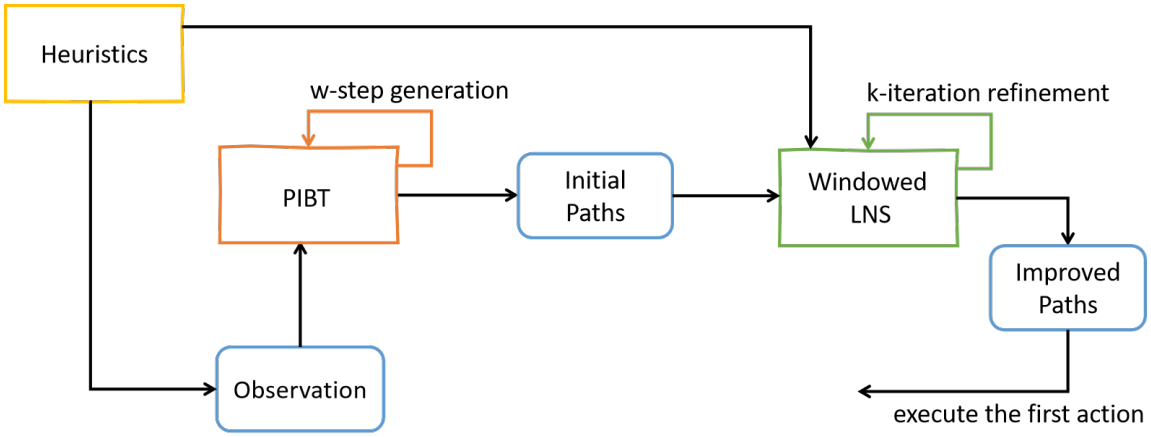


Figure 4.1: WPL's pipepline described in Section 4.1.

where v_t^a means the location of agent a at timestep t and g^a is its goal location. $cost(u, v)$ is the edge cost of moving from u to v (if $u = v$, then it is the wait cost). By default, the edge cost is 1, corresponding to taking 1 timestep. A special case arises when an agent reaches its goal in fewer than w steps; in this case, any subsequent edge costs after arrival are treated as 0. The function $h(v, g^i)$ estimates the cost of the path from location v to agent i's goal g^i. By default, we use the shortest path cost, namely the *Backward Dijkstra* heuristic, as the value of h.

4.1.2 An Anytime LMAPF Solver: PIBT-LNS

Given the time limit and large agent numbers (e.g., 1 second and $10,000$ agents), even with a planning window (e.g., 15 timesteps), most existing state-of-the-art MAPF solvers, such as ECBS [19] and PBS [20], struggle to even deliver a feasible solution. Therefore, we resort to a highly scalable algorithm, PIBT [7, 17], which can return a feasible single-step solution for $10,000$ agents within tens of seconds. The downside, however, is that PIBT has a relatively poor solution quality. Thus, we apply another anytime algorithm, LNS [21], to further improve the solution and fully exploit the given planning time. The overall pipeline of our Windowed PIBT-LNS (WPL) is illustrated in Figure 4.1, where we first call PIBT w times to generate the initial w-step paths for all agents. Then, we apply LNS to refine their w-step paths for k iterations.

In PIBT, each agent is assigned a priority, and at each timestep, every agent ranks its actions in ascending order of the heuristic function h as applied in Equation (4.1)

(e.g., the Backward Dijkstra heuristic). An agent always takes its highest-ranked action that does not collide with any higher-priority agent. If no collision-free actions exist for a low-priority agent, PIBT triggers a backtracking process, forcing the higher-priority agent to take its next best action until all agents can take collision-free actions.

In LNS, the algorithm will run k iterations of local optimization for the objective in Equation (4.1) to refine the plan. At each iteration, LNS selects a small group of agents heuristically (e.g., by selecting the most delayed agents and other agents causing the delays), and tries to optimize agents' w-step paths by replanning. If the path costs for this small group of agents improve, then LNS updates their new paths to the overall plan. For out method, we use a small group size of 8 agents. In this case, the refining iterations k could be thousands within 1 second. LNS can be further improved by parallelizing and simultaneously selecting and optimizing multiple neighborhoods, which leads to the Windowed Parallel PIBT-LNS (WPPL) solver [1] — the final version we applied in the LoRR competition.

4.2 Scalable Search to Scalable Imitation Learning

In WPL, we spend most of the computational time on LNS, and with longer computation time and more iterations of refinement, we typically get better solution quality[1]. One natural question is, can we collect the experiences of LNS search and compress it into a neural policy so that we can reactively output actions based on the current state without too much deliberate thinking?

Since we have a strong expert, we chose to use imitation learning. Compared to reinforcement learning, it is generally easier to learn team cooperation from mature search-based solvers because there is no need to explore the vast joint action space of multiple agents.

A few earlier works have applied imitation learning, and they primarily focus on mimicking bounded-suboptimal algorithms, such as ODrM* [15] or ECBS [19], as shown in the last row of Table 3.1. As a result, their training is typically restricted to small-scale instances due to the heavy computational demands of data collection. For instance, PRIMAL2 [9] and SCRIMP [11] were trained on instances with 8 agents, while MAGAT [31] was trained on instances with 10 agents. When applying these solvers to large-scale instances, the significant differences between the original training

[1]The performance of LNS will converge eventually. However, it is unbounded suboptimal.

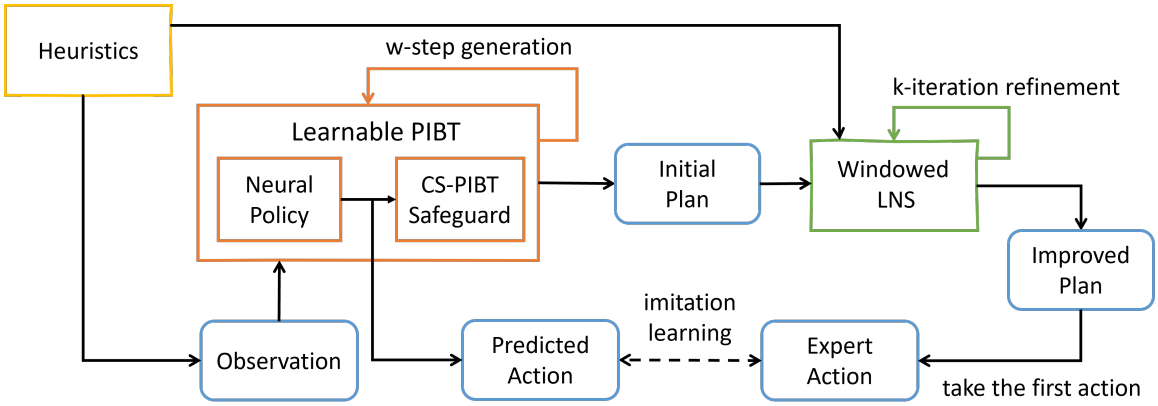


Figure 4.2: Data collection pipeline with a neural policy, described in Section 4.2.

setup and the actual application scenarios often result in a substantial decline in performance. Instead, WPL can handle large-scale instances efficiently on its own and does not suffer from this distribution shift problem significantly.

The data collection procedure for imitation learning can be easily adapted from WPL's pipeline in Figure 4.1 by replacing PIBT with a neural policy, as shown in Figure 4.2. Now, we can generate the initial w-step paths for all agents with the neural policy, and still refine them with k iterations of LNS. Then the first actions in the refined paths can be used as expert actions for later supervised training of the neural policy.

The imitation learning procedure can be repeated iteratively in a self-bootstrapping manner. After an iteration of supervised training, we obtained a better neural policy, which could generate better initial w-step paths. Then, we can potentially obtain better refined w-step paths by LNS, whose first actions are further used for the supervised learning of the neural policy.

In our implementation, for the first iteration, we still use PIBT[2] to generate the initial solution since it provides a better start than a random policy. For execution during the data collection, we use the expert actions in the first iteration and the neural policy's actions in later iterations.

[2]In Section 4.3.1, we can explain PIBT as a special case of Learnable PIBT with a probability prior by setting the bias in the logit prediction layer proportional to the distance heuristic.

4.3 Neural Policy Representation with Different Levels of Coordination

Although we have access to a strong expert, it is equally important to construct a neural policy representation that can compress the expert's reasoning and decision-making processes. In the following subsections, we introduce the techniques we adopt to support coordination at different levels, ranging from single-step to local and global.

4.3.1 Search-Based Single-Step Safeguarding

Directly using action predictions from a neural policy has one issue during inference: the generated actions may still lead to collisions, as they are independently sampled for each agent. Most earlier works "freeze" the agents' movements that potentially lead to collisions by replacing their original actions with wait actions [30], as shown in the fourth row of Table 3.1. However, this approach is inefficient as many unnecessary wait actions are introduced. To mitigate this issue, SCRIMP [11] proposes resampling agents' actions to avoid collisions; however, it struggles to scale to a large number of agents due to its heavy computational requirements.

Collision Shield PIBT (CS-PIBT) [3] proposes to use PIBT [7] to resolve single-step collisions. Recall that in PIBT, each agent ranks its actions by the distance heuristic. CS-PIBT instead applies biased sampling to sample a rank of actions from the action probabilities predicted by the neural policy.

We directly apply this idea with a simple modification: we always prioritize the action with the maximal probability predicted by the neural policy and then rank other actions as the original PIBT.[3] Importantly, if the learned actions are collision-free, then CS-PIBT will not change them. We still refer to this safeguarding procedure as CS-PIBT, but name the combination of the neural policy and CS-PIBT as Learnable PIBT (L-PIBT) from the perspective of learning. Now we also modify our data collection pipeline to its final version in Figure 4.3 by replacing the neural policy in Figure 4.2 with Learnable PIBT.

[3] A benefit of such a modification is that we can consider the modified CS-PIBT as a part of the environment because it only takes a single action rather than a probability distribution like the original CS-PIBT. In this way, we can run other reinforcement learning algorithms without any change. However, whether it will harm the inference should be investigated through further

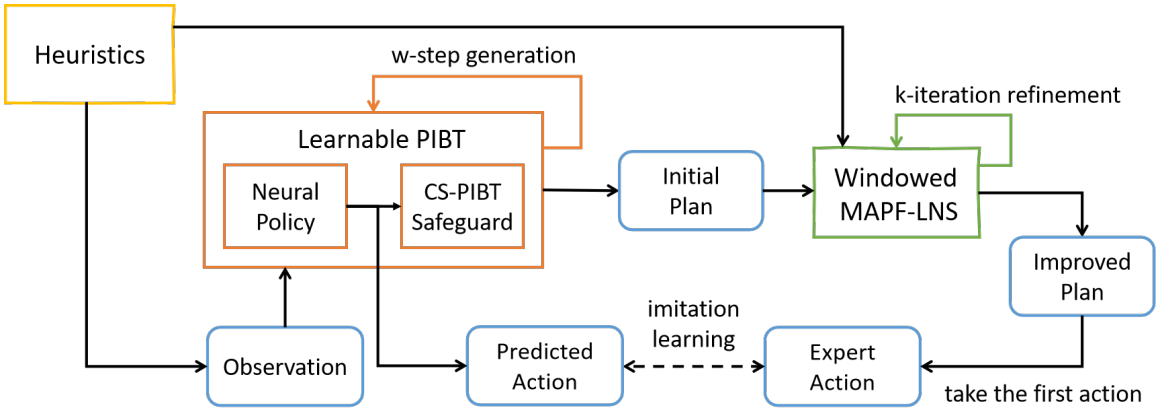


Figure 4.3: SILLM's data collection pipeline with a Learnable PIBT, described in Section 4.3.1.

4.3.2 Spatially Sensitive Local Communication

Communication is a key component in multi-agent systems, supporting efficient local coordination among agents. As shown in the second row of Table 3.1, existing works have mostly adopted attention-based communication (ABC) to aggregate information from neighboring agents, which only implicitly reason with spatial information. However, we argue that precise spatial information benefits local collision avoidance, as similar designs, such as the conflict avoidance table [41], are frequently applied in search-based solvers. Therefore, we propose a spatial sensitive communication (SSC) module that explicitly preserves the spatial relationship between agents when aggregating information.

Figure 4.4 illustrates the core structure of our neural network. The neural network comprises two Convolutional Neural Network (CNN) modules and a Spatially Sensitive Communication (SSC) module. Since all agents are homogeneous in LMAPF, they share the same network weights.

Each agent a_i has a local FoV of size $V_h \times V_w$ (set to $V_h = V_w = 11$ in our experiments), centered at its position, with a total of 5 feature channels. These channels are divided into two overlapping groups o_1^i and o_2^i. o_1^i, with a size of $4 \times V_h \times V_w$, consists of four channels representing the locations of static obstacles and dynamic obstacles (i.e., other agents), as well as absolute and relative heuristic values for global guidance (will be explained in Section 4.3.3). o_2^i, with a size of $3 \times V_h \times V_w$,

experiments.

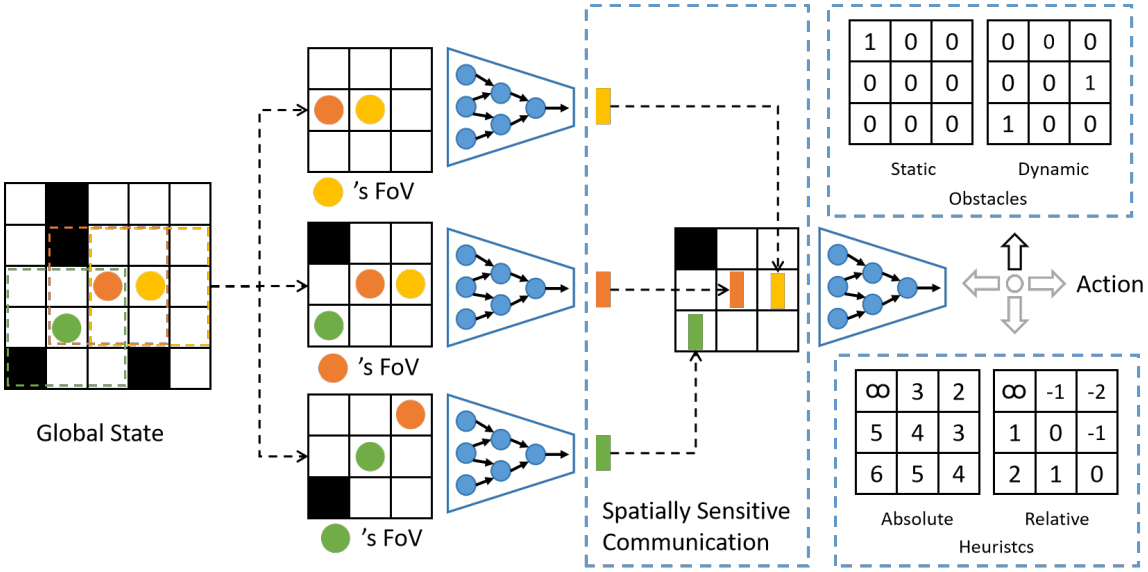


Figure 4.4: Core network structure. The global state has all static obstacles (black squares) and agents (colored circles). As an example, an agent's FoV is of size 3×3. The orange agent's unnormalized obstacle and heuristic feature maps are shown in the right upper and bottom corners. More details are covered in the section 4.3.2.

consists of three channels representing the locations of static and dynamic obstacles and the goal of agent a_i.

We first use a CNN module to convert o_1^i into a 32-channel feature vector f_i. Then, a_i gathers the feature vectors of all neighboring agents within its local FoV through communication.[4] Subsequently, the SSC module generates a matrix o_3^i of shape $32 \times V_h \times V_w$, filled with -1, and inserts the gathered feature vectors into the corresponding agents' relative positions in a_i's FoV (illustrated in Figure 4.4). o_3^i is then element-wise added to a matrix of the same shape, obtained by processing o_2^i through a Conv2d layer with a kernel size of 1. Finally, the resulting matrix is processed by an additional CNN module for further feature extraction and then decoded into probabilities over five possible actions.

4.3.3 Congestion-Aware Global Guidance

Since our neural policy only considers local observation, we need additional mechanisms to guide agents' movement towards goals. Many learning-based works incor-

[4]The communication range can be different from the range of FoV, but for simplicity, we set them to be the same in the experiments.

porate such mechanisms as summarized in the third row of Table 3.1. Some try to follow a specific shortest path [37, 42]. However, since the shortest path might not be unique, other works encode the shortest distance from every location in FOV to the corresponding goal [11, 12, 32, 34, 36]. A recent paper, Follower [13], tries to follow a shortest path that considers local traffic and is replanned at each timestep. However, no existing learning work is aware of the global traffic congestion, which is important for systems with large agent numbers.

Therefore, we systematically study three types of global guidance represented as heuristics that have been well studied in the search-based solvers. The first type of heuristics is the aforementioned **Backward Dijkstra (BD)** heuristics applied in previous learning works, which tell agents the shortest distances to their goals. Further, we evaluate two other heuristics that were not applied in the previous learning works but effectively reduce global traffic congestion in prior search-based works [1, 8].

The second type of heuristics is still Backward Dijkstra heuristics, but based on specially designed edge costs that reduce congestion offline. We call it **Static Guidance (SG)**. SG heuristics encourage agents to move in the same direction, avoiding head-on collisions. Specifically, we adopt the classic crisscross highways [43], where the encouraged directions are alternating row by row and column by column. In practice, the default cost of an edge is 2. The cost of an edge in the encouraged directions is 1, and the cost in the discouraged directions is 100,000 for warehouse or sortation maps and 3 for other maps. More advanced edge cost designs are also possible. For example, GGO [44] proposes an automatic way to optimize edge costs to maximize throughput.

The last type of heuristics is **Dynamic Guidance (DG)**, which further encodes dynamic traffic information, encouraging agents to move along short paths while avoiding congestion. We adopt the implementation in TrafficFlow [8], which plans a guide path for each agent and then asks agents to follow their guide paths as much as possible. When planning a guide path for an agent, TrafficFlow first counts the global traffic information based on the guide paths of other agents. Specifically, it counts the number of other agents visiting each location and traversing each edge along their guide paths. Then, dynamic edge costs are defined by handcrafted equations to punish the agent for moving to frequently visited locations and moving in the opposite direction of frequently visited edges. Therefore, the planned guide path by A* with

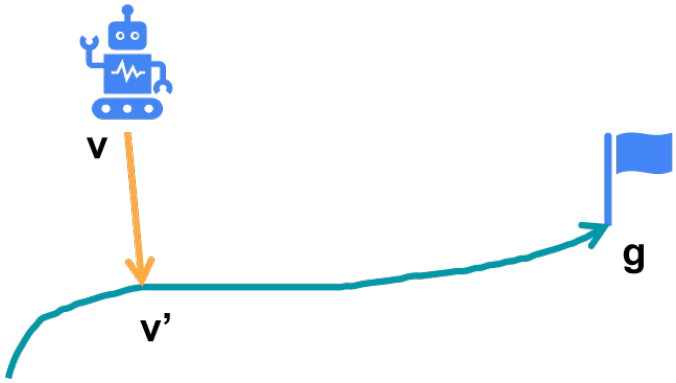


Figure 4.5: Computation of the dynamic guidance heuristic. The dark cyan curve represents the guide path to the agent's goal g, and the yellow line shows the shortest path from the robot's current location v to the guide path, where v' is the projection of v onto the guide path. The dynamic guidance heuristic is the sum of the distances from v to v' and from v' to g.

special edge costs could avoid potentially congested regions.[5]

Given the guide path of an agent i, whose current goal is g^i, the heuristic value of location v for agent i is defined as

$$h(v) = SPCost(v, v') + GPCost(v', g^i),$$

where v' denotes the closest location to v in the guide path, $SPCost(v, v')$ represents the shortest path cost from v to v', and $GPCost(v', g^i)$ is the remaining path cost from v' along the guide path to goal g^i. Notably, $SPCost$ can be easily computed by a backward A* search starting from all the locations in the guide path, while $GPCost$ can be directly obtained by summing the edge costs along the guide path. We illustrate the computation of h in Figure 4.5.

We encode these three different types of heuristics in a unified way into the neural policy's observation. Specifically, we denote the heuristic value at location v as $h(v)$ and encode the heuristic values in the FoV using a 2-channel 2D feature map of the same size as the FoV. The first channel is the absolute heuristic value normalized by the map size, i.e., the feature value at location v_i is $h(v_i)/(M_h + M_w)$, where M_h and M_w are the height and width of the map G. The second channel is the relative

[5]Readers interested in the implementation details are encouraged to refer to the original paper. We use the best variant in the paper.

difference in heuristic values, calculated by subtracting the heuristic value at the center of the FoV and dividing the FoV size, i.e., the feature value at location v_i is $(h(v_i) - h(v_c))/(V_h + V_w)$, where V_h and V_w are the height and width of the FoV, and v_c is the center of the FoV. Even though we explicitly add value normalization, we also apply a Batch Normalization to the input to accommodate the potential value range problem.

Importantly, no matter which heuristic we adopt, the h function should be consistently used in the approximate objective of WPL in Equation (4.1) so that the expert and neural policy can reason similarly.

It is worth noting that SG only changes the map weights and incurs no extra computational burden. However, DG needs to not only maintain guide paths for all the agents (the $GPCost$), but also compute the backward heuristic from the guide path (the $SPCost$) for each location considered in the PIBT or WPL. Thus, although we can run WPL with DG for hundreds of agents, we cannot afford it for larger instances with thousands of agents, which takes more than 10 seconds per timestep.

From another perspective, SG keeps the solver decentralized, but the guide path computation in DG requires a centralized server. However, the updates to guide paths do not occur at each timestep, but rather only when agents reach their current goals. Thus, the communication demand might be acceptable to some systems. For example, in our daily traffic system, drivers may ask for guide paths from a centralized server, such as Google Maps, but their driving along the road and interactions with other vehicles are decentralized.

5 Experiments

In this chapter, we evaluate our method on two benchmarks. The first is our main benchmark for evaluating our algorithms on large-scale mapf instances with $10,000$ agents [2]. The second is the benchmark used in the state-of-the-art learning paper, Learn-to-Follow [13], for fairly comparing only the decentralized solvers. The training are different for these two benchmarks and will be detailed separately in Section 5.1 and Section 5.2.

5.1 Large-Scale Instance Benchmark

5.1.1 Experiment Settings

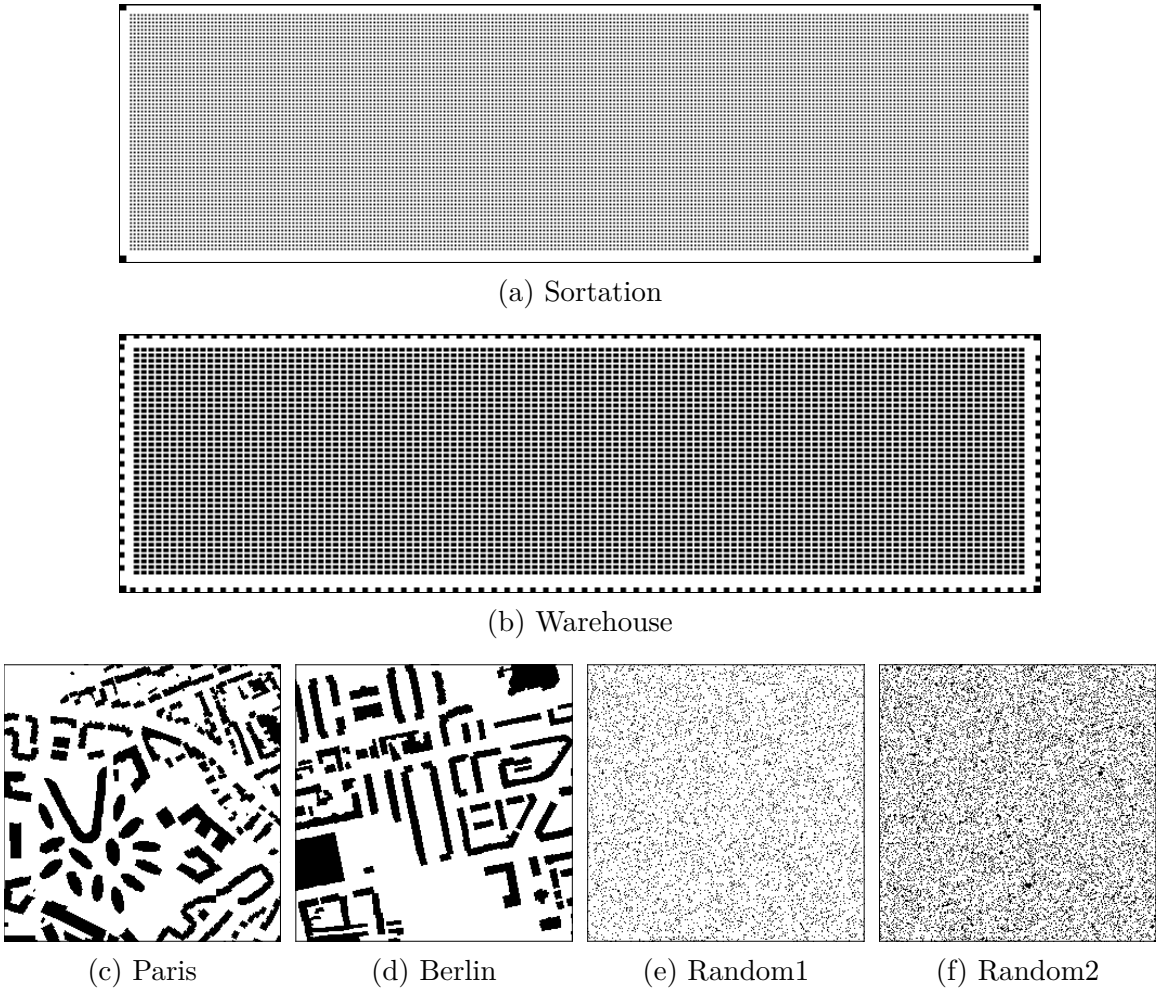
Applying imitation learning directly in large-scale settings may be possible, but complex engineering is required to deal with the large memory consumption during training. Therefore, we first downscaled the large maps to small ones but kept the original obstacle patterns, as illustrated separately in Figure 5.1 and Figure 5.2. More details about each map are recorded in Table 5.1.

We trained a neural policy with 600 agents on each small map and then evaluated it with $10,000$ agents on the corresponding large map. During training, we set the planning window w to 15 and the LNS refinement iterations k to 5000, so that each timestep takes around 1 second. For each map, we run 12 iterations of imitation learning, which roughly takes 12 hours with the computational resources detailed in Appendix A.1. The implementation is publicly available on GitHub[1].

5.1.2 Main Results

This subsection compares different variants of our solver, SILLM, with other state-of-the-art search-based and learning-based solvers. We use SCRIMP [11] and Follower [13] as learning-based baselines and PIBT [7], RHCR [6], and TrafficFlow [8] as search-based baselines. In addition, we also compare SILLM with WPPL [1], the winning solution of the 2023 LoRR competition [18]. To compare different solvers more easily, we compute scores in the same way as the competition: for each instance, we collect

[1]https://github.com/DiligentPanda/Scalable-Imitation-Learning-for-LMAPF



(a) Sortation

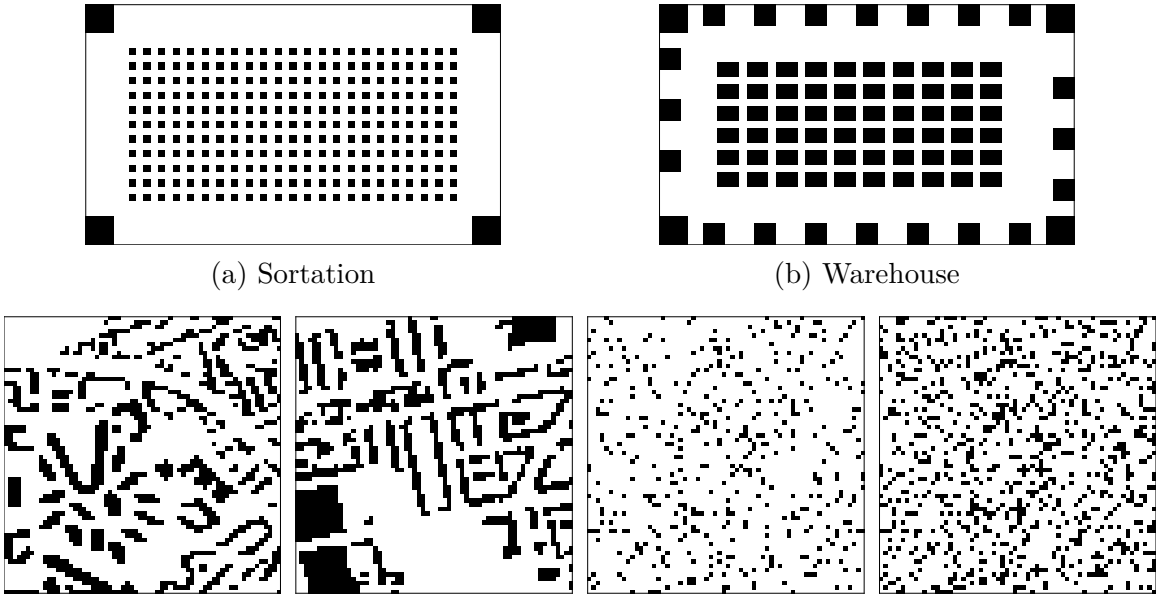
(b) Warehouse

(c) Paris (d) Berlin (e) Random1 (f) Random2

Figure 5.1: Large maps with $10,000$ agents for evaluation. The black grids indicate obstacles, and the white grids correspond to empty space. The size of subfigures might not be proportional to the actual map size for a better visual layout. The details of these maps are reported in Table 5.1.

the best throughput achieved among all the solvers evaluated in this paper. Then, a score between 0 and 1 is computed as the throughput of the solver divided by the best throughput.

First, we compare the performance of PIBT and Learnable PIBT (L-PIBT) with different global guidance on large instances in Figure 5.3. L-PIBT (solid line) consistently achieves better throughput than its counterpart (dashed line). The increase in the planning time is not negligible but acceptable. Different types of global guidance excel at different kinds of maps in our experiments. For example, Static Guidance,



(a) Sortation (b) Warehouse

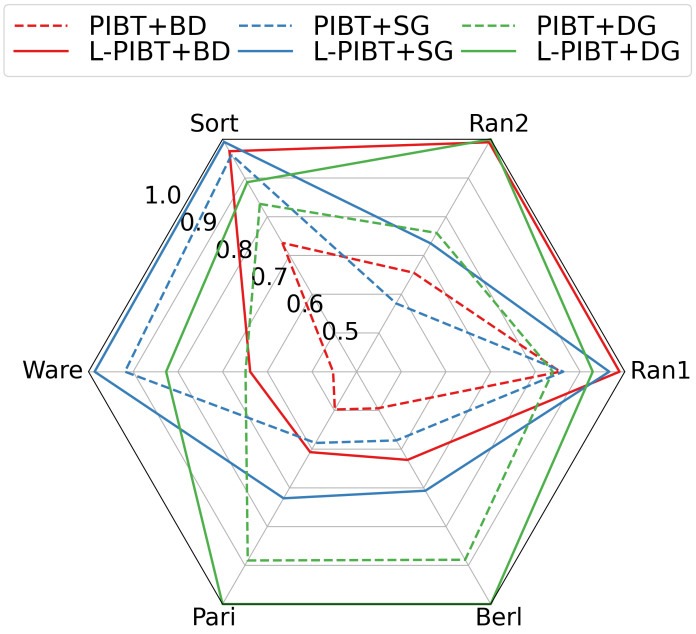
(c) Paris (d) Berlin (e) Random1 (f) Random2

Figure 5.2: Small maps with 600 agents for training. The black grids indicate obstacles, and the white grids correspond to empty space. The size of subfigures might not be proportional to the actual map size for a better visual layout. The details of these maps are recorded in Table 5.1.

Table 5.1: Table of Map Details. Sortation and Warehouse maps are from the LMAPF Competition [18]. Paris and Berlin maps are from the MovingAI benchmark [5]. Random1 and Random2 are randomly generated maps with 10% and 20% obstacle densities. We will use the underlined characters as the abbreviation for each map later. #Locs is the number of empty locations. #Agents is the number of agents in the map. Agent density is defined as the number of agents divided by #Locs. #Steps is the number of simulation timesteps.

Name	Small-Scale					Large-Scale				
	Size	#Locs	#Agents	Agent Density	#Steps	Size	#Locs	#Agents	Agent Density	#Steps
Sortation	33*57	1,564	600	38.3%	500	140*500	54,320	10,000	18.4%	3,200
Warehouse	33*57	1,277	600	47.0%	500	140*500	38,586	10,000	25.9%	3,200
Paris	64*64	3,001	600	20.0%	500	256*256	47,096	10,000	21.2%	2,500
Berlin	64*64	2,875	600	20.9%	500	256*256	46,880	10,000	21.3%	2,500
Random1	64*64	3,670	600	16.3%	500	256*256	58,859	10,000	17.0%	2,500
Random2	64*64	3,221	600	18.6%	500	256*256	52,090	10,000	19.2%	2,500

implemented as a criss-cross highway, fits the criss-cross patterns of sortation and warehouse maps better. For random instances, normal Backward Dijkstra actually



Score (Time)	BD	SG	DG
PIBT	0.61 (0.014s)	0.75 (0.014s)	0.81 (0.570s)
L-PIBT	0.80 (0.024s)	0.85 (0.023s)	0.94 (0.721s)

Figure 5.3: Comparison of L-PIBT and PIBT with different global guidance on large instances. The radar plot shows the score for each instance. The table reports the average score and the average planning time per timestep.

performs better. The potential reasons are: (1) uniformly generated random structures implicitly make agents select more distributed paths, and (2) with lower agent density and potentially less congestion, normal Backward Dijkstra is a more accurate estimation of the future timesteps. Dynamic Guidance performs better on city-type maps (i.e., Paris and Berlin), where obstacle patterns are less uniform. Regarding the average score on large instances, Dynamic Guidance performs the best, though it takes much longer to plan the guide paths and compute the backward heuristic from guide paths. In contrast, the other two guidance methods could be precomputed in advance and thus run much faster during execution.

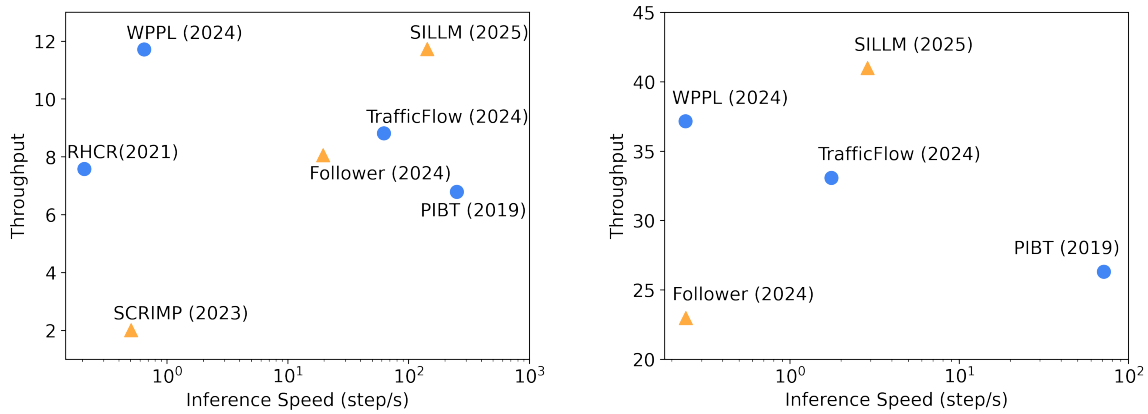
We pick the best performance achieved by L-PIBT with one of the global guidance to form the results of our solver, SILLM, in Table 5.2. We also report the performance of applying only a single type of global guidance separately as L-PIBT+BD,

Table 5.2: Comparison of different solvers. The left part is the result of downscaled small instances. The right part is the result of the original large instances. We evaluate each instance with 8 runs of different starts and goals. Each column records the mean throughput with the standard deviation in parentheses. The Time (in seconds) and Score refer to the average single-step planning time and the average score, respectively. "-" indicates unavailable data due to the excessive planning time. We rank the solvers in descending order based on their average score on large-scale instances. The best throughput of each map is marked in bold. Notably, PIBT and TrafficFlow are exactly PIBT+BD and PIBT+DG in Figure 5.3.

Algorithm	Small maps with 600 agents								Large maps with 10,000 agents							
	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score	Sort	Ware	Pari	Berl	Ran1	Ran2	Time	Score
SILLM	16.52 (0.09)	**14.28** (0.12)	8.85 (0.11)	**7.19** (0.14)	12.73 (0.11)	**10.76** (0.12)	0.007	1.00	43.98 (0.07)	42.24 (0.05)	**31.00** (0.89)	**30.52** (0.98)	53.74 (0.10)	**44.37** (0.11)	0.349	0.99
L-PIBT+DG	14.41 (0.16)	11.67 (0.54)	8.34 (0.07)	6.77 (0.28)	11.14 (0.13)	9.18 (0.14)	0.029	0.88	39.38 (0.09)	35.39 (0.26)	**31.00** (0.89)	**30.52** (0.98)	50.48 (0.14)	**44.37** (0.11)	0.721	0.94
WPPL [1]	**16.74** (0.06)	13.90 (0.16)	**8.91** (0.07)	7.10 (0.20)	**13.00** (0.09)	10.58 (0.15)	1.546	0.99	**44.28** (0.07)	**42.83** (0.04)	23.74 (0.20)	20.35 (0.29)	**54.38** (0.09)	37.30 (0.43)	4.143	0.88
L-PIBT+SG	16.52 (0.09)	14.28 (0.12)	8.85 (0.11)	7.19 (0.14)	12.34 (0.11)	10.09 (0.10)	0.007	0.98	43.98 (0.07)	42.24 (0.05)	22.54 (0.17)	21.59 (0.23)	52.49 (0.10)	32.44 (0.31)	0.023	0.85
TrafficFlow [8] (PIBT+DG)	11.78 (0.20)	9.10 (0.42)	7.44 (0.24)	5.70 (0.43)	10.35 (0.37)	8.48 (0.10)	0.016	0.76	36.89 (0.22)	27.78 (0.88)	27.51 (0.69)	27.03 (0.54)	45.62 (0.84)	33.64 (2.32)	0.570	0.81
L-PIBT+BD	14.76 (0.38)	8.69 (0.33)	7.76 (0.17)	7.16 (0.62)	12.73 (0.11)	10.76 (0.12)	0.007	0.89	42.91 (0.07)	27.34 (1.31)	18.84 (0.20)	19.16 (0.33)	53.74 (0.10)	44.02 (0.90)	0.024	0.80
PIBT+SG	13.66 (0.22)	9.91 (0.24)	6.18 (0.19)	4.50 (0.74)	11.66 (0.12)	7.46 (0.37)	0.004	0.74	42.51 (0.10)	39.34 (0.11)	18.11 (0.34)	17.62 (0.26)	46.89 (0.14)	25.57 (0.51)	0.014	0.75
PIBT [7] (PIBT+BD)	7.79 (0.36)	4.62 (0.10)	5.59 (0.15)	5.12 (0.35)	10.84 (0.08)	6.80 (0.48)	0.004	0.60	32.44 (0.10)	19.39 (2.04)	15.43 (0.34)	15.09 (0.26)	46.51 (0.08)	29.08 (1.26)	0.014	0.61
Follower [13]	10.71 (0.10)	6.68 (0.31)	7.21 (0.10)	5.60 (0.19)	10.00 (0.07)	8.14 (0.10)	0.051	0.70	33.79 (0.06)	16.26 (0.17)	7.32 (0.32)	9.11 (0.29)	41.39 (0.10)	30.03 (0.82)	4.125	0.52
RHCR [6]	10.56 (0.27)	3.38 (0.24)	6.27 (0.08)	5.19 (0.13)	12.14 (0.09)	7.96 (0.12)	4.829	0.66	-	-	-	-	-	-	-	-
SCRIMP [11]	1.01 (0.10)	0.75 (0.06)	1.42 (0.18)	0.53 (0.05)	6.29 (0.55)	2.08 (0.27)	1.982	0.17	-	-	-	-	-	-	-	-

L-PIBT+SG, and L-PIBT+DG.

Then, we compare SILLM with other search- and learning-based solvers in Table 5.2 and Figure 5.4. Regarding the average score, SILLM outperforms all other learning-based and search-based methods. Specifically, on large instances, SILLM almost doubles the score of the previous state-of-the-art learning-based method, Follower [13], and also significantly outperforms the state-of-the-art search-based method, TrafficFlow [8]. Compared to the competition-winning solution, WPPL [1], SILLM performs either closely or much better on all maps and is much faster. With Dynamic Guidance only, SILLM (DG only) also achieves a better score than any



(a) Small Maps (b) Large Maps

Figure 5.4: Comparison of SILLM with other search- and learning-based solvers. These two figures illustrate the average throughput and inference speed of various methods on small and large maps, respectively. Search-based methods are represented by blue circles, while learning-based methods are indicated by yellow triangles. For clarity, L-PIBT with single guidance is excluded from both figures. In the large-map setting, RHCR and SCRIMP are omitted due to their excessive runtime, which makes them impractical to evaluate at scale.

other baseline on large instances. It is worth noting that the main reason our SILLM outperforms WPPL in terms of solution quality is the incorporation of Dynamic Guidance. We have also tried to combine WPPL and Dynamic Guidance directly. But without acceleration from neural policy, its speed is too slow for real-time execution (it takes more than 10 seconds per timestep).

Earlier methods like RHCR [6] and SCRIMP [11] cannot scale to the large scale because of their heavy computation. RHCR and SCRIMP take minutes per timestep on large instances and are too costly to evaluate, failing to meet real-time requirements. RHCR is slow due to its use of a bounded suboptimal, but relatively slow search method, ECBS [19], for each planning window. Its performance also degrades with frequent timeouts in maps like small warehouse.[2] SCRIMP's time consumption stems primarily from its single-step collision avoidance strategy, which requires action resampling and team state value calculation.

[2]We set the time limit to 10 seconds per step for RHCR.

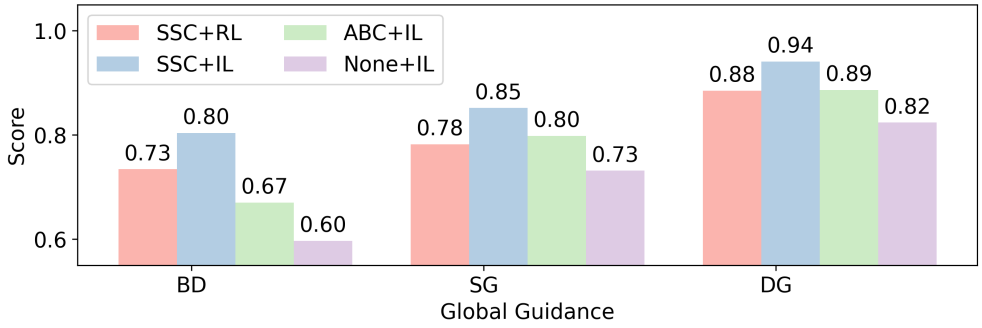


Figure 5.5: Ablation studies on large maps. SSC means our Spatially Sensitive Communication module. ABC means Attention-Based Communication module. None means no communication. RL and IL refer to reinforcement and imitation learning, respectively.

5.1.3 Ablation Study

Figure 5.5 first compares our spatially sensitive communication (SSC) with attention-based communication (ABC)[3] and no communication (None) trained by imitation learning (IL). SSC consistently outperforms ABC and None, indicating the importance of precise spatial reasoning in LMAPF.

We further compare our IL with a reinforcement learning (RL) implementation based on MAPPO [46] and find that our IL outperforms the simple MAPPO.[4] We also tried to apply MAPPO after IL but did not notice any improvement. We believe more sophisticated RL methods are needed, especially those algorithms that can properly distribute team-level rewards to individuals, which is left for future study.

5.2 Learn-to-Follow Benchmark

5.2.1 Experiment Settings

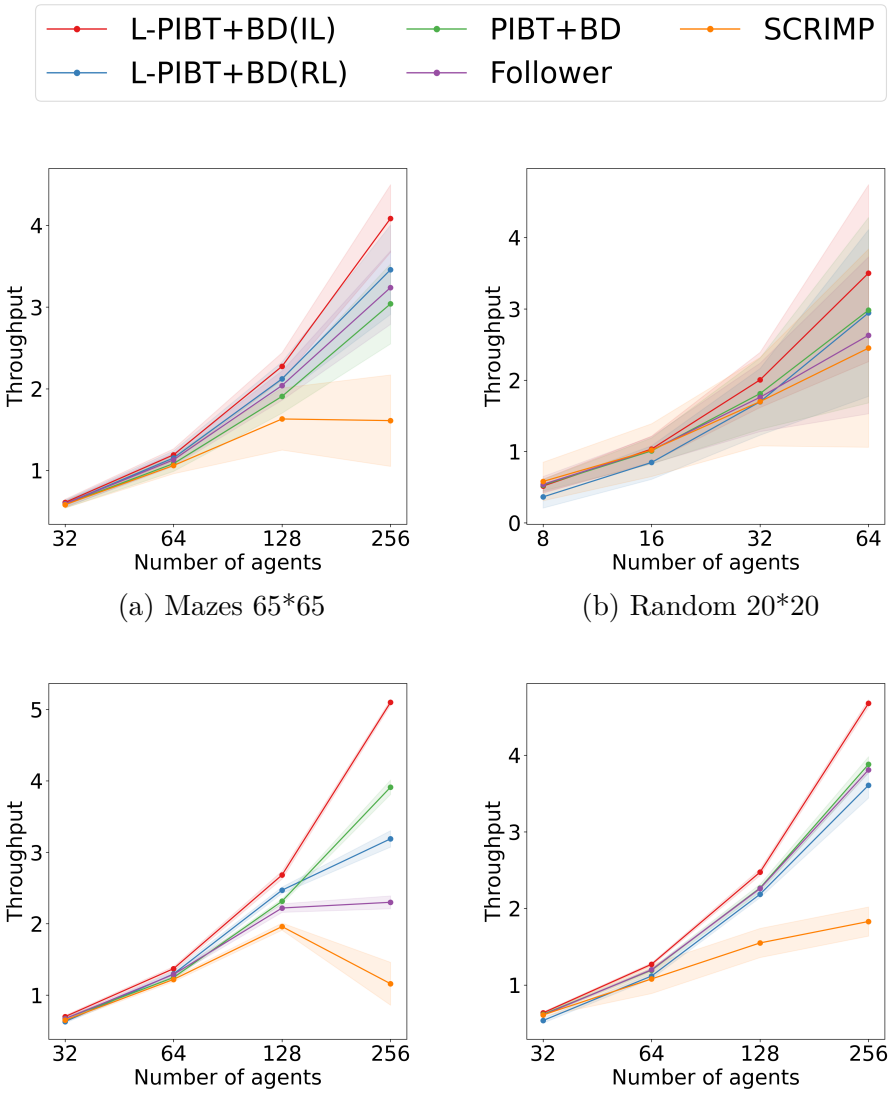
This section compares different decentralized methods on the Learn-to-Follow Benchmark [13] to validate the superiority of our SILLM (L-PIBT) in Figure 5.6, as the experiment setting in the Learn-to-Follow paper is quite different from Section 5.1. For a simple comparison, we only use Backward Dijkstra as global guidance. Notably, Learnable PIBT and Follower are trained on 40 Maze maps and then tested

[3]In our experiment, we use the encoder of a Transformer [45] as the ABC module.
[4]The reward setting is given in the

on 10 different Maze maps and other types of maps. For each pair of map and agent number, we run 10 simulations with different seeds.

5.2.2 Main Results

The experiment results are illustrated in Figure 5.6. Specifically, we compare L-PIBT trained by imitation learning and by reinforcement learning, Follower [13], SCRIMP [11], and PIBT [7]. Our L-PIBT trained by imitation learning consistently performs the best across 4 different types of maps. Notably, the state-of-the-art learning-based solver, Follower, only outperforms PIBT in Maze maps but fails to outperform PIBT in other maps, which means the generalization ability of Follower still needs improvement. In contrast, our L-PIBT trained by imitation learning is much more generalizable. As noted in Section 5.1.3, L-PIBT trained by reinforcement learning is again inferior and requires further study.



(a) Mazes 65*65

(b) Random 20*20

(c) den520d 64*64

(d) Paris_1 64*64

Figure 5.6: Evaluation on Learn-to-Follow Benchmark. The solid lines record the mean throughput of 10 simulations with different seeds, and the shaded areas indicate the 95% confidence intervals.

6 Conclusion and Future Work

6.1 Conclusion

In this thesis, we show how to scale learning-based LMAPF solvers to manage a large number of agents within a short planning time. Specifically, we design a unique communication module, incorporate efficient single-step collision resolution and various types of global guidance, and apply imitation learning from a scalable search-based solver. As a result, our proposed learning-based solver, SILLM, is capable of efficiently planning paths for up to 10,000 agents across various maps. It outperforms the best existing learning- and search-based solvers, showcasing the potential of learning-based approaches for large-scale LMAPF problems.

This work also has the potential for broader impact. The core ideas—such as anytime local optimization, hierarchical system decomposition, and search-learning integration—can be readily applied to other multi-agent systems involving high-dimensional joint action spaces and long-horizon execution.

6.2 Future Work

Several interesting future directions can be further explored to extend this work.

1. **Reinforcement Learning**.Though imitation learning shows great success, it cannot surpass the expert when sufficient computational time is given. We are interested in exploring reinforcement learning (RL) approaches that can find better solutions beyond those provided by the expert. SILLM's core techniques (e.g., architecture, learnable PIBT) can be readily applied in future RL methods.

2. **Co-Optimization of the Neural Policy and Global Guidance**. In our work, we separate the coordination into different levels. Restricted by local observation, neural policy is supposed to handle only local and short-term coordination, while global and long-term coordination is delegated to the heuristic global guidance. Recent works, GGO [44] and Online GGO [47] have proven that we could automatically optimize the global guidance. Therefore, it is promising to explore how to optimize the policy and guidance jointly.

3. **Complex Kinodynamics**. Another important issue is that in all existing learning works, including ours, no complex kinodynamics is considered. As

indicated in the LoRR competition [18], merely adding rotation actions could make the planning much harder and the system more prone to deadlocks and congestion. Additionally, for the smoothness of a robot's movement, we should plan multiple steps rather than a single step for an agent. It is a critical issue to address for real-world multi-robot systems.

A Implementation

A.1 Computation Resources

Our models are trained on servers with 72 vCPU AMD EPYC 9754 128-Core Processor, 4 RTX 4090D (24GB), and 240 GB memory. Training on each map takes around 12 hours.

A.2 SILLM

SILLM is implemented as a mixture of Python and C++ code, based on the public repos of WPPL [1] (https://github.com/DiligentPanda/MAPF-LRR2023) and Trafficflow [8] (https://github.com/nobodyczcz/Guided-PIBT.git). We remove rotation actions in the implementation of WPPL to align with its settings in other baselines. In addition, since we don't need the parallelization during the imitation learning, we always restrict the number of threads to 1 and obtain the WPL in Section 4.1.

A.3 WPPL

In Section 5.1.2, we compare our methods with the winning solution of the League of Robot Runner Competition [18], WPPL [1], the parallelized version of WPL. We still use its public repo with rotation actions removed. Instead of limiting the planning time at each step to 1 second, we limit the iterations of LNS refinement to $60,000$, roughly the total iterations used in our imitation learning. For each simulation, we use 8 parallel threads. More iterations and threads may improve the throughput a little bit, but won't change the results illustrated in Table 5.2.

A.4 MAPPO

In the ablation study (Section 5.1.3), we show that Imitation Learning is better than simple MAPPO-based Reinforcement Learning [46]. Specifically, we use the following reward function.

$$r(v, v') = h(v) - h(v') - 1 \tag{A.1}$$

where h is the heuristic function defined in Section 4.3.3, v and v' are the current and next locations of an agent. Take the Backward Dijkstra heuristics as an example. If v' is 1-step closer to the goal, the reward will be 0. Otherwise, the reward will be a negative penalty. If no other agents act as obstacles, the agent should follow its shortest path given this reward function after learning. Reward design is crucial to the performance of RL and is worth further study.

A.5 Other Baselines

For other baselines in Table 5.2, we directly use the implementation released by the corresponding papers. We re-trained models on our problem instances if they are learning-based methods.

Bibliography

[1] H. Jiang, Y. Zhang, R. Veerapaneni, and J. Li, "Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, pp. 234–242, 2024.

[2] H. Jiang, Y. Wang, R. Veerapaneni, T. H. Duhan, G. A. Sartoretti, and J. Li, "Deploying ten thousand robots: Scalable imitation learning for lifelong multi-agent path finding," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[3] R. Veerapaneni, Q. Wang, K. Ren, A. Jakobsson, J. Li, and M. Likhachev, "Improving learnt local MAPF policies with heuristic search," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 34, pp. 597–606, 2024.

[4] H. Ma, J. Li, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 837–845, 2017.

[5] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, pp. 151–158, 2019.

[6] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, pp. 11272–11281, 2021.

[7] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[8] Z. Chen, D. Harabor, J. Li, and P. J. Stuckey, "Traffic flow optimisation for lifelong multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, pp. 20674–20682, 2024.

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

[10] W. Li, H. Chen, B. Jin, W. Tan, H. Zha, and X. Wang, "Multi-agent path finding with prioritized communication learning," in *International Conference on Robotics and Automation (ICRA)*, pp. 10695–10701, 2022.

[11] Y. Wang, B. Xiang, S. Huang, and G. Sartoretti, "SCRIMP: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9301–9308, 2023.

[12] A. Skrynnik, A. Andreychuk, K. Yakovlev, and A. Panov, "Decentralized monte carlo tree search for partially observable multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, pp. 17531–17540, 2024.

[13] A. Skrynnik, A. Andreychuk, M. Nesterova, K. Yakovlev, and A. Panov, "Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, pp. 17541–17549, 2024.

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

[15] C. Ferner, G. Wagner, and H. Choset, "ODrM* optimal multirobot path planning in low dimensional search spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3854–3859, 2013.

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

[17] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 535–542, 2019.

[18] S.-H. Chan, Z. Chen, T. Guo, H. Zhang, Y. Zhang, D. Harabor, S. Koenig, C. Wu, and J. Yu, "The league of robot runners competition: Goals, designs, and implementation," in *International Conference on Automated Planning and Scheduling (ICAPS)*.

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

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

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

[22] K. Vedder and J. Biswas, "X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs," *Artificial Intelligence*, vol. 291, p. 103417, 2021.

[23] E. Lam and P. Le Bodic, "New valid inequalities in branch-and-cut-and-price for multi-agent path finding," in *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 184–192, 2020.

[24] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, pp. 11655–11662, 2023.

[25] K. Okumura, Y. Tamura, and X. Défago, "Iterative refinement for real-time multi-robot path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9690–9697, 2021.

[26] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 36, pp. 10256–10265, 2022.

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

[28] T. Phan, T. Huang, B. Dilkina, and S. Koenig, "Adaptive anytime multi-agent path finding using bandit-based large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, pp. 17514–17522, 2024.

[29] J. Tan, Y. Luo, J. Li, and H. Ma, "Reevaluation of large neighborhood search for mapf: Findings and opportunities," *arXiv preprint arXiv:2407.09451*, 2025.

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

[31] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021.

[32] Q. Lin and H. Ma, "SACHA: Soft actor-critic with heuristic-based attention for partially observable multi-agent path finding," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 5100–5107, 2023.

[33] H. Tang, F. Berto, and J. Park, "Ensembling prioritized hybrid policies for multi-agent pathfinding," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[34] Z. Ma, Y. Luo, and H. Ma, "Distributed heuristic multi-agent path finding with communication," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8699–8705, 2021.

[35] Z. Ma, Y. Luo, and J. Pan, "Learning selective communication for multi-agent path finding," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1455–1462, 2021.

[36] J. Gao, Y. Li, X. Yang, and M. Tan, "RDE: A hybrid policy framework for multi-agent path finding problem," *arXiv preprint arXiv:2311.01728*, 2023.

[37] Z. Liu, B. Chen, H. Zhou, G. Koushik, M. Hebert, and D. Zhao, "MAPPER: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11748–11754, 2020.

[38] R. Veerapaneni, A. Jakobsson, K. Ren, S. Kim, J. Li, and M. Likhachev, "Work smarter not harder: Simple imitation learning with cs-pibt outperforms large scale imitation learning for mapf," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[39] K. Okumura, "Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1501–1509, 2024.

[40] A. Andreychuk, K. Yakovlev, A. Panov, and A. Skrynnik, "Mapf-gpt: Imitation learning for multi-agent pathfinding at scale," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 39, pp. 23126–23134, 2025.

[41] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 24, pp. 173–178, 2010.

[42] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6932–6939, 2020.

[43] L. Cohen, *Efficient Bounded-Suboptimal Multi-Agent Path Finding and Motion Planning via Improvements to Focal Search*. PhD thesis, University of Southern California, 2020.

[44] Y. Zhang, H. Jiang, V. Bhatt, S. Nikolaidis, and J. Li, "Guidance graph optimization for lifelong multi-agent path finding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 311–320, 2024.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 6000–6010, 2017.

[46] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in neural information processing systems*, vol. 35, pp. 24611–24624, 2022.

[47] H. Zang, Y. Zhang, H. Jiang, Z. Chen, D. Harabor, P. J. Stuckey, and J. Li, "Online guidance graph optimization for lifelong multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 14726–14735, 2025.