# Online Guidance Graph Optimization for Lifelong Multi-Agent Path Finding

**Anonymous submission**

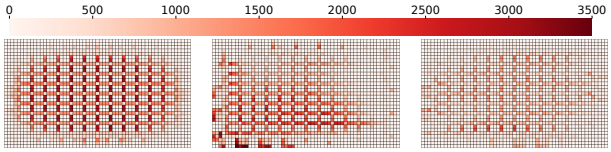

## Abstract

We study the problem of optimizing a guidance policy capable of dynamically guiding the agents for lifelong Multi-Agent Path Finding (LMAPF) based on real-time traffic patterns. MAPF focuses on moving multiple agents from their start to goal locations without collisions. Its lifelong variant, LMAPF, continuously assigns new goals to agents. To solve LMAPF, replan-based algorithms decompose LMAPF into a series of MAPF problems and solve them sequentially, while rule-based algorithms first plan paths for each agent without considering collisions and then use pre-defined rules to resolve collisions on the fly. Although replan-based algorithms yield higher solution quality than rule-based ones, they scale poorly to challenging LMAPF instances with large numbers of agents and limited planning time. On the other hand, rule-based algorithms scale to extremely challenging instances yet have poor solution quality. In this work, we focus on improving the solution quality of PIBT, the state-of-the-art rule-based LMAPF algorithm, by optimizing a policy to generate adaptive guidance. We design two pipelines to incorporate guidance in PIBT in two different ways. We demonstrate the superiority of the optimized policy over both static guidance and human-designed policies. Additionally, we explore scenarios where task distribution changes over time, a challenging yet common situation in real-world applications that is rarely explored in the literature.


## 1 Introduction

We study the problem of optimizing a *guidance policy* capable of dynamically updating guidance graphs with optimized edge weights to guide the agents' movement and improve throughput in lifelong Multi-Agent Path Finding (MAPF). MAPF (Stern et al. 2019) aims to compute collision-free paths for agents from their starts to goals. The lifelong variant of MAPF (LMAPF) constantly assigns new goals to agents upon arriving at their current goals and aims to maximize *throughput*, namely the number of goals reached per timestep. LMAPF has wide applications in automated warehouses (Li et al. 2021b; Zhang et al. 2023b,a) and game character control (Ma et al. 2017b; Jansen and Sturtevant 2008).

Many algorithms have been proposed to solve LMAPF. The *replan-based* algorithms (Ma et al. 2017a; Li et al. 2020; Kou et al. 2020; Damani et al. 2021) decompose LMAPF into a series of MAPF problems and solve them sequentially. Although replan-based algorithms possess high solution quality, they scale poorly to real-world scenarios with large maps, large numbers of agents, and limited planning time. For example, prior works (Zhang et al. 2023a,b) have shown that the throughput of RHCR (Li et al. 2021b), a state-of-the-art replan-based algorithm, degrades to almost 0 with more than 200 agents in a $33 \times 36$ small warehouse map because of unrecoverable congestion. In comparison, real-world Amazon fulfillment centers are reported to have more than 4,000 robots (Brown 2022). Other works (Yu and Wolf 2023; Li et al. 2021b) motivated by Amazon sortation centers use maps of size up to $179 \times 69$ and 1,000 agents.




(a) No guidance. (b) Offline guidance. (c) Online guidance.

Figure 1: Comparison of no guidance, offline guidance, and our online guidance with a simulation of 5,000 timesteps with 600 agents in a warehouse map of size $33 \times 57$ with 1,091 non-obstacle cells. The average and standard deviation of throughput over 10 simulations for no guidance, offline guidance, and online guidance are $3.18 \pm 0.04$, $6.42 \pm 0.09$, and $8.66 \pm 0.04$, respectively. The heatmaps show the number of times the agents take wait action in each cell, approximating the level of congestion. Our online guidance results in the most balanced traffic and thus less congestion and higher throughput.

Meanwhile, the *rule-based* algorithms (Wang and Botea 2008; Okumura et al. 2019; Yu and Wolf 2023) compute a shortest distance heuristic for each agent without considering the collisions and use pre-defined rules to resolve the collisions. Rule-based algorithms run very fast and scale to extremely large maps and large numbers of agents. Jiang et al. (2024) show that with a planning time limit of 1 second per timestep, PIBT (Okumura et al. 2019), a state-of-the-art rule-based algorithm, can handle maps as large as 54,230 vertices and as many as 10,000 agents.

However, rule-based algorithms have no guarantee on the solution quality. Zhang et al. (2024) have shown that with 150 agents, the throughput of RHCR is 24.2% better than

PIBT in a 33 $\times$ 36 warehouse map. To improve the solution quality of rule-based algorithms, prior works (Zhang et al. 2024; Chen et al. 2024; Yu and Wolf 2023; Li and Sun 2023) have explored providing *guidance* to the agents such that they automatically avoid congested areas, thereby improving throughput. Most previous works adopt a static *offline* guidance, with the *guidance graph* (Zhang et al. 2024) providing a versatile and state-of-the-art representation of guidance. The guidance graph is a directed weighted graph with edge weights specifying the cost of moving and waiting for agents. Upon optimizing a guidance graph for PIBT, the throughput gap between PIBT and RHCR is reduced to less than 4.2% (Zhang et al. 2024). Nevertheless, the offline nature of the guidance graph assumes that the area of congestion does not change, an assumption that does not always hold in the real world. For example, in automated warehouses, task distribution can shift because of sudden changes in the distribution of orders.

Therefore, instead of optimizing an offline guidance graph that provides static guidance, we optimize an *online guidance policy* capable of adapting a dynamic guidance graph over time based on real-time traffic information to improve the throughput of PIBT-based LMAPF algorithms, the state-of-the-art in rule-based LMAPF. One previous work (Chen et al. 2024) has explored using online guidance in PIBT, but their guidance policy is a handcrafted equation. Instead, we consider a parameterized model that can be optimized based on various task distributions and maps. Figure 1 shows the traffic congestion resulting from different guidance.

To incorporate the online guidance policy into rule-based LMAPF algorithms, we design two pipelines. One directly use the guidance policy to dynamically generate guidance graphs based on real-time traffic information and replaces uniform edge weights with the generated guidance graphs. The other uses the dynamically generated guidance graphs to adaptively plan better guide paths and move the agents along the guide paths while resolving collisions. To optimize the guidance policy, we follow Zhang et al. (2024) in using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (Hansen 2016), a single-objective black-box optimization algorithm, to optimize the policy.

We make the following contributions: (1) we generalize the offline, static guidance graph to an online, dynamic guidance policy capable of updating the guidance graph based on real-time traffic information, (2) we propose two methods to incorporate guidance policy into PIBT, a state-of-the-art rule-based LMAPF algorithm, and (3) we test our online guidance policy in LMAPF simulations with both uniform and dynamic task distributions and demonstrate the advantage over offline guidance and human-designed online guidance in various maps and algorithms.

# 2 Preliminary

## 2.1 Lifelong MAPF

**Definition 1 (Lifelong MAPF (LMAPF))** *Given a graph $G(V, E)$, a set of agents and their corresponding start and goal locations, LMAPF aims to move all agents from their start to their goal locations while avoiding collisions. Furthermore, agents are constantly assigned new goals upon reaching their current goals. At each timestep, agents may either move to an adjacent vertex or remain stationary. Collisions happen when two agents are at the same vertex or swap locations in the same timestep. The objective is to maximize throughput, defined as the average number of goals reached per timestep.*

## 2.2 Guidance Graph and Guidance Policy

Our definition of guidance graphs follows the previous work (Zhang et al. 2024).

**Definition 2 (Guidance Graph)** *Given a Lifelong MAPF problem defined on a graph $G(V, E)$, a guidance graph is a directed weighted graph $G_g(V_g, E_g, \boldsymbol{\omega})$, where $V_g = V$, and $E_g = E \cup \{(v, v) \mid v \in V\}$ encodes the action costs in every single vertex, including moving and waiting, for all agents. The action costs are represented collectively as the edge weights $\boldsymbol{\omega} \in \mathbb{R}_{>0}^{|E_g|}$.*

To plan with a guidance graph in MAPF, we change the objective from searching for the paths of minimal sum-of-costs, which is the sum of path lengths of all agents, in $G$ to searching for the path of minimal sum of action costs in $G_g$. The similar idea applies in LMAPF (introduced in Section 4.1).

**Definition 3 (Guidance Policy)** *Given a guidance graph $G(V_g, E_g, \boldsymbol{\omega})$, a guidance policy is a function $\pi_{\boldsymbol{\theta}} : \boldsymbol{O} \to \mathbb{R}_{>0}^{|E_g|}$ that computes the updated edge weights $\boldsymbol{\omega'} \in \mathbb{R}_{>0}^{|E_g|}$ given the observation $\boldsymbol{o} \in \boldsymbol{O}$ collected in a LMAPF simulation. The policy $\pi_{\boldsymbol{\theta}}$ is parameterized by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ is the space of all parameters.*

The guidance policy dynamically updates the guidance graph during LMAPF simulation based on real-time observations. Depending on the LMAPF algorithm, the observation consists of one or more past traffic patterns, the current distribution of tasks, and future planned paths. We discuss the choice of observation in Section 4.

# 3 Related Work

## 3.1 Lifelong MAPF Algorithms

LMAPF algorithms mainly include **replan-based** algorithms (Li et al. 2021b; Liu et al. 2019; Li et al. 2021b) and **rule-based** algorithms (Wang and Botea 2008; Okumura et al. 2019; Yu and Wolf 2023). Replan-based algorithms decompose the LMAPF problem into a sequence of MAPF problems and rely on MAPF algorithms to solve them. RHCR (Li et al. 2021b) is the state-of-the-art of this category. For every $h$ timesteps, RHCR replans collision-free paths of all agents within a time window of $w$ timesteps ($w \geq h$), ignoring collisions beyond it. While RHCR has state-of-the-art throughput with a small number of agents in small maps (Li et al. 2021b), it scales poorly to instances with large numbers of agents and limited planning time. Prior works have shown that the throughput of RHCR drops to almost zero with more than 200 agents in a 33 $\times$

36 small warehouse (Zhang et al. 2023a,b) with a per-5-timestep planning time limit of 60 seconds. Even with optimized guidance graph, Zhang et al. (2024) shows that RHCR does not scale to more than 250 agents in the same map.

Rule-based algorithms, on the other hand, run much faster than replan-based ones at the expense of solution quality. Rule-based algorithms compute a shortest distance heuristic for each agent without considering the collisions and leverage pre-defined rules to move the agents while resolving collisions. PIBT (Okumura et al. 2019) is the state-of-the-art of this category. Chen et al. (2024) shows that the planning time of PIBT per timestep is over 150 times faster than RHCR. Jiang et al. (2024) further shows that PIBT is the only existing algorithm that can handle a limited planning time of $1$ second for up to $10,000$ agents. Therefore, we focus on improving the throughput of PIBT using guidance.

**PIBT.** PIBT leverages a single-timestep rule to move agents. At each timestep, PIBT assigns a priority to each agent and ranks each agent's actions based on the shortest distance from its next location to its goal, with a preference for shorter distances. By default, an agent always tries to follow its shortest path if no higher-priority agents act as obstacles. Otherwise, a lower-priority agent needs to avoid collisions with higher-priority agents by taking the next preferred action. If a lower-priority agent fails to avoid a collision, PIBT applies a backtracking mechanism that forces higher-priority agents to take their next preferred actions and retries all the movement until all agents can take a collision-free action.

### 3.2 Guidance in MAPF

While term *guidance* has no explicit definition in MAPF, the general idea of using guidance to guide the agents' movement is widely explored. Prior works fall into two categories, namely *offline* guidance that provides static guidance to the agents, and *online* guidance in which the guidance changes over time based on real-time traffic information.

**Offline Guidance.** A static offline guidance usually leverages edge directions or edge costs to encourage the agents to move along certain edges. The pre-defined directions and costs are generated in advance and are not updated during the execution of the algorithm. Wang and Botea (2008) leverages unidirectional edges in a map to force the agents to move in one direction, eliminating the head-to-head collisions. The idea of highway (Cohen, Uras, and Koenig 2015; Cohen et al. 2016; Cohen 2020; Li and Sun 2023) predefines a subset of edges in the graph as highway-edges and encourages the agents to move along those edges.

The state-of-the-art offline guidance is the guidance graph (Zhang et al. 2024). It unifies the representation of guidance by using a directed weighted graph. The edge weights of the guidance graph define the cost of moving and waiting for all agents. The edge weights of the guidance graphs are optimized automatically either directly using a single-objective derivative-free optimizer CMA-ES (Hansen 2016) or indirectly with Parameterized Iterative Update (PIU), an iterative update procedure. Starting with a uniform guidance graph with all edge weights being 1, PIU runs the LMAPF simulator to collect the traffic information, which is used by a parameterized update model to update the guidance graph. The updated model is then optimized by CMA-ES for PIU to generate high-throughput guidance graphs.

**Online Guidance.** Online guidance is capable of adapt the guidance based on real-time traffic information during the execution of the LMAPF algorithm. Jansen and Sturtevant (2008) assigns a direction vector to each location in the map and encourages the agents to move along the direction vector by setting the movement cost of that location to be inversely related to the dot product of the direction vector and the edge cost. The direction vectors are computed from the past traffic information by using a handcrafted equation. Han and Yu (2022) computes a handcrafted temporal heuristic function to estimate the movement cost, guiding the agents' movement. Similarly, Chen et al. (2024) collects planned paths of all agents and uses a handcrafted equation to compute the movement cost. Yu and Wolf (2023) relies on a trained data-driven model to predict the movement delays of the agents and uses the delays of the movement costs.

### 3.3 Challenge of Online Guidance

Since online guidance is capable of adapt the guidance on-the-fly based on real-time traffic information, it should be able to provide better guidance for agents when the traffic pattern shifts abruptly. For example, in automated warehouses in which robots are used to transport packages from one location to another, the distribution of packages might change over time, resulting in different traffic patterns. In this case, it is necessary to provide online guidance to agents.

However, it is non-trivial to incorporate online guidance into LMAPF algorithms because of the computational overhead of computing the heuristic values. And it is unavoidable because PIBT requires the cost-to-go heuristic to plan agents' actions. With an offline guidance graph, the heuristic values only need to be computed once at the beginning. With an online guidance graph, however, the heuristic values of the entire map need to be updated once the guidance graph is updated. To tackle this issue, a recent work on using online guidance in PIBT (Chen et al. 2024) proposes a variant of PIBT, referred to as Guided-PIBT (GPIBT).

**GPIBT.** GPIBT first plans a guide path for each agent that minimizes a handcrafted congestion cost equation, which takes other agents' guide path usages into account. It then moves agents to their goals following the guide paths and resolves collisions using PIBT. Since the heuristics that guide agents to follow guide paths only need to be updated while planning the guide paths, GPIBT reduces the computational overhead of heuristic computation. However, since the guide paths are not updated until the agents reach their current goals, GPIBT is less flexible than PIBT in terms of the paths that the agents eventually take to reach the goals.

Given the state-of-the-art performance of PIBT and GPIBT, we focus on incorporating automatically optimized online guidance in them, thereby improving their performance in our work.

## 4 Approach

Figure 2 shows the overview of our methods. The outer loop (blue) shows the guidance policy optimization, while the in-

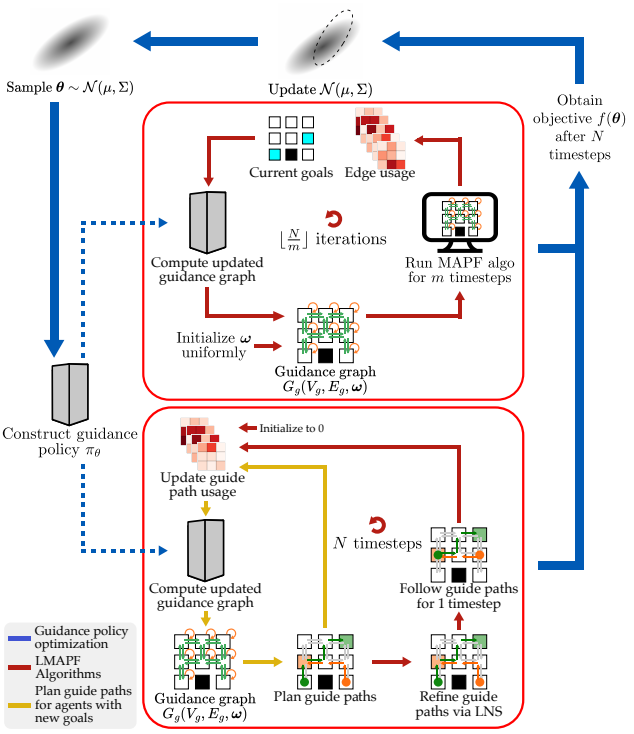



Figure 2: Overview of using guidance policy to improve PIBT-based algorithms. We incorporate guidance policies by using direct planning (above) and guide paths (below).

ner loops (red) are two ways we incorporate the guidance policy in LMAPF algorithms. We first discuss how we incorporate guidance graphs and guidance policies into PIBT-based LMAPF algorithms in Section 4.1. We then discuss the guidance policy optimization approach in Section 4.2. We will illustrate the inner loop and outer loop in detail in Section 4.1 and Section 4.2, respectively.

## 4.1 Incorporating Guidance Policy

While the guidance graph is a versatile representation of guidance, different LMAPF algorithms move agents to the goals differently. In addition, while the guidance graph is updated online by the guidance policy, we need to account for the additional computational cost associated with the update mechanism. Here, we discuss two ways of using guidance graphs and guidance policy in LMAPF algorithms.

**Direct Planning** Most LMAPF algorithms directly plan on the guidance graph by minimizing the sum of action costs encoded in the guidance graph. PIBT (Okumura et al. 2019) falls into this category. The red loop in the upper red box of Figure 2 shows the overview of this method. Starting from a uniform guidance graph with all weights being 1, we run the LMAPF algorithm for $m$ timesteps. After $m$ timesteps, we obtain the edge usage of the past $m$ timesteps and the currently assigned goals of the agents. The edge usage reflects the recent traffic information, and the current goals project the future possible congestion locations. Upon obtaining the edge usage and current goals, we use the guidance policy to update the guidance graph, starting a new iteration.

Since we end the LMAPF simulation after $N$ timesteps, we run the above procedure for $\lfloor \frac{N}{m} \rfloor$ iterations. By choosing $m$, we control the trade-off between the adaptability of the online updated guidance graph and additional computational costs associated with the online update mechanism. This is because directly planning on the guidance graph usually requires the cost-to-go heuristic, defined as the shortest path length between every pair of nodes on the guidance graph. The heuristic needs to be updated every time the guidance graph is updated, thereby making the LMAPF algorithm more computationally expensive.

**Guide-Path Planning** Given the heavy computational overhead of repeatedly computing the heuristic, another way to use guidance is applying the guide paths. The red and yellow loop in the lower red box of Figure 2 shows the overview. To plan the guide paths, we maintain the guide-path edge usage, which is the number of times each edge is used in the current guide paths of all agents. Starting from zero guide-path edge usage, we start planning the guide paths (yellow loop). We plan guide paths for all agents that are assigned new goals in each timestep. For each agent, we use the guidance policy to generate the guidance graph based on the current guide-path edge usages. We then plan the guide path on the generated guidance graph. Inspired by Chen et al. (2024), the input for the guidance policy is the guide-paths edge usage. It combines the information of past and future traffic information because guide-paths are replanned when an agent reaches its current goal; otherwise, it will keep following its current guide-path. Finally, we update the guide-path edge usage using the planned guide path and proceed to the next agent. After all agents have guide paths, we start moving the agents (red loop). We follow the previous work (Chen et al. 2024) to refine the guide paths using Large Neighborhood Search (LNS) (Li et al. 2021a) and then move the agents for one timestep. In case of collision, a pre-defined rule is applied to resolve the collisions while encouraging the agents to follow the guide paths. We repeat the above procedure for $N$ timesteps.

Compared to direct planning, guide-path planning minimizes the effort on heuristic update because an agent's guide heuristics are recomputed only when its guide path changes, which ignores other agents' state changes or guidance graph changes.

## 4.2 Guidance Policy Optimization

We aim to optimize the parameter $\boldsymbol{\theta}$ of the guidance policy to maximize the throughput given by the LMAPF simulators. Following (Zhang et al. 2024), we use CMA-ES (Hansen 2016), a single-objective derivative-free optimization algorithm, to optimize $\boldsymbol{\theta}$. The blue loop in Figure 2 shows the overview of using CMA-ES to optimize the guidance policy. CMA-ES maintains a multi-variate Gaussian distribution to represent the distribution of solutions. Starting with a standard normal Gaussian, CMA-ES samples a batch of $b$ parameter vectors $\boldsymbol{\theta}$, forming $b$ guidance policies. It then evaluates these guidance policies by running the given LMAPF simulator $N_e$ times to compute the average throughput. Finally, based on the evaluated guidance policies, CMA-ES

updates the parameters of the Gaussian towards the high-throughput region of the search space. CMA-ES repeats the above procedure until the number of guidance policy evaluations reaches $N_{eval}$.

# 5 Experiments and Analysis

## 5.1 Experiment Setup

We will conduct experiments comparing online and offline guidance (Section 5.2), optimized guidance policies versus human-designed guidance policies (Section 5.2), and emphasize the advantage of online guidance with dynamic task distribution (Section 5.2). Additionally, we demonstrate that our approach mitigates deadlock issues of PIBT-based algorithms (Section 5.2).

**Maps.** We conduct experiments on 4 maps: (1) *sortation-33-57*, (2) *warehouse-33-57*, (3) *empty-32-32*, and (4) *random-32-32*, shown on top of Figure 3. The first two have regular patterns and are used extensively to test MAPF algorithms in automated warehouses settings. The latter two are selected from the MAPF benchmark (Stern et al. 2019).

**Task Generation.** As shown on the top of Figure 3, the *sortation* and *warehouse* maps have workstations (pink) and endpoints (blue). Each agent's tasks constantly alternate between the workstations and the endpoints, simulating the warehousing scenario in which robots pick up items from the workstations and drop them off in chutes or shelves (black) reachable from the endpoints in the middle. For *empty* and *random* maps, tasks are sampled from empty spaces (white).

**Task Distribution.** We consider both static and dynamic task distributions. Static task distribution samples tasks uniformly, the most common setting in previous works (Chen et al. 2024; Zhang et al. 2023b,a). For the dynamic task distribution, tasks are sampled from the Gaussian or multi-mode Gaussian distribution, where the Gaussian centers change over time. Concretely, in *sortation* and *warehouse* maps, tasks on endpoints are sampled from a Gaussian distribution, and tasks on workstations are sampled uniformly. For *empty* and *random* maps, tasks are sampled from a multi-mode Gaussian distribution with $K$ Gaussian centers.

**Algorithm Comparison.** We compare 6 different variant approaches, each with a different LMAPF algorithm or guidance approach.

**on+PIBT** our proposed online guidance policy applied to PIBT. We optimize an online guidance policy, periodically call it to update guidance graph, and directly plan on the guidance graph.

**off+PIBT** PIBT with an offline guidance graph, optimized using PIU (Zhang et al. 2024).

**on+GPIBT** our proposed online guidance policy applied to GPIBT. We optimize an online guidance policy and call it when each agent plans its guide-path.

**[p-on]+GPIBT** GPIBT with periodically updated guidance graph. We use the same pipeline as on+PIBT. The inputs of online policy are past traffic and current task locations instead of guide-path information. We present the results of this algorithm to show the advantages of on+GPIBT.

**off+GPIBT** GPIBT with an offline guidance graph, optimized using PIU (Zhang et al. 2024). The guide-paths are generated according to the optimized offline guidance graph.

**hm+GPIBT** GPIBT with a human-designed equation as the guidance policy and generate the guide-paths (Chen et al. 2024). Here we use the SUM_OVC function (Chen et al. 2024), where edge weights are computed by the sum of vertex usages and the product of the head-on-head edge usages.

We run all LMAPF algorithms for $N = 1,000$ timesteps. In on+PIBT and [p-on]+GPIBT, we update the guidance graph for every $m = 20$ timesteps.

## 5.2 Experiment Results

Figure 3 shows the results of comparing the previously mentioned algorithms. To show the generalizability of our approach, we optimize the guidance policies and guidance graphs with the numbers of agents indicated by the black vertical lines and evaluate them with various numbers of agents. We highlight several key findings below.

**Online vs. Offline Guidance.** We first compare our online guidance policy with the offline guidance graph (Zhang et al. 2024). Therefore, we focus on comparisons between on+PIBT with off+PIBT and on+GPIBT with off+GPIBT. As shown in Figure 3, under most settings, on+PIBT and on+GPIBT generally match or outperform off+PIBT and off+PIBT throughout different numbers of agents, respectively. In the *random* map, however, off+GPIBT outperforms on+GPIBT because of deadlocks. Also, the throughput-agents curve is not concave like other maps. We discuss this further later. In addition, in the *sortation* map with static uniform task distribution, on+PIBT is (slightly) worse than off+PIBT because congested locations do not change a lot over time. Therefore, a well-optimized offline guidance graph can alleviate congestion well enough. Visualization results can be found in Figure 4. Besides, there exist outliers on some number of agents. That is because our model is not directly optimized for these cases.

**Online Guidance with Dynamic Tasks.** With dynamic task distribution, we expect that dynamic task distribution gives more advantages to online guidance over offline guidance because the guidance policy can dynamically change the guidance graph based on real-time traffic information, which depends on the task distribution. To demonstrate the advantage of online guidance, we use the ratio of improvement in throughput from offline guidance as the statistic.

Figure 5 shows the online policy improvement ratio. We show the numerical results in Appendix C.1. Clearly, online guidance policy is more advantageous than offline guidance graph in PIBT. However, the improvement ratio of GPIBT1 comparing on+GPIBT and off+GPIBT does not consistently increase when moving from static task distribution to dynamic. That is because the difference between on+GPIBT and off+GPIBT is not just online and offline, but also the guide-paths mechanism that explicitly uses other agents' future movement information. Therefore, to illustrate the improvement caused by the online mechanism only, we fur-

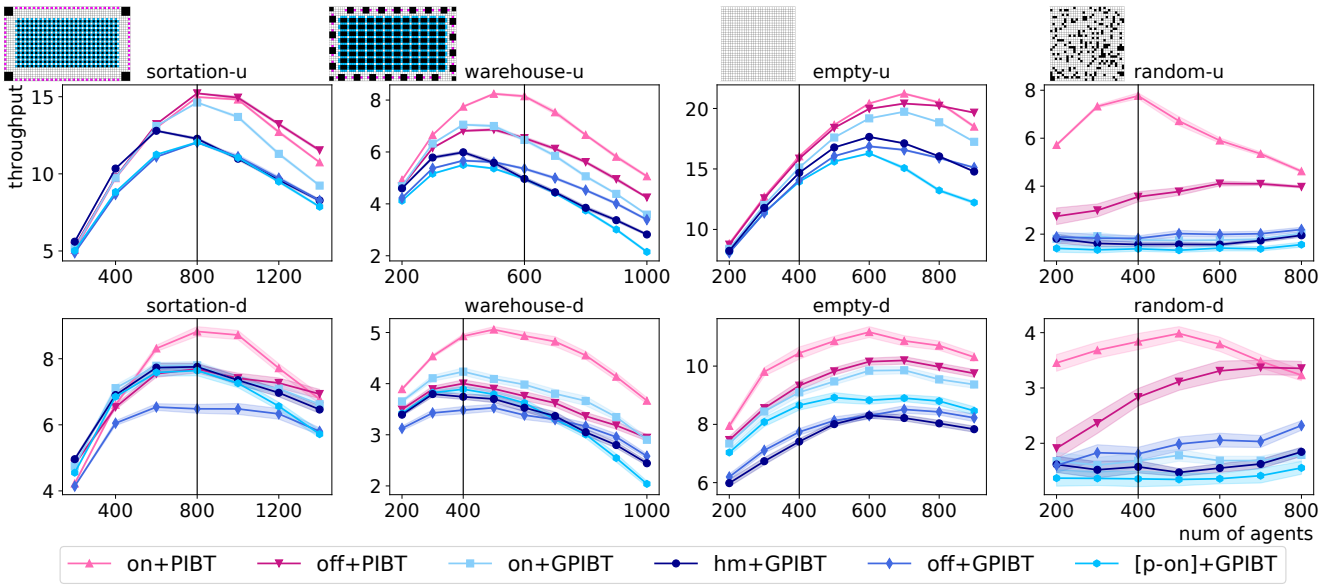



Figure 3: Throughput with different number of agents. The black vertical lines show the number of agents that are used to optimize the guidance policies. The solid line shows the average throughput over 50 LMAPF simulations and the shaded areas denote the 95% confidence interval. "u" and "d" stand for uniform and dynamic task distribution, respectively.

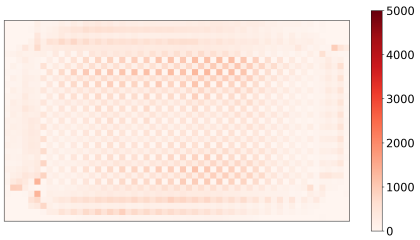


Figure 4: off+PIBT wait action on the *sortation* map with 800 agents under the uniform task distribution for 5000 timesteps

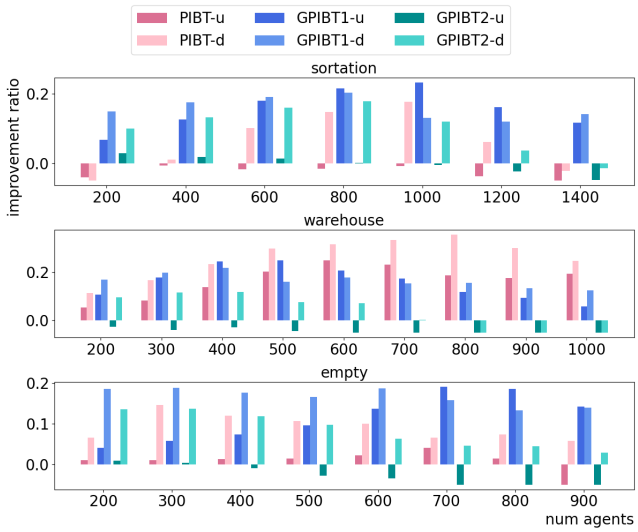



Figure 5: Improvement ratio on uniform and dynamic task distribution. *u* and *d* indicate the uniform task distribution and dynamic task distribution, respectively. PIBT compares between on+PIBT and off+PIBT. GPIBT1 compares on+GPIBT and off+GPIBT, while GPIBT2 compares [p-on]+GPIBT and off+GPIBT.

ther investigate the performance of [p-on]+GPIBT. It is clear that the ratio of GPIBT2, which compares [p-on]+GPIBT and off+GPIBT, consistently increases while moving from static to dynamic task distribution. In addition, the comparison between on+GPIBT with [p-on]+GPIBT in Figure 3 also shows the effectiveness of using guide-path information.

**Online Guidance and Deadlocks.** Although PIBT is proven deadlock-free (Okumura et al. 2019), this requires maps that do not have dead-ends. While the *sortation*, *warehouse*, and *empty* maps clearly satisfy this property, the *random* map does not. That is why the *throughput-agents* curve in Figure 3 seems quite different from the other 3 maps.

Figure 6 shows the number of finished tasks at each timestep on *random* with 400 agents, where the shaded area is the raw data, and the solid line is smoothed by averaging over 5 timesteps. It is clear that the number of finished tasks drops to zero for off+PIBT, which signals that a deadlock occurs. However, our proposed on+PIBT method can escape from deadlocks, maintaining a consistant flow of finished tasks, because the guidance graph and the heuristics are updated periodically. If a deadlock occurs, the past traffic information will direct our online policy to generate a new guidance graph, leading the agents to escape from current deadlocks.

However, on+GPIBT cannot resolve deadlocks, because

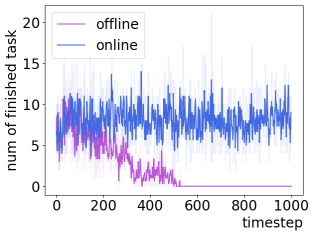
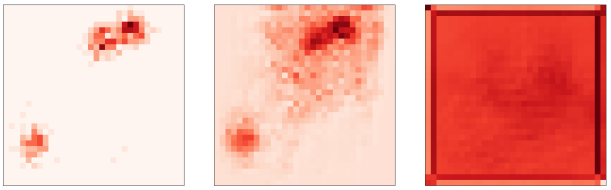



Figure 6: Finished tasks at each step on the *random* map



(a) task (b) on+PIBT (c) off+PIBT

Figure 7: Online and offline guidance given task locations on the *empty* map

it updates the guidance graph only when an agent reaches its current goal. However, when deadlocks occur, no agent is able to reach its goal, resulting in no agents being able to move. Besides, the guide-paths edge usage, the input of the guidance policy, does not reflect the existence of deadlock.

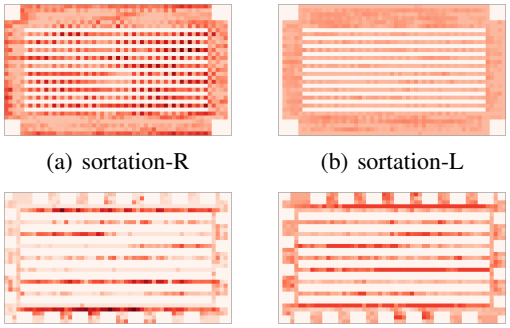
**Optimized vs. Handcrafted Online Guidance.** We also compare our optimized online guidance policy to a human-designed guidance policy (Chen et al. 2024). As shown in Figure 3, the comparison between on+GPIBT with hm+GPIBT shows the advantage of our optimized guidance policy over the human-designed guidance policy on GPIBT algorithm. We are unaware of any human-designed online guidance policy for PIBT.

### 5.3 Guidance Policy Visualization

In this section, we expand on earlier hypotheses and explain some results mentioned before by showing the guidance graph given by on+PIBT and guide-paths by on+GPIBT.

**Online guidance can capture congestion locations.** The online guidance policies outperform the offline guidance graphs for several reasons. First, since the guidance policy is capable of dynamically generating guidance graphs based on traffic information, a well-optimized guidance policy can provide as least as much guidance to the agents as an offline guidance graph. For example, if the optimal guidance is an offline guidance graph, then the guidance policy can generate such graph and make no changes to it afterwards. Second, the online policy is capable of getting more real-time information from the downstream algorithms. For instance, the online policy can capture different congestion locations and alleviate them.

Figure 7 shows the online and offline wait costs given task locations on the *empty* map with the dynamic task distribution. The tasks are sampled from a 3-mode Gaussian distribution. The darker the color is, the larger the costs are.



(a) sortation-R (b) sortation-L

(c) warehouse-R (d) warehouse-L

Figure 8: Visualization of the edge weights of online guidance graph for *sortation* and *warehouse* maps. R, L for right and left edge, respectively.

Our online guidance can capture the task information and the congestion location it indicates, while the offline guidance graph has no correlation with the task distribution at all.

**The structure of the map affects the improvement ratio of online guidance policy.** Although the *sortation* and *warehouse* maps look alike, they have different properties and thus lead to different performances in throughput. In *warehouse*, the white space between workstations (pink) and endpoints (blue) is more narrow, and the corridors among endpoints are longer. The former makes detours in the white space less useful, and the latter decreases the number of path choices when agents are in the center of the map. Therefore, when to guide agents to enter the center area is important in the *warehouse* map. Under the uniform task distribution, we can clearly see the alternative preferred direction on *warehouse* map in the generated online guidance, but can hardly capture it in *sortation* map. (See Figure 8). Therefore, we can see why, under the uniform task distribution setting, online guidance leads to dramatic improvement on the *warehouse* map but not on the *sortation* map.

We show additional visualizations of guidance graphs in *sortation* map and visualization of guide paths in Appendix C.2.

## 6 Conclusion

Lifelong MAPF is a challenging problem of immense practical importance. Once we have a large number of agents, all practical approaches to the problem must overcome congestion problems. In this paper, we show how we can use dynamically generated guidance graphs that learn from traffic patterns to improve throughput on congested maps. This leads to state-of-the-art throughput results.

## References

Brown, A. S. 2022. How Amazon Robots Navigate Congestion. https://www.amazon.science/latest-news/how-amazon-robots-navigate-congestion. Accessed: 2023-05-09.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. 2024. Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 20674–20682.

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

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 2–8.

Cohen, L.; Uras, T.; Kumar, T. K. S.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3067–3074.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. PRIMAL$_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

Han, S. D.; and Yu, J. 2022. Optimizing Space Utilization for More Effective Multi-Robot Path Planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 10709–10715.

Hansen, N. 2016. The CMA Evolution Strategy: A Tutorial. *ArXiv*, abs/1604.00772.

Jansen, M. R.; and Sturtevant, N. R. 2008. Direction Maps for Cooperative Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 185–190.

Jiang, H.; Zhang, Y.; Veerapaneni, R.; and Li, J. 2024. Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 234–242.

Kou, N. M.; Peng, C.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 9925–9932.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, 4127–4135.

Li, J.; Sun, K.; Ma, H.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2020. Moving Agents in Formation in Congested Environments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 726–734.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021b. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11272–11281.

Li, M.-F.; and Sun, M. 2023. The Study of Highway for Lifelong Multi-Agent Path Finding. *ArXiv*, 2304.04217.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1152–1160.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017a. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 837–845.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2017b. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 270–272.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2019. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 535–542.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 151–159.

Tjanaka, B.; Fontaine, M. C.; Lee, D. H.; Zhang, Y.; Balam, N. R.; Dennler, N.; Garlanka, S. S.; Klapsis, N. D.; and Nikolaidis, S. 2023. pyribs: A Bare-Bones Python Library for Quality Diversity Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 220–229.

Wang, K. C.; and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 380–387.

Yu, G.; and Wolf, M. 2023. Congestion prediction for large fleets of mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 7642–7649.

Zhang, Y.; Fontaine, M. C.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2023a. Arbitrarily Scalable Environment Generators via Neural Cellular Automata. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 57212–57225.

Zhang, Y.; Fontaine, M. C.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2023b. Multi-Robot Coordination and Layout Design for Automated Warehousing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 5503–5511.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 311–320.

# A Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced **(yes)**
- Clearly delineates statements that are opinions, hypotheses, and speculations from objective facts and results (**yes**)
- Provides well-marked pedagogical references for less-familiar readers to gain the background necessary to replicate the paper (**yes**)

Does this paper make theoretical contributions? (**no**)
If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no)
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no)
- Proofs of all novel claims are included. (yes/partial/no)
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no)
- Appropriate citations to theoretical tools used are given. (yes/partial/no)
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA)
- All experimental codes used to eliminate or disprove claims are included. (yes/no/NA)

Does this paper rely on one or more datasets? (**no**)
If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA)
- All novel datasets introduced in this paper are included in a data appendix. (yes/partial/no/NA)
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (yes/partial/no/NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations. (yes/no/NA)
- All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available. (yes/partial/no/NA)
- All datasets that are not publicly available are described in detail, with an explanation of why publicly available alternatives are not scientifically satisfying. (yes/partial/no/NA)

Does this paper include computational experiments? (**yes**)
If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. (**no**). No data is required in this paper.
- All source code required for conducting and analyzing the experiments is included in a code appendix. (**yes**)
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (**yes**)
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (**yes**)
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (**yes**)
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (**yes**)
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (**yes**)
- This paper states the number of algorithm runs used to compute each reported result. (**yes**)
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (**yes**)
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed rank). (**yes**)
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (**yes**)
- This paper states the number and range of values tried per (hyper-) parameter during the development of the paper, along with the criterion used for selecting the final parameter setting. (**yes**)

| map | distribution type | algorithm | $N_{eval}$ | $b$ | $N_e$ |
|---|---|---|---|---|---|
| *sortation* | uniform | on+pibt | 50,000 | 50 | 3 |
| | | off+pibt | 50,000 | 50 | 3 |
| | | on+gpibt | 10,000 | 100 | 2 |
| | | [p-on]+gpibt | 10,000 | 100 | 2 |
| | | off+gpibt | 10,000 | 100 | 2 |
| | dynamic | on+pibt | 12,500 | 50 | 3 |
| | | off+pibt | 12,500 | 50 | 3 |
| | | on+gpibt | 30,000 | 100 | 5 |
| | | [p-on]+gpibt | 50,000 | 100 | 5 |
| | | off+gpibt | 30,000 | 100 | 5 |
| *warehouse* | uniform | on+pibt | 25,000 | 50 | 3 |
| | | off+pibt | 25,000 | 50 | 3 |
| | | on+gpibt | 20,000 | 100 | 2 |
| | | [p-on]+gpibt | 20,000 | 100 | 2 |
| | | off+gpibt | 20,000 | 100 | 2 |
| | dynamic | on+pibt | 25,000 | 50 | 3 |
| | | off+pibt | 25,000 | 50 | 3 |
| | | on+gpibt | 50,000 | 100 | 2 |
| | | [p-on]+gpibt | 50,000 | 100 | 2 |
| | | off+gpibt | 50,000 | 100 | 2 |
| *empty* | uniform | on+pibt | 25,000 | 50 | 3 |
| | | off+pibt | 25,000 | 50 | 3 |
| | | on+gpibt | 10,000 | 100 | 2 |
| | | [p-on]+gpibt | 10,000 | 100 | 2 |
| | | off+gpibt | 10,000 | 100 | 2 |
| | dynamic | on+pibt | 25,000 | 50 | 3 |
| | | off+pibt | 25,000 | 50 | 3 |
| | | on+gpibt | 10,000 | 100 | 2 |
| | | [p-on]+gpibt | 10,000 | 100 | 2 |
| | | off+gpibt | 10,000 | 100 | 2 |
| *random* | uniform | on+pibt | 50,000 | 50 | 3 |
| | | off+pibt | 50,000 | 50 | 3 |
| | | on+gpibt | 100,000 | 100 | 2 |
| | | [p-on]+gpibt | 10,000 | 100 | 2 |
| | | off+gpibt | 100,000 | 100 | 2 |
| | dynamic | on+pibt | 50,000 | 50 | 3 |
| | | off+pibt | 50,000 | 50 | 3 |
| | | on+gpibt | 50,000 | 100 | 5 |
| | | [p-on]+gpibt | 10,000 | 100 | 5 |
| | | off+gpibt | 50,000 | 100 | 5 |

Table 1: CMA-ES hyperparameters.

| | *sortation* | *warehouse* | *empty* | *random* |
|---|---|---|---|---|
| $\mu$ selection | *ep* | *ep* | *ap* | *ap* |
| $\sigma$ | 1.0 | 1.0 | 0.5 | 0.5 |
| K | 1 | 1 | 3 | 3 |
| $N_d$ | 200 | 200 | 200 | 200 |

Table 2: Dynamic task distribution hyper-parameters. *ep* stands for endpoints. *ap* stands for any points on the whole map. $\mu$ and $\sigma$ denote the mean and standard deviation of Gaussian distribution, $K$ denotes the $K$-mode Gaussian, and $N_d$ denotes the task distribution change interval.

# B Detailed Experiments Setups

In this section, we present the detailed setups of the experiments presented in Section 5.

## B.1 Hyper-parameters

**CMA-ES.** Table 1 shows the hyper-parameters in CMA-ES. $b$ stands for batch size, $N_e$ stands for the number of simulations used to evaluate one solution sampled from the CMA-ES Gaussian distribution, and $N_{eval}$ stands for the total number of evaluation during the optimization. While the hyper-parameters vary, we use the same hyper-parameters for the settings that we make comparisons.

**Task setup.** As mentioned in Section 5.1, we apply Gaussian and multi-mode Gaussian as the dynamic task distributions on different maps. Table 2 shows the detailed hyper-parameters. To sample the tasks, we regard the map as a 2D space. The Gaussian center $\mu = (x, y)$ is selected from a subset of all locations of the map, where $x$ and $y$ are positions in the map. $N_d$ serves as the task distribution change interval. After $N_d$ steps, the Gaussian centers are randomly sampled again, and agents' new tasks are sampled from the updated (multi-mode) Gaussian distribution.

## B.2 On+PIBT Guidance Policy

The guidance policy for on+PIBT is a 3-layer Convolutional Neural Network (CNN). The kernel sizes on each layer are 3, 1, and 1, respectively. We choose ReLU and BatchNorm2d to serve as the non-linearity layers. The observation of the guidance policy includes past traffic and current tasks. Concretely, the past traffic is the edge usage (including self-edge) in previous $m$ steps. It is encoded in shape $(5, h, w)$, where 5 denotes the outgoing edges in 5 directions (right, up, left, down, self) of each vertex, and $(h, w)$ denotes the height and width of the map. The current tasks encodes the current task locations for all agents with the shape of $(h, w)$. For location $(x, y)$, the value is the number of tasks on $(x, y)$. Each channel of the observation is normalized in $(0, 1)$ to stabilize the model output. The output of the guidance policy is the guidance graph edge weights of shape $(5, h, w)$. The total number of parameters is 3,119.

We use a small CNN with small kernel sizes, not only because the model's observation is a 3D tensor, but also to ensure the number of parameters is not too large so that it is easier for CMA-ES to optimize.

Algorithm 1: Search for guide-paths by using guidance policy

1: **Input** start location $s$; goal location $goal$; online policy $\pi_\theta$; current all agents guide-paths edge usage $U_e$
2: **Notation** OPEN, open list in A* search, which is a priority heap; $h(x)$, any admissible heuristic function for location $x$.
3: root ← search-node(loc=s, g=0, h=h(s), parent=none)
4: OPEN ← {root} {initialize open list}
5: **while** OPEN $\neq \phi$ **do**
6: curr ← OPEN.pop()
7: curr.close()
8: **if** curr.loc= $goal$ **then**
9: break
10: **end if**
11: $obs$ ← Windowed-Obs(curr.loc, win_size, $U_e$)
12: $cost \leftarrow \pi_\theta(obs) + 1$. $\{cost \in \mathbb{R}^4\}$
13: **for** $nid, n$ in $enumerate$(curr.loc.neighbors) **do**
14: $cost_n = cost[nid]$
15: next ← search-node(loc=$n$, g=curr.g+$cost_n$, $h = h(n)$, parent=curr)
16: **if** $n$ is not equal to the location of any nodes in OPEN **then**
17: OPEN.push(next)
18: **else**
19: {∃existing ∈ OPEN s.t. existing.loc=next}
20: **if** next.g < existing.g **then**
21: assert not existing.is_closed()
22: OPEN.update(existing, next)
23: **end if**
24: **end if**
25: **end for**
26: **end while**

## B.3 On+GPIBT Guidance Policy

**Guidance policy implementation.** As mentioned in Section 4, we use the model before one agent plans its guide-path. In implementation, we use a *lazy* mechanism instead. We only compute the edge weights of the guidance graph which are expanded when searching for the shortest guide-paths on the guidance graph. With this lazy mechanism, the model avoids computing useless edge weights on the guidance graph, and the search process could be faster. Algorithm 1 shows the pseudo-code of the search process for the guide-path of each agent. Overall, it is an A* search. Line 3 and 4 initialize the OPEN list with the start location $s$. Line 5 to Line 26 is the main *while* loop of the A* search. Line 6 pops the node with least $f$ values (the sum of $g$ and $h$). Line 7 then closes the popped node. It is the preparation steps before expanding the *curr* node. Line 8 checks whether the expanded node is the goal location. We then compute the model observation (Line 11) and use the guidance policy to compute the edge weights on the guidance graph (Line 12). We explain the windowed observation in the next subsection. Using the weights, we continue to expand the current node. Line 13 to Line 25 is the *for* loop for the expanding process, the same as the typical A* search.

| agents num | 600 | 800 | 1000 |
|---|---|---|---|
| PIBT | -0.02, **0.10** | -0.02, **0.15** | -0.02, **0.18** |
| GPIBT1 | 0.18, **0.19** | **0.22**, 0.20 | **0.23**, 0.13 |
| GPIBT2 | 0.01, **0.16** | 0.00, **0.18** | -0.01, **0.12** |

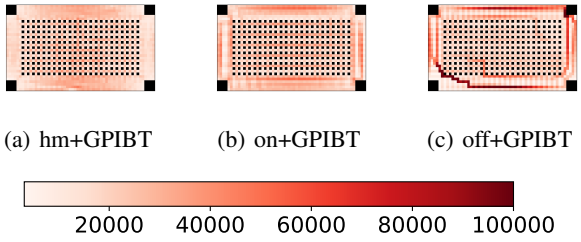
Table 3: Improvement ratio table for the *sortation* map.

| agents num | 400 | 500 | 600 |
|---|---|---|---|
| PIBT | 0.14, **0.23** | 0.20, **0.30** | 0.25, **0.31** |
| GPIBT1 | **0.24**, 0.22 | **0.25**, 0.16 | **0.21**, 0.18 |
| GPIBT2 | -0.03, **0.12** | -0.04, **0.07** | -0.07, **0.07** |

Table 4: Improvement ratio table for the *warehouse* map.

| agent num | 400 | 500 | 600 |
|---|---|---|---|
| PIBT | 0.01, **0.12** | 0.01, **0.11** | 0.02, **0.10** |
| GPIBT1 | 0.07, **0.18** | 0.10, **0.17** | 0.14, **0.19** |
| GPIBT2 | -0.01, **0.12** | -0.03, **0.10** | -0.03, **0.06** |

Table 5: Improvement ratio table for the *empty* map




(a) hm+GPIBT (b) on+GPIBT (c) off+GPIBT

Figure 9: Vertices usage of Guide-paths.

**Guidance policy model parameters.** We use win_size to denote the window size of the observation. The observation of the guidance policy is the windowed guide-paths edge usage, encoded in the shape of (4, win_size, win_size). The window center is located at the current expanding node. 4 channels stand for right, up, left, and down guide-paths usage of other agents. We choose win_size=5. The output of the model is the weights of the guidance graph on 4 outgoing edges of the current expanding node.

We use a simple quadratic function as guidance policy. Every term in the windowed guide-path edge usage is associated with an optimizable linear variable. We further include quadratic variables on *contra-flow* of the windowed guide-path usage. That is, for each pair of adjacent vertex $u$ and $v$ included in the guide-path usage, we consider the product term of guide-paths usage on $(u, v)$ and $(v, u)$. The total number of parameters for on+GPIBT online guidance policy model is 560.

### B.4 Compute Resource

We run our experiments on 4 machines: (1) a local machine with a 64-core AMD Ryzen Threadripper 3990X CPU, 192 GB of RAM, and an Nvidia RTX 3090Ti GPU, (2) a local machine with a 64-core AMD Threadripper 7980X CPU, 128 GB of RAM, and an Nvidia RTX 1080Ti GPU, (3) a Nectar Cloud instance with a 32-core AMD EPYC-Rome Processor and 64GB RAM, and (4) a high-performing cluster with numerous 64-core AMD EPYC 7742 CPUs, each with 256GB of RAM. The CPU runtime is measured on the machine (1).

### B.5 Relevant Software Library

We follow the previous work (Zhang et al. 2024) and implement CMA-ES using Pyribs (Tjanaka et al. 2023). We use the open-source Guided-PIBT implementation of Chen et al. (2024), and PIBT implementation of Jiang et al. (2024).

## C Additional Experiments Results

In this section, we show the following additional experiment results: (1) numerical results of improvement ratio, (2) visualization of on+GPIBT guidance policy, (3) runtime of different algorithms, (4) experiments with LNS on GPIBT-based algorithms, and (5) ablation results on the guidance update interval $m$.

### C.1 Numerical Results of Improvement Ratio

Tables 3 to 5 show the numerical results of the improvement ratio in Figure 5 of Section 5.2. Each cell in the tables contains 2 numbers, denoting the improvement ratio with the uniform and dynamic task distribution, respectively. In PIBT, the improvement ratio of dynamic task distribution is consistently larger than that of uniform task distribution. In GPIBT, we first compare the on+GPIBT and the off+GPIBT, which is indicated by GPIBT1 in the tables. We observe that the improvement ratio in the dynamic task distribution setting is not consistently larger than the uniform one. We hypothesize that the co-effect of the online mechanism and the guide-paths mechanism makes the advantage of online mechanism not obvious. To illustrate the effect of the online mechanism only, we further compare [p-on]+GPIBT and off+GPIBT (denoted by GPIBT2) in the tables. Its improvement ratio consistently increases when moving to the dynamic task distribution.

### C.2 Visualization of on+GPIBT Guidance Policy

Figure 9 shows the vertices usage of guide-paths on the *sortation* map with 800 agents in a 1,000-timesteps simulation under the uniform task distribution. The vertices usage counts the number of times each vertex is used by the guide-paths throughout the simulation. We increment the guide-path vertices usage even if guide-paths do not change. While the guide-paths of hm+GPIBT prefer the center area, our on+GPIBT makes better utilization of the marginal area. In addition, since the hm+GPIBT and on+GPIBT methods are online and use existing guide-paths as the observation, they can make the guide-paths distributed more evenly on the map. The offline method, on the other hand, does not leverage other agents' guide-paths, resulting in repeatedly planning the same guide-paths among agents.

| map-type | on+PIBT | off+PIBT | on+GPIBT | [p-on]+GPIBT | off+GPIBT | hm+GPIBT |
|---|---|---|---|---|---|---|
| *sortation*-u | 25.499±0.334 | **11.284±0.299** | 16.752±0.214 | 5.735±0.698 | **2.943±0.078** | 3.326±0.131 |
| *sortation*-d | 24.080±0.388 | **8.024±0.259** | 9.504±0.578 | 5.117±0.834 | **2.176±0.075** | 2.605±0.104 |
| *warehouse*-u | 15.522±0.683 | **5.883±0.306** | 7.902±0.265 | 3.826±0.692 | **1.440±0.029** | 1.475±0.036 |
| *warehouse*-d | 14.926±0.769 | **5.084±0.388** | 4.074±0.269 | 3.618±0.715 | **1.203±0.047** | 1.292±0.0483 |
| *empty*-u | 11.137±0.298 | **6.405±0.196** | 10.322±0.317 | 3.665±0.603 | **1.787±0.066** | 2.026±0.092 |
| *empty*-d | 9.815±0.460 | **4.439±0.306** | 8.482±0.548 | 3.149±0.636 | **1.200±0.056** | 1.267±0.051 |
| *random*-u | 8.823±0.544 | **3.390±0.460** | 1.654±0.268 | 2.287±0.613 | 0.610±0.064 | **0.564±0.050** |
| *random*-d | 7.806±0.684 | **3.261±0.411** | 1.575±0.247 | 2.278±0.639 | 0.630±0.060 | **0.561±0.041** |

Table 6: CPU runtime (in seconds) table for all algorithms for 1,000 timesteps. Type $u$ and $d$ stand for uniform and dynamic task distribution. The numbers of agents are 800, 600, 400, and 400 for the *sortation*, *warehouse*, *empty* and *random* map.

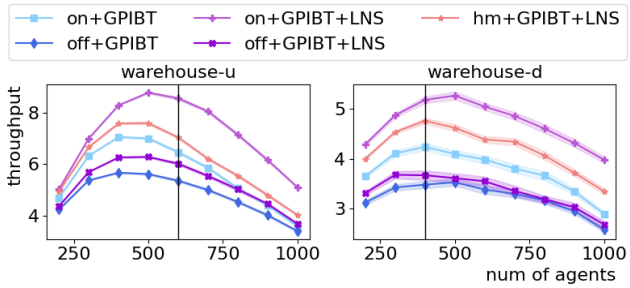



Figure 10: GPIBT with LNS results.

| algo | on+GPIBT+LNS | off+GPIBT+LNS | hm+GPIBT+LNS |
|---|---|---|---|
| uniform | 42.145±0.562 | **7.828±0.194** | 8.383±0.111 |
| dynamic | 38.553±1.528 | **6.680±0.364** | 8.961±0.321 |

Table 7: CPU runtime (in seconds) of LNS-based algorithms for 1000 steps. *Uniform* and *dynamic* indicate the task distribution.

## C.3 Runtime

Table 6 shows the CPU runtime (in seconds) of all algorithms. The runtime for on+PIBT is about 2-3 times slower than off+PIBT because of recomputing the heuristics. For GPIBT-based methods, the runtime of off+GPIBT and hm+GPIBT are similar because they require negligible efforts to compute the edge weights of the guidance graph when planning the guide-paths. [p-on]+GPIBT does not need to compute edge weights when searching guide-path, but it needs to compute observation when it updates the guidance graph. On+GPIBT is up to 4 times slower than hm+GPIBT and off+GPIBT because it updates the guidance graph when each agent replans its guide-path. However, the computational overhead caused by the online mechanism is acceptable. The average runtime per timestep is no larger than 0.026 seconds in all experiments, which is fast enough for the realistic setting.

## C.4 GPIBT with LNS

Section 4 mentioned that the GPIBT-based methods can incorporate LNS to improve the guide-paths quality and further improve throughput. In this subsection, we show the results of on+GPIBT+LNS, off+GPIBT+LNS and hm+GPIBT+LNS. We optimize the guidance policies or guidance graphs with LNS in the loop during optimization.

LNS refines agents' guide paths with an iterative procedure. In each LNS iteration, LNS selects *group_size* agents to replan their guide-paths from their current locations. If the new sum of all agents' action costs is smaller, then LNS accepts the new guide-paths. At each timestep, LNS is terminated either if it has exceeded $N_{iter}$ iterations or if it has been running for more than 8 seconds.

**Experiments setup.** We focus on *warehouse* map with uniform and dynamic task distribution. We set LNS hyper-parameters as $N_{iter} = 10$ and *group_size* $= 10$. We set CMA-ES hyper-parameters as $b = 100$, and $N_{eval} = 2$. $N_{eval}$ is set to $20,000$ for uniform task distribution and $50,000$ for dynamic task distribution. All experiments using CMA-ES are optimized with 600 agents for the uniform task distribution and 400 agents for the dynamic one.

**Results.**Figure 10 show the *throughput-agents* curve with LNS experiments. Comparing on+GPIBT with on+GPIBT+LNS shows that LNS substantially improves the throughput. The throughputs of on+GPIBT+LNS and the hm+GPIBT+LNS both dominate that of off+GPIBT+LNS, indicating that under LNS setting, all online methods are better than the offline method. Interestingly, the improvement of on+GPIBT over off+GPIBT is larger than off+GPIBT+LNS over off+GPIBT, indicating that the online mechanism is more helpful than LNS. That is because off+GPIBT ignores other agents' guide-paths. LNS only helps when an agent deviates from its guide-path, replanning the agent's guide-path from the deviated location instead of continuing to follow the previously planned guide-path. However, ignoring other agents' guide-paths cannot lead to higher throughput.

Table 7 shows the CPU runtime of all LNS-based algorithms. It is much slower than the algorithms without LNS (Table 6). However, the runtime is still acceptable. The most expensive one (on+GPIBT+LNS on uniform task distribution) is only 0.042 seconds per timestep.

## C.5 Ablation on Guidance Update Interval

Section 4.1 mentioned the guidance update interval $m$ for PIBT. The selection of $m$ is a trade-off between the adaptability of the guidance policy to varying traffic patterns and

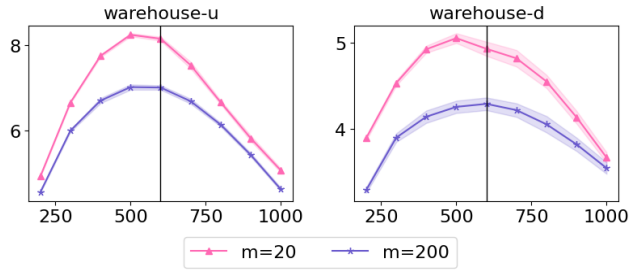



Figure 11: *throughput-agents* curve of $m = 20$ and $m = 200$.

| $m$ | 20 | 200 |
|---|---|---|
| uniform | 15.522±0.683 | **7.486±0.822** |
| dynamic | 14.926±0.769 | **6.736±0.884** |

Table 8: CPU runtime (in seconds) for different $m$ on the *warehouse* map for 1,000 timesteps. *Uniform* and *dynamic* indicate the task distribution.

the computational overhead. Updating the guidance graph more frequently could lead to better throughput at the expense of more computational overhead.

**Experiments setup.** In this section, we compare the performance of $m = 20$ and $m = 200$. We optimize all guidance policies with CMA-ES with 600 agents for the uniform task distribution and 400 agents for the dynamic one.

We focus on *warehouse* map with uniform and dynamic task distribution. We set CMA-ES hyper-parameters $N_{eval} = 25,000$, $b = 50$, and $N_e = 3$. All experiments with CMA-ES are optimized on 600 agents and 400 agents for uniform and dynamic task distribution, respectively.

**Results.** Figure 11 shows the ablation results of $m = 20$ and $m = 200$. We observe that the throughput of $m = 20$ dominates $m = 200$ with various numbers of agents. Table 8 shows the CPU runtime for different $m$. We observe that $m = 20$ is one time slower than $m = 200$. Given the trade-off of runtime and throughput with the choice of $m$, it depends on the exact planning time limit of the downstream application scenario of our algorithm to determine the exact value of $m$.

## D Errata

The optimization number of agents on *warehouse-33-57* map with dynamic task distribution for on+PIBT and off+PIBT settings are 600 instead 400 mentioned in Figure 3.