# Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding (Extended Abstract)*

**Zhe Chen**[1], **Daniel Harabor**[1], **Jiaoyang Li**[2], **Peter J. Stuckey**[1,3]

[1]Department of Data Science and Artificial Intelligence, Monash University, Melbourne, Australia
[2]Robotics Institute, Carnegie Mellon University, Pittsburgh, USA
[3]OPTIMA Australian Research Council ITTC, Melbourne, Australia
{zhe.chen,daniel.harabor,peter.stuckey}@monash.edu, jiaoyangli@cmu.edu

## Abstract

Multi-Agent Path Finding (MAPF) is a fundamental problem in robotics that asks us to compute collision-free paths for a team of agents, all moving across a shared map. Existing scalable approaches struggle as the number of agents grows, as they typically plan free-flow optimal paths, which creates congestion. To tackle this issue, we propose a new approach for MAPF where agents are guided to their destination by following congestion-avoiding paths. Empirically, we report large improvements in overall throughput for lifelong MAPF while coordinating more than ten thousand agents.

## Introduction

Multi-Agent Path Finding (MAPF) (Stern et al. 2019) is a fundamental problem navigating a team of agents from their start locations to goal locations without any collisions. In the classical, sometimes called *one-shot*, MAPF problem (Stern et al. 2019), each agent is assigned a single goal location. A related setup, known as *lifelong* MAPF (Li et al. 2021), continuously assigns new goal locations to agents as they arrive at their current goal locations.

These problems have been intensely studied, with a variety of substantial advancements reported in the literature. Leading optimal (Shen et al. 2023) and bounded-suboptimal (Li, Ruml, and Koenig 2021) MAPF algorithms now scale to hundreds of agents with solution quality guarantees. Yet some real applications require up to *thousands of simultaneous agents*, and at this scale, only unbounded suboptimal approaches are currently applicable.

One leading framework for unbounded suboptimal MAPF is Priority Inheritance with Back Tracking (PIBT) (Okumura et al. 2022). PIBT-based approaches use rule-based collision avoidance to plan paths. They compute paths timestep by timestep, which is extremely efficient. For this reason, PIBT-based methods usually scale to substantially large teams of agents. However, this type of planner guides agents toward their goals using individually optimal *free-flow* heuristics (i.e., considering only travel distance while ignoring other agents), a strategy known to create high levels of congestion. Moreover, its computed solutions tend to have higher costs than other approaches.

*Full AAAI-24 paper: https://arxiv.org/abs/2308.11234



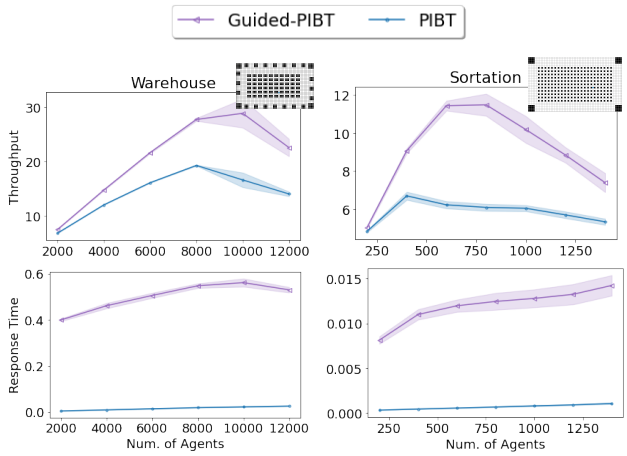


Figure 1: Average throughput (top) and response time in second (bottom). Shaded regions show standard deviation.

## Methodology

To tackle the drawbacks of PIBT, we propose to compute *congestion-aware* time-independent optimal paths. These paths are called *Guide Paths* and will be followed by agents in PIBT. In particular, we update the edge costs of the graph after the computing of one time-independent shortest path, so that the edges that appear on that path become more expensive to traverse. We adapt these general ideas for MAPF by first defining a model for computing traffic costs and then applying these costs to compute *guide paths* and new congestion-aware move policies for PIBT.

### Traffic Costs

Given an undirected graph $G = (V, E)$, we define $f_{v_1,v_2}$ as the *flow* from vertex $v_1$ to $v_2$ given a set of agents whose time-independent shortest paths each have the state $v_1$ followed by $v_2$. Thus $f_{v_1,v_2}$ is the number of agents traverse the edge $(v_1, v_2)$ from $v_1$ to $v_2$ direction in the current path assignment. Note that $f_{v_2,v_1}$ indicates traversing through the same edge but from $v_2$ to $v_1$ direction.

We define the *vertex congestion* $c_v$ of a vertex $v \in V$ as $c_v = \frac{n \times (n-1)}{2}$ where $n = \sum_{v' \in V:(v',v) \in E} f_{v',v}$ is the total number of agents entering vertex $v$. $c_v$ represents the

| Metric | ALG | 200 | 400 | 600 | 800 | 1000 |
|---|---|---|---|---|---|---|
| Throughput | RHCR-PBS | **6.0±0.1** | **11.1±0.1** | 6.5±0.6 | 5.2±0.5 | 3.5±0.3 |
| | Guided-PIBT | 5.0±0.1 | 9.1±0.1 | **11.4±0.3** | **11.5±0.6** | **10.2±0.7** |
| Response Time per Planner Run | RHCR-PBS | 0.114±0.002 | 1.449±0.078 | 9.985±0.074 | 10.051±0.028 | 10.03±0.013 |
| | Guided-PIBT | **0.008±0.0** | **0.011±0.001** | **0.012±0.001** | **0.012±0.001** | **0.013±0.001** |

Table 1: RHCR vs. Guided PIBT on Sortation centre map, with number of agents varies from 200 to 1000. It compares the Throughput and Response Time per planner run.

least total delay that will occur assuming all agents enter the vertex at the same time, since in the best case each agent will have to wait for all the agents preceding it. Apportioning this congestion to each agent equally leads to a cost per agent of $p_v = \lceil \frac{c_v}{n} \rceil = \lceil \frac{n-1}{2} \rceil$.

Define the *contraflow congestion* $c_e$ of undirected edge $e \equiv (v_1, v_2) \in E$ as $c_e = f_{v_1,v_2} \times f_{v_2,v_1}$. This formulation reflects our observation that higher-priority agents move into a corridor which then forces lower-priority agents, already inside the corridor, to reverse direction.

## Computing Guide Paths

Traversing through each edge $e = (v_1, v_2)$ is given a two-part weighted cost $(c_e, 1 + p_{v_2})$, where 1 indicates the free-flow (i.e., zero congestion) cost of using edge $e$. We then search for (lexicographically) shortest paths for each agent using this two-part cost, i.e. first minimising contraflow costs, and then weighted edge costs.

We begin by initialising flow counts to zero, and then plan agents one by one and update the flow counts at the same time. Once all agents have guide paths we call the procedure PathRefinement, which uses a local search to improve the computed guide paths.

## Guide Heuristics

We then modify PIBT so that each agent tries to follow a congestion-aware *guide path*. For each agent $a_i$ we thus compute a *guide heuristic* $h_i(v)$ using backward Breadth First Search (BFS). Given a vertex $v \in V$ we compute a two-part value $h_i(v) = (dp, dg)$, where $dp$ is the shortest free-flow distance, from $v$ to the guide path. The value $dg$ meanwhile is the shortest remaining distance to the goal $g_i$, as reached subsequently by strictly following the guide path. When computing its move policies, PIBT then prefers the vertex $v$ with the smallest $h_i(v)$.

# Empirical Evaluation

We compare Guided-PIBT against the original PIBT and RHCR (Li et al. 2021) for Lifelong MAPF, where agents continuously receive new tasks when they finish tasks. Implementations[1] are written in C++ and evaluated on a Nectar Cloud VM instance with 32 AMD EPYC-Rome CPUs and 64 GB RAM. We run large-scale experiments on two maps: (1) a $500 \times 140$ *Warehouse* map with 3200 simulation timesteps limit; (2) a $33 \times 57$ *Sortation* centre map with 450 simulation timesteps limit. For each map and each number of agents, we evaluate 24 randomly sampled instances.

Figure 1 measures the average throughput and average response time, which is the average time the planner returns actions for all agents at each timestep, and compares against baseline PIBT. The `Guided-PIBT` shows a great advantage over PIBT on throughput. We noticed that in lifelong MAPF there is a peak agent density, beyond which adding more agents decreases throughput due to increasingly severe congestion. Compared with PIBT, our methods shifted this peak to the right on each map, with `Guided-PIBT` improving Warehouse by 2000 agents and Sortation by 200 agents.

Table 1 show the throughput and response time per planner run comparisons against RHCR-PBS (Li et al. 2021), a leading framework in solving lifelong MAPF problems, on the *Sortation* centre map. The planning window is set to 10 and the execution window is 5, which indicates the simulator calls the MAPF planner every 5 timesteps with the bounded time horizon set to 10. The time limit for the planner is set to 10 seconds. The results show that our Guided PIBT easily scales to a larger team size with higher throughput, and uses dramatically less computing resources.

[1] https://github.com/nobodyczcz/Guided-PIBT

# References


Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12353–12362.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11272–11281.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Shen, B.; Chen, Z.; Li, J.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2023. Beyond pairwise reasoning in multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, 384–392.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Twelfth Annual Symposium on Combinatorial Search*, 151–158.