# Planning and Execution in Multi-Agent Path Finding: Models and Algorithms (Extended Abstract) *

**Yue Zhang, Zhe Chen, Daniel Harabor, Pierre Le Bodic, Peter J. Stuckey**

Monash University, Australia
{Yue.Zhang, Zhe.Chen, Daniel.Harabor, Pierre.LeBodic, Peter.Stuckey}@monash.edu

## Abstract

In applications of Multi-Agent Path Finding (MAPF), it is often the sum of planning and execution times that needs to be minimised (i.e., the *Goal Achievement Time*). Yet current methods seldom optimise for this objective. Optimal algorithms reduce execution time, but may require exponential planning time. Non-optimal algorithms reduce planning time, but at the expense of increased path length. To address these limitations we introduce PIE (Planning and Improving while Executing), a new framework for concurrent planning and execution in MAPF. We first show how PIE for one-shot MAPF improves practical performance compared to sequential planning and execution. We then adapt PIE to Lifelong MAPF, a popular application setting where agents are continuously assigned new goals and where additional decisions are required to ensure feasibility. We examine a variety of different approaches to overcome these challenges and we conduct comparative experiments vs. recently proposed alternatives. Results show that PIE substantially outperforms existing methods for One-shot and Lifelong MAPF.

## Introduction

Multi-Agent Path Finding (MAPF) (Stern et al. 2019) is the problem of finding collision-free paths for a team of agents. Conventional MAPF focuses on solving the "one-shot" version of this problem, which is solved when all agents are at their goal. Existing studies typically solve MAPF by assuming that necessary computation time is available up front (Lam et al. 2022; Li et al. 2021b,a; Okumura 2023). Smaller times are preferable but typically not reflected in the corresponding objective functions, which instead aim to minimise action costs; e.g, Makespan (Yu and LaValle 2013) or Sum-of-Costs (Stern et al. 2019). The main advantage of this approach, sometimes known as *offline planning* is that execution times are as small as possible, subject to timeout limits (which can range from seconds to hours). The main drawback to offline planning is a mismatch between the model assumptions and the requirements of real applications, which can be entirely *online*. In other words, if a plan is not immediately available real-world agents simply *wait in place*, until the planner can provide instructions.

*This paper has been accepted to ICAPS 2024



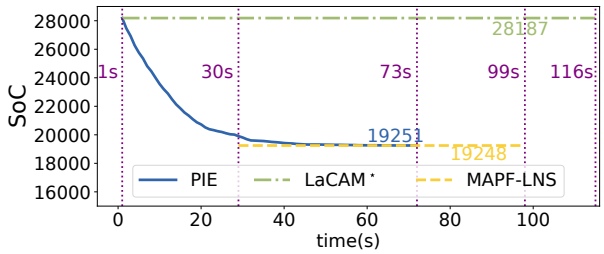


Figure 1: Planning and execution costs for 3 different MAPF algorithms on a small grid map; `random-32-32-20` with 400 agents and unit action costs. PIE and LaCAM* compute the same initial solution and begin execution after 1s. MAPF-LNS plans for a further 29s then begins execution.

In this work, we propose a new *concurrent planning* framework, which we call PIE: Planning and Improving while Executing, to optimise the sum of planning time and execution time for each agent. PIE leverages fast solvers to quickly compute and commit to a small number of actions for each agent. During the execution of these actions, PIE optimises the remaining paths of agents and then commits to a new set of actions. Figure 1 illustrates the concrete advantages of PIE for MAPF (in blue) compared with two leading offline planners: MAPF-LNS (Li et al. 2021a) (the best known algorithm for anytime MAPF, in yellow) and LaCAM$^\star$ (Okumura 2023) (the best known algorithm for scalable MAPF, in green). The graph shows Sum-of-(executed action)-Cost (SoC) over time, with the endpoint of each line indicating the end of execution (i.e., GAT for the last arriving agent). We make three observations: (i) PIE finishes executing substantially faster than either offline planner; (ii) the execution costs for PIE are very similar to MAPF-LNS, which requires 29 seconds of additional compute; (iii) advantages are magnified when considering up-front wait costs: +400 for PIE and LaCAM$^\star$ and +12000 (400×30) for MAPF-LNS.

## Planning and Improving while Executing

PIE has several components, which must be instantiated:
**Initial Planning Time** ($T_{init}$): this variable is the time allowed to compute an initial solution. $T_{init}$ is also counted as

Algorithm 1: PIE Framework

**Input:** $\langle G, A \rangle$; $T_{init}$, initial planning time limit; $T_{action}$, execution time for one action; $k$, number of actions per commit.

1: $T_{exec} \leftarrow T_{action} * k$
2: $\pi \leftarrow$ Plan_Improve($\langle G, A \rangle, \emptyset, T_{init}$)
3: Commit $\pi_k$
4: $\pi \leftarrow \pi \setminus \pi_k$
5: **while** (Execution of $\pi_k$) **do**
6: Update $A.starts$, $A.goals$ from TO
7: $\pi \leftarrow$ Plan_Improve($\langle G, A \rangle, \pi, T\_exec$)
8: Commit $\pi_k$
9: $\pi \leftarrow \pi \setminus \pi_k$

a waiting cost for every agent in the SGAT.

**How Long to Commit** ($k$): this variable is the number of actions that the agents commit to during each execution phase. Once committed, these $k$ actions cannot be changed.

**Execution time** ($T_{action}$): this variable specifies the time required to execute a single action. Multiplying by $k$ gives the time available for the planner to compute the next set of actions before the agents incur additional waiting time.

**Planner**: the main ingredient in PIE is the planner. We suggest algorithms that can incrementally improve the solution until time out. However, any MAPF planner can be used.

Pseudocode for the PIE framework is shown in Algorithm 1, we take as input $T_{init}$, $k$, $T_{action}$, the map $G$ and a set of agents $A$ with initially assigned start and goal locations. PIE starts by generating an initial solution $\pi$ and improves $\pi$ within the runtime limit $T_{init}$ (line 2). The algorithm then commits the first $k$ actions of $\pi$, and updates the solution $\pi$ to be the uncommitted part of the solution. Then agents iteratively commit and execute (lines 5-9). The loop terminates when all agents stay at goals. During each execution, the planner will plan and improve the uncommitted part of the solution with runtime limit $T_{exec}$ (line 7). After planning, the planner commits the next $k$ actions and updates the uncommitted solution $\pi$ (line 8-9).

We instantiate PIE for one-shot MAPF (where each agent has a single target) and for Lifelong MAPF (where agents are continuously assigned new tasks).

**PIE for One-Shot MAPF:** We combine LaCAM$^\star$, which we use to compute fast feasible solutions, and MAPF-LNS, which we use to improve the costs of uncommitted actions for planning and improving solutions.

**PIE for Lifelong MAPF:** In Lifelong settings, neither LaCAM$^\star$ nor MAPF-LNS can be directly applied due to: (1) agents constantly receive new goals during ongoing execution, and new conflict-free paths to new goals are constantly required; (2) the planner is not aware of where the agent should go after reaching the goal and planner will fail when planning for agents with the same goals.

See the full version of this paper for a complete discussion of the choices of different components for PIE, different strategies for adapting PIE and their trade-offs.

## Experiments: Lifelong MAPF

We run experiments in a VM instance with 32GB RAM, 16 AMD EPYC-Rome CPUs. $T_{init}$ is set to 1s. Figure 2 shows the throughput of PIE, RHCR (Li et al. 2021c), the existing start-of-the-art Lifelong MAPF solver, and an approach of simply replanning every commit using LaCAM$^\star$ on two maps from the grid-based MAPF benchmarks ((Stern et al. 2019)). PIE$_L$ denotes replan all agents when receiving new goals, while PIE$_F$ means replan only affected agents when receiving new goals. PIE$_F$ is the highest throughput approach until the number of agents reaches a point where Replan Affected can not find a solution for affected agents within the commit time, where PIE$_L$ takes over.

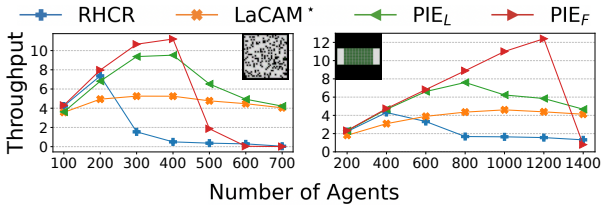


Figure 2: Throughput of RHCR, LaCAM$^\star$, PIE$_L$ and PIE$_F$.

## Conclusions and Future Work

In this paper, we generate an efficient approach to planning and improving while executing for One-Shot MAPF and Lifelong MAPF problems. Overall PIE provides significantly greater performance than competing approaches. Future work will extend PIE to consider execution delaies, which is an important feature for real-life applications.

## Acknowledgements

This work is supported by the Australian Research Council under grant DP200100025, and by a gift from Amazon.

## References

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144: 105809.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *IJCAI*, 4127–4135.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021b. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

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

Okumura, K. 2023. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *IJCAI*.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SOCS*, volume 10, 151–158.

Yu, J.; and LaValle, S. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*, volume 27, 1443–1449.