Concurrent Planning and Execution in Lifelong Multi-Agent Path Finding with Delay Probabilities

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 multi-agent systems, when we account for the possibility of delays during execution, online planning becomes more complicated, as both execution and planning should be able to handle delays when agents are moving. Lifelong Multi-Agent Path Finding (LMAPF) is the problem of (re)planning the collision-free moves of agents to their goals in a shared space, while agents continuously receive new goals. PIE (Planning and Improving while Executing) is a recent approach to LMAPF which concurrently replans later parts of agents' trajectories while execution occurs. However, the execution is assumed to be perfect. Existing approaches either use policy-based methods to quickly coordinate agents every timestep with instant delay feedback, or deploy an execution policy to adjust a solution for delays on the fly. These approaches may introduce large amounts of unnecessary delays to agents due to their planner guarantees or simple delay-handling policies. In this paper, we extend PIE to define a framework for solving the lifelong MAPF problem with execution delays. We instantiate our framework with different execution and replanning strategies, and experimentally evaluate them. Overall, we find that this framework can substantially improve the throughput by up to a factor 3 for lifelong MAPF, compared to approaches that handle delays with simple execution policies.

Introduction

When planning online, planners are under time pressure, as agents may have to wait for the planner to compute the solution for execution. An efficient way to handle this problem is concurrent planning and execution (Karaman et al. 2011; Hönig et al. 2019; Zhang et al. 2024). That is, for a given parameter $k \geq 1$, at timestep t, the planner commits to executing the next k steps of the current plan. During times steps $t..t+k-1$ it computes/improves the remainder of the plan. This repeats every k timesteps. However, in multi-agent systems, unexpected delays can occur in execution, due to mechanical differences, communication delays, robot dynamics, etc. When delays occur, two desynchronisation problems arise when following the plan:

- *Problem (1)*: the committed plan may be invalid, because directly executing the paths may cause conflicts;
- *Problem (2)*: the uncommitted plan may be invalid, as the planner assumes agents start from the last location on their committed path, but their actual position after k timesteps may not match the committed plan.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

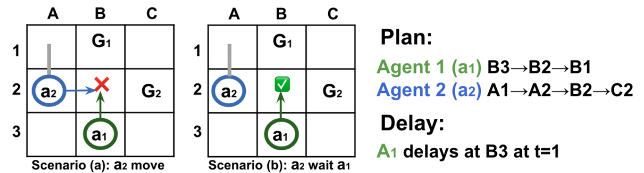

Figure 1: Motivating examples (Snap-shot for timestep 1). Due to delays, a_1 cannot execute the first action ($B3$ to $B2$) and stays at $B3$. Scenario (a) assumes every agent executes its given path directly, resulting a conflict. Scenario (b) uses simple execution policies, which wait for a_2 to traverse $B2$ first. Grey line is the executed path.

Figure 1 shows examples of such problems. Suppose the planner is executing the first $k = 2$ steps of the plan, meanwhile, the planner is computing the future plan assuming a_1 starts at $B1$ and a_2 starts at $B2$. In Scenario (a), a_2 directly moves to $A2$ according to its given path. Then both agents want to occupy $B2$ at timestep 2, and therefore, cause Problem (1). In Scenario (b), a_2 follows a fixed execution policy: in case of delay, to wait for a_1 to enter and leave $B2$ first, at the end of the commit window leaving a_1 at location $B2$ and a_2 at location $A2$, which causes Problem (2). Thus, if the planner is building new future paths assuming agents follow the committed plan without delays, these future paths may be infeasible due to the deviations between the actual location of each agent and their expected locations.

Lifelong Multi Agent Path Finding (LMAPF) (Stern et al. 2019; Ma et al. 2017; Li et al. 2021b) is an important problem in multi-agent systems, which requires coordinating the simultaneous actions of agents so that they reach continuously assigned new target locations without any collisions. It is the main abstraction for many practical applications, such as autonomous warehouses (Li et al. 2021b), unmanned aerial vehicles (Ho et al. 2019, 2022) and autonomous vehicles (Li et al. 2023). In LMAPF, one way to handle delays is through offline $k-$robust planning, i.e. their execution is safe with up to k timestep of delays (Atzmon et al.

2018, 2020; Chen et al. 2021). However, this method sacrifices solution quality for safety, making plans longer than necessary. It also cannot handle delays longer than k time steps and requires replanning in such cases. Alternatively, delays can be managed by online planning, such as planning in a step-by-step manner using policy-based planners like PIBT (Okumura et al. 2022; Okumura, Tamura, and Défago 2021). These methods are efficient when agents need to frequently adjust their paths due to changes and potential delays, as their computation time is negligible. However, limiting to only lookahead one step often leads to poor solution quality. Another approach to tackle these disadvantages is building optimised execution policies from an offline near-optimal plan. For example, in (Ma, Kumar, and Koenig 2017; Hönig et al. 2019), authors propose to build a dependency graph from the given plan and follow it to execute agents, which solves Problem (1). Hönig et al. (2019) further apply this approach in LMAPF. To solve Problem (2), the authors propose to only execute the future plan after all robots have reached the last location of the current committed path with execution policies. However, it introduces additional delays for agents to synchronise. For example, one agent's delay before its last action will cause all other undelayed agents to wait in place.

In this paper, we handle the two problems raised by delay probabilities in LMAPF. Our first contribution is the introduction of a framework that effectively manages this problem while maintaining persistent execution and minimising unnecessary waiting. This framework combines replanning and execution policies. Following Zhang et al. (2024), we call this new framework Planning and Improving while Executing with Delay Probabilities (PIE-D). Our second contribution is that we instantiate PIE-D with different replanning strategies and execution policies and run large-scale empirical evaluations. We show that our framework outperforms two existing delay-handling baselines with up to three times throughput improvement in LMAPF.

Preliminaries

In this section, we introduce our problem setting: LMAPF with delay probabilities. We begin by defining the LMAPF problem, then describing the concurrent framework and incorporating the delay model. Then we discuss existing ways of handling delays.

Problem Setting

Lifelong MAPF The input consists of an undirected grid map $G = (V, E)$, where V is a set of vertices (grid cells) and E is a sequence of edges that connects adjacent cells, and a set of m agents $A = \{a_1, \dots, a_m\}$, where each agent has a start location and a set of goal locations. We assume that the goals are assigned by a Task Oracle (TO) and we do not know all the goal locations assigned to each agent *a priori*. In this work, we follow the LMAPF setting in Li et al. (2021b) where the goal assignment from TO is not within the planner's control and we only reveal enough goals to each agent so that the agents's path is always longer than a commit window k. Time is discretised into timesteps. At each timestep, each agent can either perform a *move* action, that transits from the current location to an adjacent location, or a *wait* action, that stays at the current location for this timestep. Each action takes 1 second to execute. A *vertex conflict* occurs when two agents occupy the same vertex at the same time, while an *edge conflict* occurs when two agents traverse the same edge at the same time (Stern et al. 2019). The task for the planner is to plan a conflict-free solution, which consists of m paths, one for each agent, that transit each agent from its start to all the goal locations. The objective is maximising *throughput*, which is the average number of tasks finished per timestep (i.e., the total number of goals reached divided by total time).

Concurrent Planning and Execution in LMAPF The existing solution for solving online LMAPF is concurrent planning and execution (Hönig et al. 2019; Zhang et al. 2024). One leading framework is called Planning and Improving during Execution (PIE) (Zhang et al. 2024), in which the planner fully utilises execution time to plan and improve agents' paths. This framework iteratively commits a window of k steps of the solution for execution. This commitment refers to a decision point regarding the next portion of a planned trajectory, which means no further modifications to the planned portion will occur. During the execution, the planner continuously plans and refines solutions beyond this committed window. PIE reports high throughput for Lifelong MAPF problems. However, in PIE, the planning assumes perfect execution without delays and the locations of agents at the end of the commit window are assumed known, which is not true if delays are possible.

Delay Model In the real world, agents do not execute their plans perfectly. Sometimes they are delayed. We model delays by following MAPF with Delay Probabilities (MAPF-DP) (Ma, Kumar, and Koenig 2017). In MAPF-DP, during the execution, an agent has a probability p to get delayed at its current location for a period of time $l \in [d_{min}, d_{max}]$ instead of moving to the next location according to its computed path. In addition to the MAPF-DP model, we make two realistic assumptions: (1) The original MAPF-DP model requires the execution to avoid the so-called *following conflicts* to simplify the execution policy implementation. Similar to Su, Veerapaneni, and Li (2024), we extend the MAPF-DP by allowing the following conflicts. (2) In some MAPF-DP settings, e.g., simulated railway networks (Laurent et al. 2021), the l duration of delay is known at the time the delay occurs. This knowledge allows the planner to incorporate the delay duration into future decisions. In this paper, we consider a more realistic and common setting where l is unknown to the planners.

Delay Recovery Approaches

One way to handle delays is to dynamically reschedule agents along their planned paths. This type of approach is called an execution policy. The Minimal Communication Policy (MCP) (Ma, Kumar, and Koenig 2017) is a complete plan-execution policy. It works by identifying critical dependencies between the vertex usage of agents, which is equivalent to the idea of Temporal Plan Graph (TPG) or Action Dependency Graph (ADG) (Hönig et al. 2019) on grid-

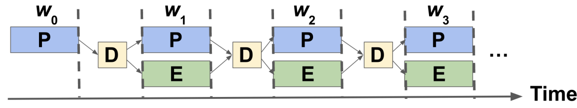

Figure 2: Overview of PIE-D framework. In each k-step execution window (w_n), the Executor (E) executes the path from the current commit, while the Planner (P) is planning paths for the future commit windows in parallel. The Dummy Simulation (D) is invoked between each window.

based MAPF problem, and outputs move commands that follow these dependencies during plan execution. In Figure 1 there is dependency between a_1 and a_2 at $B2$, in the plan a_1 gets to use $B2$ first. Recent studies optimise the dependency graph by switching dependencies (Su, Veerapaneni, and Li 2024; Feng et al. 2024). In (Kottinger et al. 2024), authors also propose a similar approach that rescheduling agents by introducing delays in their original paths. These advances can reduce the amount of unnecessary waiting by allowing certain agents to move first, but require additional computational overhead for an online environment.

Another way is using existing planning approaches to replan (Li et al. 2021b). However, in online LMAPF, agents may need to stop and wait for a feasible plan to be computed. In this case, Priority Inheritance with Backtracking (PIBT) (Okumura et al. 2022) can be used. PIBT is a single-step planning approach to coordinate agents' movements and avoid collisions. It can rapidly compute next collision-free actions to recover from delays. Okumura, Tamura, and Défago (2021) further enhanced PIBT by using a given MAPF plan as a hint, which is equivalent to use PIBT as an execution policy to execute the MAPF plan. We call this approach PIBT-I in this work. However, this approach does not guarantee a feasible execution.

Planning while Executing with Delays

To solve LMAPF with delays, existing approaches either through planning, which may suffer from the trade-offs of solution quality and computation time; or through execution policy, which may lead to unnecessary delays of synchronising between the planner and execution. In this section, we show a new framework to solve this problem that combines the advantages of both approaches while mitigating the effects of delays. We call it Planning and Improving while Executing with Delay Probabilities (PIE-D).

Components As shown in Figure 2, this framework contains the following components:

- **Planner (P)** is responsible for generating and improving plans while agents are moving. It has three functions, including (1) initial path planning before starting execution (2) replanning for agents that have new goal locations or are different from their original path due to delays, and (3) using any remaining time to improve the current feasible solution with MAPF-LNS (Li et al. 2022).
- **Executor (E)** refers to the real-time execution of agents, which manages the actual movement of agents according to the given plan. It ensures that the agents follow the plan as closely as possible. To handle execution delays, the Executor follows an execution policy, for example, MCP, to ensure feasibility.
- **Dummy Simulation (D)** mitigates the desynchronisation problem by simulating the next commitment. After the current $k-$step execution, there may be actions that are committed but not executed due to delays. The Dummy Simulation will get those actions, and append the next planned actions together to run a $k-$step simulation for the next commit window, assuming no delays, to predict the next $k-$actions that commit for the next execution, and the next start/goal locations for the next planning phase. In other words, the Executor will follow the simulated commitment to execute, and the planner uses the prediction to plan and improve future plans.

Algorithm 1: PIE-D Framework

Require: $\langle G, A\rangle$; k, Commit Window, P, Planner; E, Executor; D, Dummy_Simulation
1: $\pi \leftarrow$ P.Initial_Planning(A)
2: **while** not interrupted **do**
3: $states \leftarrow$ E.Get_Current_States()
4: $\pi_k, A.starts, A.goals \leftarrow$ D.Simulate(π, $states$)
5: $\pi \leftarrow \pi \setminus \pi_k$
6: In parallel do:
7: E.execute(π_k, k))
8: $\pi \leftarrow$ P.Plan($\langle G, A\rangle, \pi$)

Algorithm As shown in Algorithm 1, the framework starts with the Planner component generating an initial path plan, π, for all agents (line 1). Then the framework retrieves the current states, including the current location for each agent, and any path that is committed to the Executor but not executed (line 3). Then the Dummy Simulation simulates the next k steps π_k based on $states$ and π, and predicts the $A.starts$, which are agents' locations at the end of k steps, and $A.goals$, which are the updated goal locations if agents should arrive at their current goal locations in their next k steps (line 4). Then π_k is committed to the Executor (line 5) for execution. As the Executor executes π_k for k steps (line 7), the future path after π_k, π, is simultaneously planned and improved by the *plan* function in Planner (line 8). During planning, we first select the subset of agents A' that need replanning. This includes agents that have infeasible paths due to delays or empty paths due to update in goals. If A' is not empty, the Planner first replans a feasible solution. Then the Planner continues to improve the plan until the execution finishes. The same concurrent planning and executing then dummy simulation process continues as long as the system is not interrupted (e.g., simulation timestep limit not reached). Note that Dummy Simulation, line 4, takes negligible time.

Example We use Figure 1 as an example to illustrate the process. Now suppose we commit $k = 1$ step each time and the execution policy we use is MCP.

First, after initial planning, we input: (1) the first action from the Planner ($B3 \rightarrow B2$ for a_1 and $A1 \rightarrow A2$ for a_2); and (2) the current states and the unexecuted path from the

Executor, to the dummy simulation. We run the dummy simulation with the inputs and assume there are no delays. Since all agents are at the start. After the dummy simulation, the commits remain unchanged, which means the dummy simulation outputs: (1) the path $A1 \rightarrow A2$ for a_1 and $B3 \rightarrow B2$ for a_2 to the Executor; and (2) the start locations $A2$ and $B2$ are sent to the Planner. Then during the execution (timestep 0 to timestep 1), the Planner is planning for these two agents from $A2$ and $B2$. At timestep 1, in the Executor, a_1 is at $B3$ due to delays and a_2 is at $A2$. A part of a_1's path, which is $B3 \rightarrow B2 \rightarrow B1$, has not been executed. In the Planner, the first actions are $B2 \rightarrow B1$ for a_1 and $A2 \rightarrow B2$ for a_2. We then input them into the dummy simulation. In the simulation, we simulate the path $B3 \rightarrow B2 \rightarrow B1$ for a_1 (combine the unexecuted path in the Executor and next $k = 1$ action in the Planner) and $A2 \rightarrow B2$ for a_2 with no delays. Following MCP rules, a_2 needs to stay to let a_1 traverse $B2$ first. Therefore, after simulation, the path becomes $B3 \rightarrow B2 \rightarrow B1$ for a_1 and $A2 \rightarrow A2 \rightarrow B2$ for a_2. Now the simulation outputs: (1) the first actions from the simulated path ($B3 \rightarrow B2$ for a_1 and $A2 \rightarrow A2$ for a_2) to the Executor; and (2) the start locations sent to the Planner, which are $B3$ and $A2$ for these two agents. The same concurrent planning and execution then simulate process repeats until being interrupted. Note without this simulation, the Planner will assume agents start with a_1 at $B1$ and a_2 at $B2$, which is incorrect as the agents cannot reach these two locations in the next execution window.

Instantiations of PIE-D

In this section, we discuss different possibilities to instantiate PIE-D, including execution policies and planners.

Execution Policy Choices We consider execution policies that are fast and able to react in real-time while agents are moving. The choices of execution policies are as follows:

MCP: Minimal Communication Policy (Ma, Kumar, and Koenig 2017) dynamically re-schedules agents' move actions in the event of delays. MCP guarantees successful completion of an initially feasible plan, but introduces unnecessary waits to maintain dependencies between actions. For instance in Scenario (b) of Figure 1, to ensure the complete execution of the given MAPF solution, the undelayed a_2 needs to wait for a_1, which has a higher priority in the visiting order of location $B2$, and when the higher priority a_1 gets delayed, both agents are delayed (a_2 cannot reach $B2$ until a_1 leaves $B2$). In concurrent planning and execution, agents have the opportunity to replan when they deviate from the planned paths. Thus, strictly maintaining action dependencies is not essential.

PIBT-I: (Okumura, Tamura, and Défago 2021) use time-independent paths as a hint to attempt to plan for agents to follow the paths. Given a MAPF plan, PIBT-I extracts a time-independent path p'_i for each agent a_i that omits any waits. Then at each timestep, PIBT plans the next actions following a priority. For each agent a_i, it attempts to follow p'_i by moving towards a location $v \in p'_i$ selected as follows. Given agent a_i currently at location c, (1) if $c \in p'_i$ and j is the index of c on p'_i, indicating $c = p'_i[j]$, $v = p'_i[\min(j + 1, |p'_i| - 1)]$ is the location after index j, any location, except the goal location of a_i, at or before j are then deleted from p'_i, (2) otherwise, v is chosen as the nearest location on p'_i to c. When the desired location of a higher-priority agent is occupied by a lower-priority agent, the lower-priority agent temporarily inherits the priority of the higher-priority agent to move first to give way to the higher one. If the lower-priority agent has no choice but to wait, the algorithm backtracks to the higher-priority agent to look for other available actions. This approach does not maintain the action dependencies in π, which can alleviate the unnecessary waiting introduced by MCP-like approaches, such as Scenario (b) in Figure 1 (a_2 may enter $B2$ before a_1). However, directly following a time-independent path may cause large differences between the execution results and the original plan. For example, the wait actions in the original plan may be essential to avoid future conflicts with other agents. Simply Removing them leads to potential conflicts. In this case, agents may be pushed away from their path to avoid collisions, and the planned future paths may become useless, as agents end up far from their expected next window start locations. Moreover, more agents may require replanning in the next execution phase.

PIBT-D: To take more advantage of the time-dependent paths in the current solution, we propose to use PIBT to follow the original *time-dependent plan* (PIBT-D) to move instead of a time-independent plan and let agents with less delay choose to move first. That is, at each timestep t, PIBT will first prioritise agents based on the number of delays occured. This means an agent with a smaller number of delays can choose its next action first. Then when deciding the next action for that agent, this policy will move agents to the location that is nearest to the location on t in their time-dependent plan. By doing so, the execution has fewer agents affected by delayed agents and for those are not affected, they follow the original plan as much as possible.

Planner Choices We use the planner in PIE (Zhang et al. 2024), which guarantees to always return a feasible solution within a given execution time and reports the highest throughput against other frameworks. PIE leverages a fast solver to quickly compute a solution and uses an anytime solver to optimise the uncommitted paths during execution. In PIE, if an agent arrives at its goal within the commit window, the planner directs it to a dummy goal and assumes it disappears [1] after the window. This ensures the feasibility of the commitment if future goals are not revealed to planner. In this work, we use the same approach but direct agents to subsequent goals instead since we always reveal enough goals. PIE assumes perfect executions. However, when delay events occur, the planner needs to replan for agents affected by delays. We consider the following planners in PIE-D:

Replan Affect: Similar to the approach proposed by Zhang et al. (2024), we modify it to replan agents with new goals and those affected by delays, known as Replan Affect.

[1] *Disappearance* means the agent disappears from the plan, but not the map, since it will always get a goal and be replanned for the next commit window. PIE directs agents to a dummy goal because it assumes only one task is revealed to each agent.

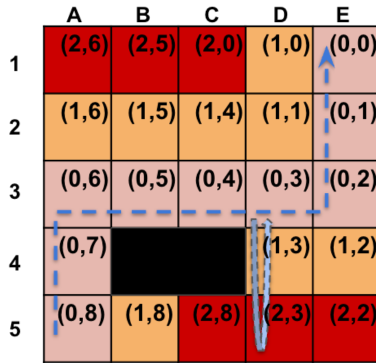

Figure 3: An example of the computed Guidance Heuristic. Suppose the original path from A5 to E1 is: $A5 \rightarrow A4 \rightarrow A4 \rightarrow A3 \rightarrow B3 \rightarrow C3 \rightarrow D3 \rightarrow D4 \rightarrow D5 \rightarrow D4 \rightarrow D3 \rightarrow E3 \rightarrow E2 \rightarrow E1$. After removing wait actions and the loop (marked as the light blue curve on the map), the guidance path is $A5 \rightarrow A4 \rightarrow A3 \rightarrow B3 \rightarrow C3 \rightarrow D3 \rightarrow E3 \rightarrow E2 \rightarrow E1$ (marked as dashed deep blue on the map). Then the heuristic is a pair, with the first value on each grid cell indicating the distance to the nearest guidance location, and the second value indicating the distance from the nearest guidance location to the goal on the guidance path. When calculating heuristics, the heuristic is the sum of the two values, tie-breaking on the first value.

Replan Affect reports high throughput when only replanning for agents with new goals. However, when delays occur, more agents are required to replan, which consumes significantly longer runtime. Thus, this approach suffers from timeout failures under stringent time constraints in online problems, especially as the number of affected agents grows.

Simulate then Replan: Since approaches like MCP ensure the feasible execution of a plan, the planner first simulates the execution of the existing plan using MCP, assuming no additional delays. The simulation provides the planner with a feasible plan that has mitigated the influence of delays (Li et al. 2021a). The planner then replans agents with new goals. This method reduces the overall computational load by avoiding replanning for delayed agents. However, since MCP may introduce unnecessary waits to agents affected by delays, the simulated plan may be inefficient. Additionally, this approach only works if the execution policy in the Executor is complete and action dependencies are preserved.

Replan All with Partial Solution as Guidance: Another way to quickly obtain a collision-free solution is to use a faster and more scalable solver. In Zhang et al. (2024), the authors use LaCAM* (Okumura 2023) as a replanner to replan for all agents, referred to as *Replan All*. LaCAM* offers the advantages of rapid planning and high scalability, but it may suffer from poor solution quality, and requires additional time for improvement. More importantly, *Replan All* plans for all agents from scratch and discards the uncommitted paths, in which the planner spent significant effort to improve quality. To mitigate this, we use paths from the previous planning phase (optimised but outdated due to delays) as guidance to warm-start the search. This way the search

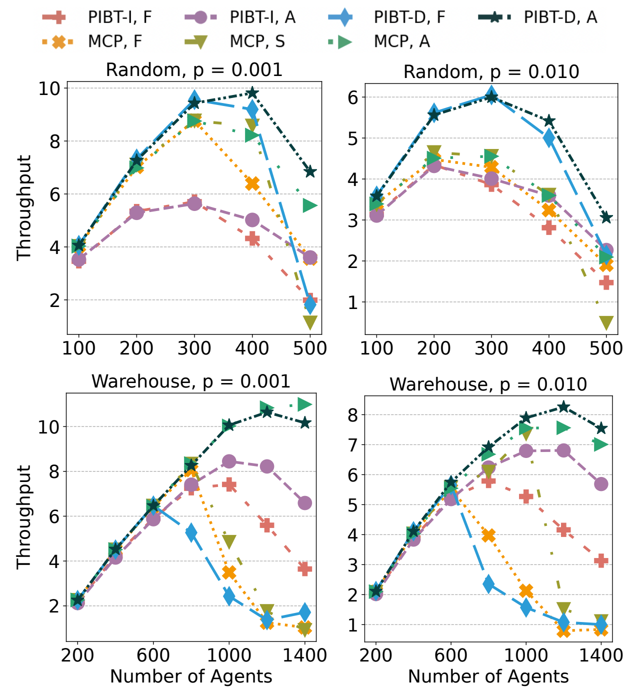

Figure 4: Throughput of differing Replanning (MCP, PIBTI and PIBTD) and After-Goal strategies (**F**: Replan Affect, **S**: Simulate then Replan and **A**: Replan All with Partial Solution as Guidance). Note **S** is only compatible with MCP. p is the delay probability.

quickly returns high-quality solutions. The guidance path is a time-independent path constructed from the existing uncommitted paths. Figure 3 illustrates how the heuristic table is computed. We modify LaCAM* to let each agent follow a heuristic indicating the sum of the distance to the nearest location on a guidance path and the distance from this location to the goal along the guidance path. Following this, LaCAM* generates paths that closely align with the previous uncommitted path and preserves prior search efforts as much as possible. A similar approach is employed in (Chen et al. 2024), where the authors compute traffic flows to evaluate congestion and utilise it to compute guidance heuristic.

Experiments

We implement our framework in C++[2] on top of PIE (Zhang et al. 2024). The experiments are conducted on a cloud instance with 32GB RAM, 16 AMD EPYC-Rome CPUs. We run experiments on four maps from different domains using grid-based Multi-Agent Path Finding (MAPF) benchmarks sourced from (Sturtevant 2012). These maps are named *random-32-32-10* (referred to as Random), *warehouse-10-20-10-2-1* (referred to as Warehouse), *ht_mansion_n* (referred to as Game), and *Paris_1_256* (referred to as City). For each map, we generate instances for different number of agents from 100 to 500 with increments of 100 for Ran-

[2]Code and benchmark instances are available at https://github.com/YueZhang-studyuse/LMAPF-delay

Success Rate (%)

Replan Approaches	m						
	200	400	600	800	1000	1200	1400
F	100	99.5	97	35.5	2.2	1.33	0.68
S	100	100	99.5	97.5	89.44	22.01	11.11
A	100	100	100	100	100	100	100
A_N	100	100	100	100	100	100	100

Average Runtime (s)

m	Replan Approaches			
	F	S	A	A_N
200	$0.016_{\pm 0.004}$	$0.013_{\pm 0.004}$	$0.213_{\pm 0.029}$	$0.015_{\pm 0.001}$
400	$0.086_{\pm 0.246}$	$0.028_{\pm 0.009}$	$0.040_{\pm 0.004}$	$0.032_{\pm 0.002}$
600	$0.362_{\pm 0.566}$	$0.095_{\pm 0.291}$	$0.068_{\pm 0.017}$	$0.053_{\pm 0.002}$
800	$2.272_{\pm 1.071}$	$0.215_{\pm 0.545}$	$0.094_{\pm 0.019}$	$0.083_{\pm 0.006}$
1000	$2.948_{\pm 0.35}$	$0.544_{\pm 0.922}$	$0.134_{\pm 0.048}$	$0.123_{\pm 0.006}$
1200	$2.977_{\pm 0.23}$	$2.503_{\pm 0.978}$	$0.180_{\pm 0.067}$	$0.186_{\pm 0.015}$
1400	$2.981_{\pm 0.23}$	$2.779_{\pm 0.696}$	$0.238_{\pm 0.096}$	$0.254_{\pm 0.024}$

Average Solution Cost ($\times$1000)

m	Replan Approaches			
	F	S	A	A_N
200	$1.123_{\pm 0.973}$	$1.120_{\pm 0.994}$	$1.137_{\pm 0.101}$	$1.270_{\pm 0.123}$
400	$2.266_{\pm 0.185}$	$2.274_{\pm 1.879}$	$2.334_{\pm 0.185}$	$2.860_{\pm 0.289}$
600	$3.467_{\pm 0.265}$	$3.486_{\pm 2.653}$	$3.667_{\pm 0.282}$	$4.733_{\pm 0.376}$
800	-	$4.792_{\pm 5.972}$	$5.076_{\pm 0.672}$	$7.597_{\pm 0.935}$
1000	-	$6.288_{\pm 0.985}$	$6.698_{\pm 1.036}$	$11.96_{\pm 0.644}$
1200	-	-	$9.038_{\pm 2.595}$	$20.09_{\pm 2.311}$
1400	-	-	$11.70_{\pm 3.082}$	$28.65_{\pm 4.287}$

Table 1: Success rate (%), average runtime (s) and average solution cost of different replan approaches with $p = 0.01$ in Warehouse. The execution policy is MCP. m is the number of agents. For each number of agents in each map, we average the planning time and solution cost of the replanner (path improving not included) in each commit. Replanning fails if it exceeds the 3s runtime timelimit. The solution cost is the sum of the path length (number of actions in a path) for each agent, and we only include the cost from the success replan. '-' means the success rate is less than 80%. The subscripted value of the number indicates the standard deviation. $\mathbf{A}_N$ denotes Replan All using LaCAM* with distance-to-goal heuristics, **F, S, A** are the same as Figure 4.

dom, 200 to 1400 with increments of 200 for Warehouse and Game, and 1000 to 5000 with increments of 1000 for City. The start and goal locations are randomly positioned.

For each timestep, we sample for each agent with a given delay probability p ($p \geq 0$) to be delayed, and if an agent is delayed, the length of delay ($l \in [d_{min}, d_{max}]$ and $d_{min}, d_{max} > 0$) is uniformly sampled from a range. We set $l \in [1, 10]$ for all the experiments. For a fair comparison, we sample once for each instance and use the same delays when comparing different approaches. For each instance, we run LMAPF evaluations for each algorithm once for 600 timesteps of LMAPF simulation.

Experiment 1: Comparison of different strategies. We first show the effectiveness of different strategies in our framework. For strategies, we compare the proposed three execution policies combined with different path replanners. For the Simulate then Replan strategy, we only use MCP since it is the only complete execution policy. We set commit length $k = 3$ and test on Random and Warehouse with small and large delays ($p = 0.001, 0.01$).

As shown in Figure 4, Replan All with Partial Solution as Guidance achieves the best throughput and scalability with different execution policies, while the other two methods cannot scale with large agent team size. This is because Replan All always succeeds, while the other two approaches may frequently time out during execution when scaling up.

We use MCP execution policy and warehouse map with delay probability $p = 0.01$ as an example to show the performance of different replanners. Table 1 shows the average runtime and success rate of the commits in different numbers of agents. We also present the results of LaCAM* with distance-to-goal heuristics (denoted as A_N) to highlight our approach with guidance heuristics. The results show that compared to A_N, Replan All with the previous path as guidance helps maintain the previously optimised path as much as possible, resulting in up to two times improvement on the solution cost compared to A_N. The other two appraoches have lower solution cost and runtime when the team size is small. However, when scaling up, they often time out, while Replan All has a smaller amount of runtime and a high success rate across all instances.

For execution policies, PIBT-D is found to achieve the highest throughput in dense environments or when the delay probability is large, while MCP performs slightly better in larger maps and when less delay happens (Warehouse with $p = 0.001$ delays). With larger delay probabilities, MCP performance drops, as more agents are affected for following the dependencies. This is because, in such situations, MCP helps maintain the original path while not being affected by unnecessary delays due to small delay probabilities.

Experiment 2: Performance against baselines. We compare the throughput of our framework with PIBT-D, Replan All against two baselines:

Baseline 1: we use PIBT with instant delay feedback. That is, at every time step, we run PIBT to decide the next actions for each agent with the current state. Then we execute the next actions with delays and update the current state for each agent based on the execution results so that PIBT is planning directly with the delay result from the Executor.

Baseline 2: We use the original PIE planner with the simple delay handling method in (Hönig et al. 2019), in which agents follow MCP execution to fully execute to a commit cut, i.e., locations extracted from the Dependency graph. We set the commit cut to be the last location for each agent on the committed path.

For Baseline 2 and our approach, we vary the commit length: either short commit ($k = 3$) or long commit ($k = 10$). For delay probabilities, we set $p = 0.001, 0.004, 0.007, 0.01$.

As shown in Figure 5, for Baseline 1, the solution is often far from optimal even with instant delay feedback. Baseline 2, which tries to synchronise agents to their commit cuts with the given optimised path, is worse than Baseline 1. This is because fully executing the commit cuts causes more delays to be introduced while waiting for the delayed agent. It is worth noting that in most cases smaller commits are better, as we can get more frequent feedback and replan quicker

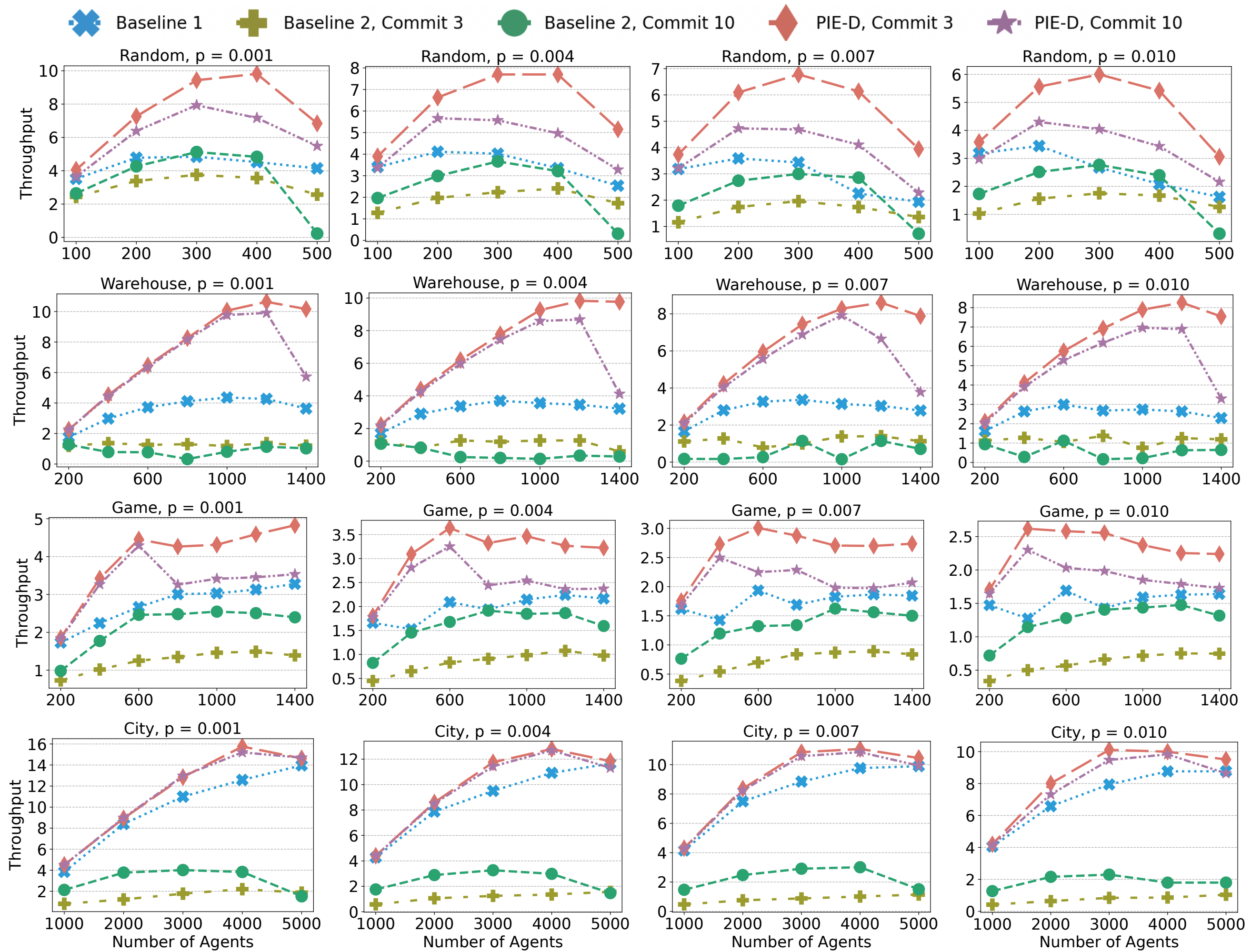

Figure 5: Throughput of our framework against baselines.

except for Baseline 2, as agents with long delays will lead to more waits in a small commit window, i.e., all other agents wait at the last locations for one delayed agent.

The results show our framework significantly outperforms the two baselines for different delay probabilities. We also notice an odd throughput drop-down with 800 agents and an increase after that in the Game map. This is because, at that point, agents start to be congested, and the throughput gains less benefit from increasing the number of agents, while the delay effects in execution become larger. As the delay probability increases, the throughput keeps decreasing after scaling to 800 agents. In addition to this, when scaling to 5000 agents in City map, Replan All starts to time out in most commits, which decreases the performance of PIE-D to be the same or even worse than Baseline 1. This is because when timing out, PIE-D with Replan All has no chance to improve the path.

Conclusion and Future Work

In this work, we consider desynchronisation between planning and execution, which often occurs in multi-agent systems and is caused by unexpected delays during execution. We study this problem in LMAPF, our main contribution is a new approach to planning and execution which has the advantages of both. We use reactive policies and simulated predictions to reduce the de-synchronisation gap, and we employ concurrent online replanning to restore feasibility and to continually re-optimise the global objective. Our results show that we significantly mitigate the impact of delays with up to three times throughput improvements compared to existing delay-handling approaches in LMAPF.

Key takeaways include: (1) In online multi-agent settings with delays, an incumbent plan is almost always out of sync with execution. (2) Even under disruption, there is significant benefit from drawing upon the previous (now infeasible) incumbent. (3)Restoring feasibility via concurrent replanning is always better than conventional reactive policies.

Future work may explore more advanced heuristics or machine learning techniques to predict future delays and adapt plans proactively to improve overall performance. Additionally, instantiating the Planner to consider windowed planning approaches may be beneficial for further improvement.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust multi-agent path finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, 2–9.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67: 549–579.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic flow optimisation for lifelong multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20674–20682.

Chen, Z.; Harabor, D. D.; Li, J.; and Stuckey, P. J. 2021. Symmetry breaking for k-robust multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12267–12274.

Feng, Y.; Paul, A.; Chen, Z.; and Li, J. 2024. A Real-Time Rescheduling Algorithm for Multi-robot Plan Execution. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 201–209.

Ho, F.; Geraldes, R.; Goncalves, A.; Rigault, B.; Sportich, B.; Kubo, D.; Cavazza, M.; and Prendinger, H. 2022. Decentralized Multi-Agent Path Finding for UAV Traffic Management. *IEEE Trans. Intell. Transp. Syst.*, 23(2): 997–1008.

Ho, F.; Salta, A.; Geraldes, R.; Goncalves, A.; Cavazza, M.; and Prendinger, H. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, 131–139.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters*, 4(2): 1125–1131.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. 2011. Anytime motion planning using the RRT. In *2011 IEEE international conference on robotics and automation*, 1478–1483. IEEE.

Kottinger, J.; Geft, T.; Almagor, S.; Salzman, O.; and Lahijanian, M. 2024. Introducing Delays in Multi Agent Path Finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, 37–45.

Laurent, F.; Schneider, M.; Scheller, C.; Watson, J.; Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Makhnev, K.; Svidchenko, O.; et al. 2021. Flatland competition 2020: MAPF and MARL for efficient train coordination on a grid world. In *NeurIPS 2020 Competition and Demonstration Track*, 275–301. PMLR.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10256–10265.

Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2021a. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *Proceedings of the international conference on automated planning and scheduling*, volume 31, 477–485.

Li, J.; Lin, E.; Vu, H. L.; Koenig, S.; et al. 2023. Intersection coordination with priority-based search for autonomous vehicles. In *AAAI*, volume 37, 11578–11585.

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

Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Ma, H.; Li, J.; Kumar, T. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 837–845.

Okumura, K. 2023. Improving LaCAM for scalable eventually optimal multi-agent pathfinding. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 243–251.

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

Okumura, K.; Tamura, Y.; and Défago, X. 2021. Time-independent planning for multiple moving agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11299–11307.

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 *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2): 144–148.

Su, Y.; Veerapaneni, R.; and Li, J. 2024. Bidirectional Temporal Plan Graph: Enabling Switchable Passing Orders for More Efficient Multi-Agent Path Finding Plan Execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17559–17566.

Zhang, Y.; Chen, Z.; Harabor, D.; Le Bodic, P.; and Stuckey, P. J. 2024. Planning and Execution in Multi-Agent Path Finding: Models and Algorithms. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 707–715.