Towards Holistic Approach to Robust Execution of MAPF Plans

David Zahrádka[1,2][a], Denisa Mužíková[1], Miroslav Kulich[2][b], Jiří Švancara[3][c], and Roman Barták[3][d]

[1]*Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic*

[2]*Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic*

[3]*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic*

david.zahradka@cvut.cz, muzikden@fel.cvut.cz, kulich@cvut.cz, {svancara, bartak}@ktiml.mff.cuni.cz

Keywords: Path-Planning, Multi-Agent Environment, Plan Execution, Action Dependency Graph, Replanning.

Abstract: Multi-agent path finding (MAPF) deals with the problem of navigating a set of agents in a shared environment to reach their destinations without collisions. Even if the plan is collision-free, some delay during plan execution may lead to collision of agents if they execute the plans blindly. In this position paper, we discuss the concept of robust execution of MAPF plans by exploiting an action dependency graph. We suggest how to evaluate the effect of delay by computing a slack-like value for not yet visited locations, and we propose a three-layer architecture – retiming, rescheduling, and replanning to handle delays effectively.

1 INTRODUCTION

Coordinating a fleet of moving agents is an important problem with practical applications such as warehousing (Wurman et al., 2008), airplane taxiing (Morris et al., 2016), or traffic junctions (Dresner and Stone, 2008). *Multi-Agent Path Finding* (MAPF) is an abstract model of this coordination problem, where we are looking for collision-free paths for agents moving over a graph. The problem of finding an optimal MAPF solution has been shown to be NP-hard for a wide range of cost objectives (Surynek, 2010). In MAPF research, two classes of methods can be identified. Methods from the first class do not guarantee optimality (or even completeness), but are computationally nondemanding and scale well to a large number of agents, for example, WHCA* (Silver, 2005), SIPP (Phillips and Likhachev, 2011), MAPF-LNS (Li et al., 2021a), LACAM* (Okumura, 2023). Provably optimal and bounded suboptimal algorithms form the second class. Prominent optimal algorithms are, for example, CBS (Conflict-Based Search) (Sharon et al., 2015; Li et al., 2019), or approaches transforming MAPF into SAT (Boolean Satisfiability) (Barták et al., 2017), ASP (Answer Set Programming) (Erdem et al., 2013), and MIP (Mixed Integer Programming) (Yu and LaValle, 2016). Bounded suboptimal solvers are often based on CBS - ECBS (Barer et al., 2014) and its new variant, EECBS (Li et al., 2021b) or variants of SAT solvers (Surynek, 2020).

When real robots execute the found plans, the robots might be delayed due to unexpected circumstances such as mechanical problems, localization uncertainties, unprecise control, or encounters with unmodelled agents, e.g. humans. These delays lead to a desynchronization of plans, and consequently, the robots may crash into each other or get stuck.

Such situations can be prevented by generating *k-robust plans*, i.e. plans that are collision-free even if any agent is delayed by a limited number of k time steps (Atzmon et al., 2020; Chen et al., 2021). Nevertheless, these plans are typically unnecessarily longer than non-robust plans and still lead to a collision if some agent is delayed more than k time steps.

A more realistic approach is presented by Ma et al. (2017). The authors assume probability of a delay for each agent's action, propose CBS-based solver which minimizes an approximated expected makespan, and show that real makespans of executed plans are lower than of plans generated by standard solvers not assuming delays.

Wagner et al. (2022) introduce the Space-Level Conflict-Based Search, an ECBS-based planner that provably reduces the number of coordination actions, that is, locations where the decision on the passing

[a] https://orcid.org/0000-0002-7380-8495
[b] https://orcid.org/0000-0002-0997-5889
[c] https://orcid.org/0000-0002-6275-6773
[d] https://orcid.org/0000-0002-6717-8175

Zahrádka, D., Mužíková, D., Kulich, M., Švancara, J. and Barták, R.
Towards Holistic Approach to Robust Execution of MAPF Plans.
DOI: 10.5220/0013319700003890

In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 1*, pages 624-631
ISBN: 978-989-758-737-5; ISSN: 2184-433X

order of two agents must be made.

However, an arbitrary good plan is not guaranteed to be realized perfectly, and thus monitoring of plan execution has to be done to ensure its collision-free and deadlock-free realization. The first attempt to make such a plan executor is MAPF-POST (Hönig et al., 2016). MAPF-POST postprocesses a plan of a MAPF solver and builds in a polynomial time a temporal network, a graph where nodes represent events corresponding to an agent entering a location and edges represent temporal precedences between events. The graph is augmented by additional vertices that provide a guaranteed safety distance between agents. The authors proved that the execution is collision and deadlock-free if it is consistent with the graph.

Gregoire et al. (2017) introduce RMTRACK, a robust multi-robot trajectory tracking, based on similar ideas to Hönig et al. (2016), but in a different terminology: RMTRACK works in a coordination space and ensures that the realized trajectory remains in the same homotopy class as the planned trajectory.

The Action Dependency Graph (ADG) (Hönig et al., 2019) is, similarly to MAPF-POST, a temporal network, but utilizes a precedence relation on actions rather than locations. This makes the graph simpler with less communication among agents and has stronger guarantees on collision-free operation.

Wu et al. (2024) follow up with Wagner et al. (2022) and present the Space-Order CBS that directly plans a temporal graph, and thus minimizes the number of coordination actions explicitly.

Although the mentioned techniques guarantee safe execution, the final quality of realized trajectories can substantially degrade with the increasing frequency of delays. Recent approaches thus investigate possibilities how to *reschedule* agent's actions, i.e., switch the order the agents passing a given location.

Berndt et al. (2020) and Berndt et al. (2024) present Switchable Action Dependency Graph (SADG) by creating a reverse to each dependence between actions of different agents. Rescheduling, the decision whether to take the original or reverse for each action is then formulated as a mixed integer linear program that is solved in a closed-loop feedback scheme.

Similarly, Zahrádka et al. (2023) formulate rescheduling as a modification of the job-shop scheduling problem and utilize the Variable Neighborhood Search metaheuristic to solve it.

Another approach to repair the plan is to introduce additional delays. Kottinger et al. (2024) formulate the optimization problem aiming to find a minimal number of additional delays in the reduced graph built as a superposition of the original paths. The problem is solved by an arbitrary MAPF solver substantially faster than replanning from scratch.

Finally, Feng et al. (2024) propose the Switchable Edge Search. The approach aims to find a new temporal graph based on the given SADG making use of an A*-style algorithm.

The disadvantage of the above-mentioned approaches is that they run the optimization *during* execution. To avoid this, a Bidirectional Temporal Plan Graph (BTPG) (Su et al., 2024) allowing one to switch these orders during execution based on a "first-come-first-served" manner as build *before* the execution. This is done by consecutively checking whether edges indicating dependencies between actions of different agents can be transformed into a bidirectional pair without causing an oriented cycle in the BTPG.

Despite the enormous activity in finding a robust planning & execution framework for MAPF in recent years, several open questions still need to be answered. One of the most interesting open questions is when to reschedule/replan and whether rescheduling still provides a good quality solution or whether it is a good moment for replanning. This position paper attempts to take the first step in this direction. Specifically, we introduce an architecture that monitors the execution process and decides when it deviates significantly from the plan, so it is beneficial to reschedule/replan. This architecture exploits an action dependency graph to calculate the slack of actions, which is then used to decide which type of execution modification to select.

2 BACKGROUND

2.1 Multi-Agent Path Finding

The *Multi-Agent Path Finding instance* (MAPF instance) $\mathcal{M}$ is a pair $\mathcal{M} = (G, A)$, where G is a graph $G = (V, E)$ representing the shared environment and A is a set of mobile agents. Each agent $k \in A$ is defined as a pair $k = (s_k, g_k)$, where $s_k \in V$ is a starting location of agent k and $g_k \in V$ is a goal location of agent k.

The task is to find a valid plan π_k for each agent $k \in A$ such that it is a valid path from s_k to g_k. Time is considered discrete and at each timestep i, an agent can either wait in its current location or move to a neighboring location. Therefore, the plan is a sequence of *move* or *wait* actions. At the end of an agent's plan, the agent remains in its goal location and does not disappear. Furthermore, we require that each pair of plans π_k and $\pi_{k'}$, $k \neq k'$ is collision-free. Based

on MAPF terminology (Stern et al., 2019), there are five types of possible collisions we may want to forbid (see Fig. 1). In this work, we do not avoid following conflicts, since they can be executed safely, and edge conflicts are automatically resolved by preventing vertex conflicts.

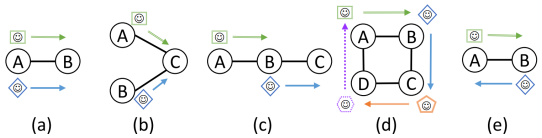


Figure 1: Conflicts between two or more agents. (a) *edge conflict*, (b) *vertex conflict*, (c) *following conflict*, (d) *cycle conflict*, (e) *swapping conflict*. Figure taken from Stern et al. (2019).

The quality of the plan is measured by the lengths of the plans $|\pi_k|$, i.e. the number of actions each agent needs to perform until reaching its goal location. The two most often used cost functions in the literature are *makespan* defined as the maximum of the plan lengths ($T_{max} = \max_{k \in A} |\pi_k|$) and *sum of costs* (or *flowtime*) as the sum of the plan lengths ($T_{sum} = \sum_{k \in A} |\pi_k|$).

2.2 Action Dependency Graph

The *Action Dependency Graph* (ADG) (Hönig et al., 2019) is a directed acyclic graph $G_{ADG} = (V_{ADG}, E_{ADG})$, where V_{ADG} are the move actions in agents' plans and E_{ADG} are directed edges that represent dependencies between actions. Each action $a_i^k \in V_{ADG}$ is a tuple $(p_i, p_{i'})$ that means that an agent k is moving from position p_i in the time step i to position $p_{i'}$ in the next time step. If $(a_i^k, a_{i'}^k) \in E_{ADG}$, there is a temporal precedence between these two actions: k can start executing $a_{i'}^k$ only after a_i^k is completed.

The edges E_{ADG} are divided into two categories. Type 1 edges (E^1) represent the temporal precedence between actions within the plan of a single agent: it may start moving from a specific position only after it has finished moving into it. Type 2 edges (E^2) indicate dependencies between actions of different agents: an agent may start moving into a position only after another agent, scheduled to be there earlier, finished moving out. An example instance and the resulting ADG can be seen in Fig. 2.

During execution, agents report whenever they finish an action, and a robust execution manager keeps track of the execution state by marking E_{ADG} as completed. Once an action has all its incoming edges marked as complete, it is assigned to an agent. In this way, ADG guarantees safe execution of any feasible 1-robust MAPF plan even if the execution of some or all actions is arbitrarily delayed, as long as the delay is finite.

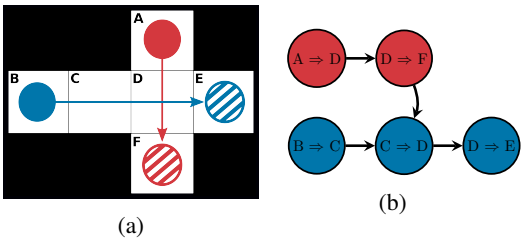


Figure 2: Example MAPF solution (a) and its ADG (b). Filled circles are agents and striped circles are their goals.

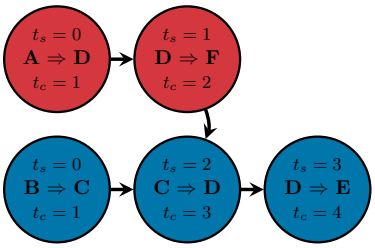


Figure 3: ADG with estimated starting times $t_s(a)$ and completion times $t_c(a)$.

3 EXECUTION ARCHITECTURE

In this section, we propose a *holistic architecture* that enables i) using robust execution methods to safely execute plans; ii) monitor the execution progress and delays of robots, and iii) decide when delays in execution are significant enough that it is beneficial to reschedule or even replan. Everything is performed during execution on the robust execution layer.

3.1 Execution Monitoring

The delays that robots accumulate during execution do not endanger the safety of the plan as long as a robust execution method is used. However, each delay may increase the length of the execution for two reasons. The first reason is prolonging of the robot's own plan execution. Execution makespan may be increased if the robot with the longest plan is delayed or if another robot is delayed sufficiently. When measuring flowtime, it is increased with each delay of any robot.

The second reason is due to the interactions between the robots. If there is a crossroad through which multiple robots pass, the robots have a strict order in which they do so. This ordering is given by the MAPF plan and is represented by the Type 2 edges in G_{ADG}. Because of this, a delayed robot may in turn delay other robots. In extreme cases, this may result in a cascade that includes the entire robotic fleet, increasing the impact of even a single delayed action.

The length of the execution can be estimated by

input : G_{ADG}
output: G_{ADG} with computed $\hat{t}_s$ and $\hat{t}_c$

begin
  for $i = 0 \rightarrow T_{max}$ **do**
    $V(i) \leftarrow$ all actions at time step i
    for $a' \in V(i)$ **do**
      $A^P = \{a : \forall a \in V_{ADG} : e(a,a') \in E_{ADG}\}$
      if $A^P == \emptyset$ **then**
        $\hat{t}_s(a') = 0$
        $\hat{t}_c(a') = \hat{t}_x(a')$
        continue
      end
      $\hat{t}_s(a') = max_{a \in A^P}(\hat{t}_c(a)) + 1$
      $\hat{t}_c(a') = \hat{t}_s(a') + \hat{t}_x(a')$
    end
  end
end

Algorithm 1: Computing estimates of action start times $\hat{t}_s$ and completion times $\hat{t}_c$.

exploiting the G_{ADG} using Algorithm 1. We can obtain the expected length of execution of the actions $\hat{t}_x(a)$ by constructing a model of the robot's movement capabilities. The expected completion of the action a is then equal to the sum of its estimated start $\hat{t}_s(a)$ and its estimated length $\hat{t}_x(a)$: $\hat{t}_c(a) = \hat{t}_s(a) + \hat{t}_x(a)$. We augment V_{ADG} with this value and proceed to the following action, which can start only after all preceding actions are finished: $\hat{t}_s(a') = \max(\hat{t}_c(a) : \exists(a,a') \in E_{ADG})$. This is repeated until we have $\hat{t}_c(a) : \forall a \in V_{ADG}$. In this way, it is possible to obtain the estimated execution time $\hat{T}_C$ of the plan before starting the execution. Ideally, this should be equal to the cost of the plan, as given by the MAPF solver. An example ADG with estimated action completion times can be seen in Fig. 3.

However, $\hat{T}_C$ provides only the lower bound of the plan's execution length T_C, as delays during execution will increase the real execution time T_C. To improve the estimate as execution progresses, G_{ADG} can be updated as actions are being executed. The same procedure that was used for initial estimation can be used to update the estimate by replacing $\hat{t}_c(a)$ with the *real action completion time* $t_c(a)$. After finishing an action, it's corresponding vertex in V_{ADG} is updated with $t_c(a)$. Then, we follow all outgoing edges in G_{ADG} to update $\hat{t}_s(a)$ and $\hat{t}_c(a)$ of all dependent future actions as well. This propagates not only the information about the delay into the rest of the robot's plan thanks to E^1, but also to all other robots with which it interacts using E^2, while not interfering with the estimated times of independent actions. By doing this, we get an update on the plan's estimated completion time $\hat{T}_C$. The above process is called *retiming* as it updates the start times of actions based on the current situation but without changing the plans (paths).

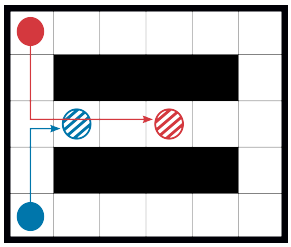


Figure 4: If the red agent is significantly delayed, it should take the longer way around.

3.2 Resolving Delays

As robots progressively deviate from the original plan due to delays, a new plan that better reflects the current configuration might decrease the execution time T_c. That is true even if the original plan is optimal. We can obtain such a plan by replanning. Consider the example in Fig. 4. In the optimal plan, both robots drive through the corridor, and since they cannot fit next to each other, one robot must drive earlier than the other. If the robot that is supposed to enter the corridor first is delayed during execution and no robust execution method capable of retiming was used, the robots may collide. If the execution method was only capable of retiming, the second robot would have to wait to maintain their order as given by the MAPF plan, also getting delayed itself. However, with a long enough delay, it may become beneficial to either *reschedule* to let the second robot drive first or *replan* to let one of the robots use the long detour. In the solution in Fig. 2a, if the red robot is delayed, rescheduling may allow the blue robot to pass first, reducing the total execution time. In the solution shown in Fig. 4, rescheduling would produce the same schedule as the original due to the blue robot blocking the corridor after reaching its goal. If the delayed red robot took a longer detour around the wall, the blue robot could reach its goal faster at a small cost increase for the red robot, reducing T_c.

As robots are progressively delayed, it may happen that another plan would lead to a smaller T_C if selected. The problem is that to know whether such a plan exists or not, we would need to have it. Since MAPF is an NP-hard problem (Surynek, 2010), it is impractical to judge whether replanning would be beneficial by finding a new plan. More so when the state of the execution itself is changing in the meantime. It is therefore crucial to approximate or estimate the benefit that replanning would bring by other, simpler means.

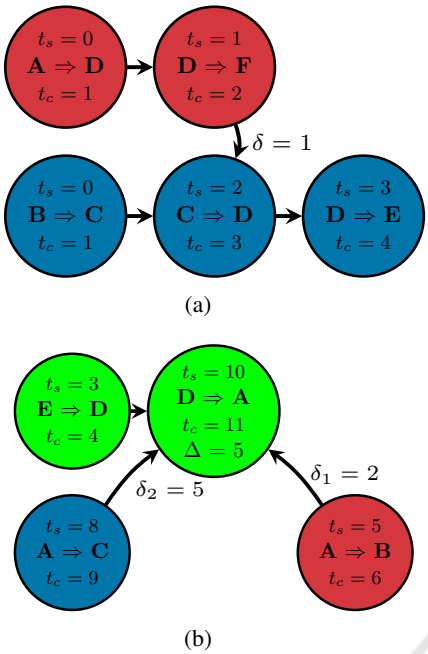


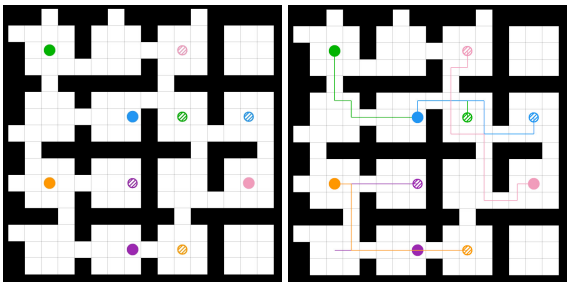
Figure 5: (a) ADG with computed slack δ for a Type 2 edge. (b) Example of Δ computation for a robot (green) that moves into a vertex after two other agents (red and blue) pass through.

For this purpose, we suggest using the edges of G_{ADG}, which provide information about mutual dependencies of actions. The Type 1 edges only contain information about the ordering of specific actions in a single robot's plan. If a robot is completely independent of others, even if it accumulates a significant delay, replanning would not be beneficial. That is because the robot is probably already on its optimal path and, therefore, will arrive at its goal as soon as possible. However, Type 2 edges contain information about interactions between different robots. Therefore, they can be used to express how one robot's delay is affecting the others, who were counting on the delayed robot to be further in the plan than it is. Rescheduling or replanning may help in this case, because these robots may perhaps continue their plans sooner if they were rescheduled in front of the delayed robot, or find another path to their goal.

For every Type 2 edge $E^2 = (a_i^{r_1}, a_j^{r_2})$ between the actions of robots r_1 and r_2 we can calculate a *time reserve*, or *slack*, $\delta(E^2) = t_c(a_i^{r_1}) - t_c(a_{j-1}^{r_2})$. An example can be seen in Fig. 5a and it can be computed after Line 12 in Algorithm 1. It is a simple way to estimate how long r_2 must wait before starting an action due to the temporal dependence on r_1. The $\delta(E^2)$ is interpreted as positive for the vertices on the head end of E^2 (waiting robots – r_2) and negative for the vertices on the tail end (robots that cause waiting – r_1). As execution progresses and the estimates $\hat{t}_c(a) : a \in V_{ADG}$ are updated with the real measured $t_c(a)$, δ is also updated to reflect the current situation, providing an estimate of waiting times in the rest of the plan.

However, simply looking at the time for how long the robots will be waiting may be misleading, since an optimal MAPF plan may require some robots to wait to synchronize the fleet. In other words, it may be beneficial to wait until another robot passes through a corridor and enter afterwards rather than taking a long detour. Therefore, we want to compare the current slack with the *initial slack* δ_0, calculated at the start of the execution, to obtain $\bar{\delta}(E^2) = \delta(E^2) - \delta_0(E^2)$. Each V_{ADG} may have multiple incoming E^2, which means that there is temporal precedence of multiple robots in the same place, and $\bar{\delta}(E^2)$ only captures dependencies between two robots. By computing the slack of an action $\Delta(a) = \min(\bar{\delta}(E^2) : \forall E^2 = (a', a), \exists (a', a) \in E_{ADG})$, it is possible to express whether a robot will be waiting according to the plan or in contrast to it. An example can be seen in Fig. 5b. Decreasing $\Delta(a)$ means that a robot is getting delayed relative to some other robot. If it is negative, it means that a robot that had spare time is now behind its plan, either due to its slowdown, or because someone else is potentially ahead. In either case, it shows that the execution is not proceeding as expected, and interventions may be necessary to mitigate the delays. By computing $\Delta(a)$, we can estimate whether a delayed robot is interfering with the other robots execution, and use this information to decide whether it is beneficial to reschedule or replan without costly computation of alternative plans. The state of the whole execution considering all robots is then expressed by $\Delta^F = \min_{a \in V}(\Delta(a))$.

Using ADG with slack-augmented vertices, we obtain a *holistic* architecture to execution optimization. The execution is safe and robust due to retiming performed by ADG. If some robot is delayed enough and in a way that affects the rest of the fleet, as measured by Δ^F, we can perform rescheduling, which would optimize the execution of the original plan. If Δ^F decreases beyond a certain threshold by significant and possibly wide-spread delays, replanning can be triggered to find a new plan that would allow robots to take different paths. The frequency of both rescheduling a replanning can be controlled by thresholds, which would specify when a delay is too significant for retiming, and, subsequently, for rescheduling.



(a) (b)

Figure 6: Experimental `room-16-16-4` instance (a) and its solution (b). Filled circles are robots, strafed circles of the same color are their goals. Lines in (b) represent optimal paths of robots.

4 PRELIMINARY EXPERIMENTS

We show that by using a temporal G_{ADG} with vertices V_{ADG} augmented with *slack* $\Delta(a)$ it is possible to predict when replanning would be beneficial. The experiments were carried out on a simple MAPF instance with 5 agents seen in Fig. 6a on `room-16-16-4` map, a smaller version of `room-32-32-4` map from the movingAI benchmark (Stern et al., 2019). An optimal solution, as seen in Fig. 6b, was obtained using CBS (Conflict-based Search) (Sharon et al., 2015) which was then executed using a simulator with ADG as a robust execution method. In one experiment, the blue robot was delayed for 5 seconds at some point during execution. In the second experiment, blue and cyan robots were delayed the same way. First, we measured the execution length of the plan with ADG only capable of retiming. Then, we measured the execution length with ADG augmented with slack that was capable of triggering replanning. Finally, we measured how a random replanning policy performs, which replans after a random action is completed. Both methods replanned exactly once during the whole execution and both used CBS to find the new plan. The estimated duration of every action was $\hat{t}_x = 1$ second. We repeated the experiments 5 times for the same delay of the same robots and reported the average execution times. For the random replanning policy, the experiment was repeated 50 times. We assumed that replanning was instant and did not count the time required to find a new plan towards the execution time, pausing the simulation. This approach was used to avoid the need to ensure that the execution was persistent (Hönig et al., 2019), which could influence the measured execution times.

As can be seen in the results reported in Table 1 while retiming ensures safe execution, it yields the worst execution cost both in makespan and in the sum of costs. Replanning at random timesteps improves the execution cost. Using the proposed Δ-based replanning policy improves the execution cost even further since the replanning is triggered at timesteps when it is more beneficial.

Table 1: Measured average execution lengths with a single delayed robot (blue) and with two delayed robots (blue and orange). T_{max} corresponds to the execution length of the slowest agent (makespan) and T_{sum} corresponds to the sum of execution lengths for all robots (sum of costs).

Method	$T_{max}[s]$	$T_{sum}[s]$
Blue delayed		
Retiming	27.58	108.84
Retiming + random replan	24.14	99.79
Retiming + Δ-based replan	21.13	94.58
Blue and orange delayed		
Retiming	27.91	117.95
Retiming + random replan	25.33	112.35
Retiming + Δ-based replan	22.07	103.26

5 CONCLUSIONS

The position paper argues that the current approaches to handling uncertainty during the execution of MAPF plans are unsatisfactory and suggests an integrated holistic architecture that uniformly handles three approaches to managing execution uncertainty: retiming, rescheduling, and replanning. The primary motivation is to execute the plans safely, even if there are disturbances reflected in delays of agents while maintaining plan quality. Based on the notion of slack, the architecture provides a mechanism to decide what type of execution modification should be used. The preliminary experiment indicates that the proposed architecture brings lower execution times even if there are disturbances during plan execution.

ACKNOWLEDGEMENTS

The research was supported by the Czech Science Foundation Grant No. 23-05104S. The work of David Zahrádka was supported by the Grant Agency of the Czech Technical University in Prague, Grant number SGS23/180/OHK3/3T/13. The work of Jiří Švancara was supported by Charles University project UNCE 24/SCI/008. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

REFERENCES

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.

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Seventh Annual Symposium on Combinatorial Search*.

Barták, R., Zhou, N.-F., Stern, R., Boyarski, E., and Surynek, P. (2017). Modeling and Solving the Multi-Agent Pathfinding Problem in Picat. In *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 959–966.

Berndt, A., Duijkeren, N. V., Palmieri, L., and Keviczky, T. (2020). A feedback scheme to reorder a multi-agent execution schedule by persistently optimizing a switchable action dependency graph. *CoRR*, abs/2010.05254.

Berndt, A., Duijkeren, N. V., Palmieri, L., Kleiner, A., and Keviczky, T. (2024). Receding horizon re-ordering of multi-agent execution schedules. *IEEE Transactions on Robotics*, 40:1356–1372.

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

Dresner, K. and Stone, P. (2008). A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research (JAIR)*, 31:591–656.

Erdem, E., Kisa, D. G., Oztok, U., and Schüller, P. (2013). A General Formal Framework for Pathfinding Problems with Multiple Agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Feng, Y., Paul, A., Chen, Z., and Li, J. (2024). A real-time rescheduling algorithm for multi-robot plan execution. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34:201–209.

Gregoire, J., Čáp, M., and Frazzoli, E. (2017). Locally-optimal multi-robot navigation under delaying disturbances using homotopy constraints. *Autonomous Robots 2017 42:4*, 42:895–907.

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*, 4:1125–1131.

Hönig, W., Kumar, T. K., Cohen, L., Ma, H., Xu, H., Ayanian, N., and Koenig, S. (2016). Multi-agent path finding with kinematic constraints. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26:477–485.

Kottinger, J., Geft, T., Almagor, S., Salzman, O., and Lahijanian, M. (2024). Introducing delays in multi-agent path finding. In *The International Symposium on Combinatorial Search*.

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*, pages 4127–4135, Montreal, Canada. International Joint Conferences on Artificial Intelligence Organization.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., and Koenig, S. (2019). Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6087–6095.

Li, J., Ruml, W., and Koenig, S. (2021b). EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14):12353–12362.

Ma, H., Kumar, T. K., and Koenig, S. (2017). Multi-agent path finding with delay probabilities. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31:3605–3612.

Morris, R., Pasareanu, C., Luckow, K., Malik, W., Ma, H., Kumar, T., and Koenig, S. (2016). Planning, Scheduling and Monitoring for Airport Surface Operations. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 608–614.

Okumura, K. (2023). Improving lacam for scalable eventually optimal multi-agent pathfinding. In Elkind, E., editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 243–251. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Phillips, M. and Likhachev, M. (2011). SIPP: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219:40–66.

Silver, D. (2005). Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 1(1):117–122.

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, pages 151–158.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:17559–17566.

Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1261–1263.

Surynek, P. (2020). Bounded Sub-optimal Multi-Robot Path Planning Using Satisfiability Modulo Theory (SMT) Approach. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11631–11637, Las Vegas, NV, USA. IEEE.

Wagner, A., Veerapaneni, R., and Likhachev, M. (2022). Minimizing coordination in multi-agent path finding

with dynamic execution. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 18:61–69.

Wu, Y., Veerapaneni, R., Li, J., and Likhachev, M. (2024). From space-time to space-order: Directly planning a temporal planning graph by redefining cbs. *arXiv*, 2404.15137.

Wurman, P. R., D'Andrea, R., Mountz, M., and Mountz, M. (2008). Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29:9–20.

Yu, J. and LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, PP(99):1–15.

Zahrádka, D., Kubišta, D., and Kulich, M. (2023). Solving robust execution of multi-agent pathfinding plans as a scheduling problem. *Planning and Robotics, ICAPS'23 Workshop*.