A Fast Rescheduling Algorithm for Real-Time Multi-Robot Coordination [Extended Abstract]

Adittyo Paul*, **Ying Feng***, **Jiaoyang Li**

Carnegie Mellon University
{adittyop, yingfeng}@andrew.cmu.edu, jiaoyangli@cmu.edu

Introduction

Multi-Agent Path Finding (MAPF) is an optimization problem of finding collision-free paths for a team of given agents A on a given graph with the minimum sum of travel times. Each agent $i \in A$ has a start and a goal location and performs a move or a wait action at each discrete timestep. A path for an agent is a series of locations indicating the position where the agent should be at every timestep. Two agents collide when they are at the same location or swapping their locations at the same timestep. The applications of MAPF are vast, including intralogistics, train scheduling, aircraft towing, and video games. In practice, however, it is not always possible for agents to tightly follow the planned paths as delays may occur due to various issues, including external disturbance and hardware failures. Hönig et al. (2016) thus proposed a TPG execution framework to address this issue.

Definition 1 (TPG)**.** A Temporal Plan Graph (TPG) (Hönig et al. 2016) is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that represents the precedence relationships of a given MAPF solution. A vertex v_n^i corresponds to the n^{th} location in the path of agent i. An edge $(v_n^i, v_{n'}^{i'})$ indicates that agent i' can move to vertex $v_{n'}^{i'}$ only after agent i has visited vertex v_n^i. Edges are categorized into two types. *Type 1 edges* connect vertices of the same agent and are of format $(v_n^i, v_{n+1}^i), \forall n, i$. *Type 2 edges* connect vertices of different agents, ensuring that the order in which agents visit the same location does not change. We thus add a Type 2 edge $(v_n^i, v_{n'}^{i'})$ iff agent i visits a certain location before agent i'.

During execution, instead of asking the agents to follow the MAPF solution precisely in time, we ask them to *follow* the TPG, i.e., we move an agent to its next vertex v_{n+1}^i only if all vertices $v_{n'}^{i'} : (v_{n'}^{i'}, v_{n+1}^i) \in \mathcal{E}$ has been visited. This ensures the success of the execution, i.e., no deadlocks and no collisions (Hönig et al. 2016). However, it often causes unnecessary waiting (Berndt et al. 2020) since, after a delay happens, changing the ordering of the agents at some locations may lead to better solutions. Motivated by the scenario when it is desirable to keep the agents to their pre-planned paths but modify the precedence dependencies at specific locations on the fly, we develop an optimal A*-based algorithm, called *Switchable-Edge Search*, to re-optimize the TPG in real-time when delays happen.

Algorithm 1: Switchable-Edge Search (SES)

Input: Switchable TPG $\mathcal{G}$ and start vertices S
Output: Optimal non-switchable TPG

```
1  R ← Node(G, S);
2  R.g ← 0;
3  R.h ← COMPUTEHEURISTIC(R);
4  OPEN ← {R}
5  while OPEN is not empty do
6      N ← OPEN.pop()
7      Δ_g ← 0; V ← N.V;
8      while true do
9          if ∃v^i_{c^i} ∈ V with switchable edge (v^i_{c^i}, v^j_n) ∈ N.G
             then
10             F ← Node(N.G, V); B ← Node(N.G, V);
11             Set (v^i_{c^i}, v^j_n) non-switchable in F.G and B.G;
12             Reverse the direction of (v^i_{c^i}, v^j_n) in B.G;
13             for N' ∈ {F, B} do
14                 if N'.G^- has cycles then continue;
15                 N'.g, ← N.g + Δ_g;
16                 N'.h ← COMPUTEHEURISTIC(N');
17                 OPEN ← OPEN ∪ {N'};
18             break;
19         V' ← STEPAGENTS(N.G, V);
20         if V' = {v^i_{-1} : i ∈ A} then return N.G;
21         if V' = V then break;
22         Δ_g ← Δ_g + |{v^i_{c^i} ∈ V : v^i_{c^i} ≠ v^i_{-1}}|; V ← V';
23 throw exception "No solution found";
```

Switchable-Edge Search

Reversing the directions of Type 2 edges of a TPG allows us to change the ordering for the agents to visit the same location. Therefore, when a delay occurs, we search for the set of modifications to the TPG that results in the optimal ordering given the delay. Inspired by previous work (Berndt et al. 2020), we introduce *switchable TPG*.

Definition 2 (Switchable TPG)**.** A Switchable TPG is a TPG such that a Type 2 edge $(v_n^i, v_{n'}^{i'})$ is marked switchable

*These authors contributed equally.

iff neither agent i nor agent i' has visited vertices v_n^i or $v_{n'}^{i'}$. Such edges are considered valid to reverse, which implies swapping the visit order of agent i and i' to the location.

Given a switchable TPG, we aim to determine the direction of each switchable edge by *Switchable-Edge Search (SES)*; see Algorithm 1. *SES* performs an A* search and outputs an optimal TPG that minimizes the sum of the travel times of all agents (assuming no delays occur in the future).

Node Each node N in *SES* contains a switchable TPG $N.\mathcal{G}$ and a set of the current (TPG) vertices of all agents $N.V = \{v_{c^i}^i : i \in A\}$. We use $N.\mathcal{G}^-$ to denote the subgraph of $N.\mathcal{G}$ that ignores all switchable edges. The root node R contains the initial switchable TPG formed from the MAPF solution with all agents at their start vertices, i.e., the vertices that agents are occupying when *SES* is called (Line 1). A node is a goal node iff every agent i is at its last vertex v_{-1}^i. The g-value of node N is the sum of timesteps that has moved all agents from their start vertices to $N.V$, and its admissible h-value is the minimum sum of timesteps required to move all agents from $N.V$ to their target vertices following $N.\mathcal{G}^-$.

Node Expansion We develop a simulator STEPAGENTS (Line 19) that moves all agents forward by one timestep following $N.\mathcal{G}$ and returns the new locations of the agents. When expanding a node N, we repeatedly run STEPAGENTS (Lines 8 to 22) until (1) all agents reach their last vertices, in which case we find an optimal TPG (Line 20), (2) no agents make progress, in which case we prune node N (Line 21), or (3) we encounter an unspecified precedence dependency (Lines 9 to 18), i.e., the current vertex of an agent is incident to a switchable edge, in which case we consider both possible directions of the edge by spawning two copies of the TPG: one fixes the edge as non-switchable, and the other reverses the direction of the edge and then fixes it as non-switchable. The two TPG copies, along with the current locations of the agents, are interpreted as two new nodes and added to the open list if no cycles exist.

Cycle Detection Once we fix the direction of a switchable edge e, it can lead to a deadlock, i.e., form a directed cycle in the TPG with other non-switchable edges. Although deadlocks can be detected when we compute heuristics since we simulate future movements and will observe that no agent moves yet some of them have not reached their last vertices. However, this process is time-consuming. We thus add a deadlock-detection mechanism to efficiently identify the deadlock TPG before computing the heuristic. Since the directed cycle, if it exists, must contain edge e, we run a depth-first search from the source vertex of e to detect it.

Heuristic Calculation A naïve way of computing the h-value of node N is to repeatedly run simulator STEPAGENTS to move all agents from $N.V$ to their last vertices following $N.\mathcal{G}^-$. Since $N.\mathcal{G}^-$ contains only non-switchable edges, we can replace this time-consuming simulation-based method with a search-based method. Given that $N.\mathcal{G}^-$ is acyclic (because, otherwise, it leads to a deadlock, and the h-value is set to infinity), it is easy to prove that the minimum timestep required for agent i to reach its target vertex is the longest path from the current vertex of any agent to the last vertex of agent i. We can find such longest paths for all agents in polynomial time by topological sort. Therefore, the simulation-based method and the search-based method always find the same admissible h-value.

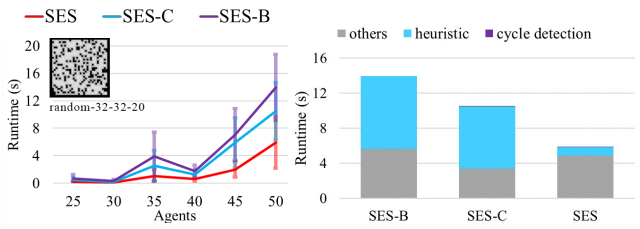

Figure 1: Left figure shows average runtimes (indicated by lines) with standard error (indicated by vertical bars) over 25 trials on a 32 × 32 random map with a time limit of 90 seconds. Right figure shows runtime breakdown for 50 agents.

Experiments

We implement in C++ three versions of our proposed algorithms: *SES-B(asic)*, which uses the simulation-based heuristic without cycle detection, *SES-C(ycleDetection)*, which uses the simulation-based heuristic with cycle detection, and *SES*, which uses the search-based heuristic with cycle detection. We assume that an agent can successfully move to a new location with a probability of 97% and, otherwise, gets delayed for 20 timesteps at its current location. Figure 1 plots the runtimes of the algorithms after the first delay happens, which are measured on a General Purpose Linux Computer running Ubuntu 18.04 LTS with 4 Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz processors. As shown, cycle detection has negligible runtime overhead. The runtime of the heuristic computation can be significantly sped up with cycle detection and the search-based method. Thus, *SES* runs the fastest and solves MAPF instances with 50 agents with an average runtime of only 5.92 seconds. Regarding solution quality, the sum of travel times of the agents using SES, for example, for 50 agents, is 14% better on average than that of the original TPG.

Acknowledgments

This research is supported by the CMU Manufacturing Futures Institute, made possible by the Richard King Mellon Foundation.

References

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. In *ICAPS Workshop on Distributed and Multi-Agent Planning*, 1–9.

Hönig, W.; Kumar, T. K. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *International Conference on Automated Planning and Scheduling*, 477–485.