Multi-Agent Pathfinding with Predefined Paths: To Wait, or Not to Wait, That Is the Question [Extended Abstract]

Jiří Švancara[1], **Etienne Tignon**[2], **Roman Barták**[1], **Torsten Schaub**[2,3], **Philipp Wanko**[2,3], **Roland Kaminski**[2,3]

[1] Charles University, Prague, Czech Republic
[2] University of Potsdam, Potsdam, Germany
[3] Potassco Solutions, Potsdam, Germany

svancara@ktiml.mff.cuni.cz, tignon@uni-potsdam.de, bartak@ktiml.mff.cuni.cz, torsten@cs.uni-potsdam.de, wanko@uni-potsdam.de, kaminski@cs.uni-potsdam.de, tignon@uni-potsdam.de

Abstract

Multi-agent pathfinding is the task of navigating a set of agents in a shared environment without collisions. Finding an optimal plan is a computationally hard problem, therefore, one may want to sacrifice optimality for faster computation time. In this paper, we present our preliminary work on finding a valid solution using only a predefined path for each agent with the possibility of adding wait actions. This restriction makes some instances unsolvable, however, we show instances where this approach is guaranteed to find a solution.

Introduction

Multi-agent pathfinding (MAPF) is the problem of navigating a fixed set of mobile agents in a shared environment (map) from their initial locations to target locations without any collisions among the agents (Silver 2005). This problem has numerous practical applications in robotics, logistics, digital entertainment, and automatic warehousing.

An *instance of MAPF* is a graph G and a set of agents A. *Agent* $a_i \in A$ is represented by a start-goal pair $a_i = (s_i, g_i)$. A *solution to MAPF* is a sequence of vertices that navigate agent a_i from s_i to g_i without any collisions with the other agents. The time is discretized and at each timestep, the agents either move into a neighboring vertex or they wait in their current vertex (Stern et al. 2019).

As opposed to the classical setting of MAPF, we set an extra constraint on the movement. Given a predefined (s_i, g_i) path $\mathcal{P}^i$ we allow each agent to either move forward on the predefined path or wait. As a consequence, each agent can enter each vertex on $\mathcal{P}$ only once and wait there or move out. Adding the extra restriction can make an instance unsolvable. A simple example can be seen in Figure 1. A recent study showed that the problem of deciding an existence of any solution with this extra constraint is NP-Hard (Abrahamsen et al. 2023).

The Wait-Graph Method

We consider all agents and the interaction of their starting and goal locations. Agent a_i waits for agent a_j if:

- a_j's start s_j is on the path $\mathcal{P}^i$ of a_i, or

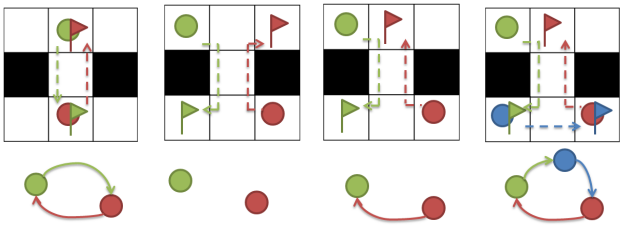

Figure 1: Example instances that are solvable in the classical MAPF. With the extra constraint (a) is unsolvable, (b) and (c) are solvable, and (d) is unsolvable, however, omitting any single agent from (d) makes it solvable.

- a_i's goal g_i is on the path $\mathcal{P}^j$ of a_j.

We create a *wait-graph* $W = (V, E)$, where V are agents and "a_i waits for a_j" creates a directed edge (a_i, a_j). Figure 1 shows examples of the wait-graph.

Proposition 1. *If there are no cycles in the wait-graph W (i.e. W is a directed acyclic graph (DAG)) the instance can be solved by navigating one agent after each other on their $\mathcal{P}$ while other agents wait. The order of the agents is given by the topological ordering of W.*

Proof. By induction, we take an agent a_i that is not waiting for anyone (such an agent must exist since W is DAG) and run it to its goal. Since a_i is not waiting for anyone, there is no other agent on its path and no other agent needs to go through a_i's goal. □

If there is no cycle in the wait-graph, we are guaranteed there is a solution. The reverse implication does not hold. There are instances where a cycle in the wait-graph can be resolved. In fact, we identified instances where the cycle can be resolved by creating another cycle which in turn can be resolved. As the decision problem is NP-Hard it is not possible to resolve all cycles in all cases in polynomial time.

The Extended Wait-Graph Method

We extended the concept of the wait-graph to order not only the agents but every vertex occupation along their path.

Given an instance (G, A) and a path $\mathcal{P}^i$ for every agent, we define the *vertex occupations set* $\mathcal{V}$. For every vertex $v \in \mathcal{P}^i$, $\mathcal{V}$ contains the tuple (a_i, v) which represents the presence of agent a_i on the vertex v. We can now express every situation when a presence of an agent on a vertex must happen before (or wait for) another (i.e., "(a_i, v) waits for (a_j, v')"):

1. "(a_i, v') waits for (a_i, v)" if a vertex v is on the path of a_i, and v' is the next vertex on the same path.
2. "(a_i, v) waits for (a_j, v)" or "(a_j, v) waits for (a_i, v)" if a vertex v is on the paths of both agents a_i and a_j.
3. "(a_i, s_j) waits for (a_j, s_j)" if a_j's start s_j is on the path of a_i.
4. "(a_i, g_i) waits for (a_j, g_i)" if a_i's goal g_i is on the path of a_j.
5. "(a_i, v) waits for (a_j, v)" implies "(a_i, v') waits for (a_j, v')" if a vertex v is on the paths of both agents a_i and a_j and, for both agents, the next vertex on their paths is v'.
6. "(a_i, v) waits for (a_j, v)" implies "(a_i, v') waits for (a_j, v')" if a vertex v is on the paths of both agents a_i and a_j and the next vertex v' on the path of a_i is also the previous vertex on the path of a_j.

Situation 1 maintains the ordering of vertices on each path. Situations 2 and 6 address relationships between paths that could lead to conflicts (vertex and edge conflicts respectively). Situations 3 and 4 address the special cases when an agent starts or finishes its path on the path of another agent. Situation 5 expresses that one agent cannot pass another agent when they follow each other.

We create an *extended wait-graph* $EW = (\mathcal{V}, E)$, where $\mathcal{V}$ is the vertex occupations set and E consists of directed edges $((a_i, v), (a_j, v'))$ meaning "(a_i, v) waits for (a_j, v')".

Proposition 2. *An instance (G, A) can be solved iff it has an extended wait-graph EW such that EW is a directed acyclic graph (DAG).*

We omit the proof of Proposition 2 due to space limitation. We claim that once an acyclic extended wait-graph EW is obtained, we are able to define a solution using the property "(a_i, v) waits for (a_j, v')" implies that the presence of a_j in vertex v' happens before the presence of a_i in vertex v. Furthermore, this solution is conflict-free due to the constraints set by situations 2 and 6. Getting an exact plan using scheduling has already been addressed in the literature. For example, ASP with difference constraints can generate a plan by scheduling the visit of each agent in compliance with the waiting restrictions (Abels et al. 2019).

Related Work

We identified related works that are similar to our setting of the problem and the approach to solving it. We describe the differences between these works and our work.

Wait-for graph as detection of deadlocks in concurrent systems (Silberschatz, Galvin, and Gagne 2018). In this concept, a similar graph to our wait-graph is built to represent agents waiting for resources used by other agents. If a cycle is detected, it means that there is a deadlock in the system. The difference to our wait-graph is that a cycle does not necessarily mean a deadlock, or rather no solution in our case.

MAPF as a scheduling problem (Barták, Svancara, and Vlk 2018). In this work, MAPF is modeled as a scheduling problem, where a layered graph is used (not the same layered graph as a time-expanded graph used in reduction-based MAPF solvers). If only a single layer is used, the agents are not allowed to return to a vertex and are allowed only to either move forward or wait. This is similar to our setting, however, in (Barták, Svancara, and Vlk 2018) the task is still to find a path for the agents, while we are asking if there is a solution given a path for each agent.

MAPF-POST (Hönig et al. 2016). In this work, the task is, for a given non-conflicting plan, to find an exact schedule of arrival to each vertex using STN (simple temporal network). The work deals with real robots, so the original non-conflicting plan is not assuming the motion constraints of the robot, hence an exact schedule in needed. While they start with a plan, we want to decide if there is one. An STN can decide this as well, however, STN needs ordering of visits of the agents in the vertices, this is unknown in advance.

Acknowledgments

Research is supported by Czech-American scientific cooperation project LTAUSA19072, Charles University project UNCE/SCI/004, and DFG grant SCHA 550/15, Germany.

References

Abels, D.; Jordi, J.; Ostrowski, M.; Schaub, T.; Toletti, A.; and Wanko, P. 2019. Train Scheduling with Hybrid ASP. In *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019*, 3–17.

Abrahamsen, M.; Geft, T.; Halperin, D.; and Ugav, B. 2023. Coordination of Multiple Robots along Given Paths with Bounded Junction Complexity. *CoRR*, abs/2303.00745.

Barták, R.; Svancara, J.; and Vlk, M. 2018. A Scheduling-Based Approach to Multi-Agent Path Finding with Weighted and Capacitated Arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018*, 748–756.

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 *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*, 477–485.

Silberschatz, A.; Galvin, P. B.; and Gagne, G. 2018. *Operating System Concepts, 10th Edition*. ISBN 978-1-118-06333-0.

Silver, D. 2005. Cooperative Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117–122.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019*, 151–159.