The Thirteenth International Symposium
on Combinatorial Search (SoCS 2020)

On Modelling Multi-Agent Path Finding as a Classical Planning Problem

Jindřich Vodrážka, Roman Barták, Jiří Švancara
Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, Praha, Czech Republic
vodrazka@ktiml.mff.cuni.cz, bartak@ktiml.mff.cuni.cz, Jiri.Svancara@mff.cuni.cz

Abstract

The paper shows how to encode Multi-Agent Path Finding problem as a sequential planning problem, specifically, how to encode various collision constraints and parallel actions.

Introduction

Mutli-Agent Path Finding (MAPF) deals with finding plans for a set of agents moving from their start locations to their goal locations in a shared environment represented by a graph. At every timestep, each agent can either move to a neighboring node or stay in the current node. Agents should not collide. Using a recently proposed terminology (Stern et al. 2019), we distinguish five types of collisions (Figure 1). We define two commonly used settings of movement:

Pebble motion, where no two agents can be present in a single node at any time (conflicts (a) and (b)) and an agent can move to a neighboring node only if the node is currently empty (conflicts (c), (d), and (e)).

Parallel motion, where no two agents can be present in a single node at any time or use the same edge at the same time (conflicts (a), (b), and (e)). Moving to a neighboring node is allowed provided that it is empty by the time the agent arrives. The following and cycle conflicts are allowed.

In this paper we shown how to represent the two specified settings of MAPF as a classical planning problem. Note that actions in the MAPF plans are parallel (actions at the same time step), while the classical plan is a sequence of actions.

Pebble Motion

To model the pebble motion variant, only a `move` action is needed. An agent may `move` if it is `at` a node and the target node is `free`. The effect is that the location of the agent and node occupation is updated accordingly. Only one agent at a time is moving while others are waiting. It is easy to see that this indeed models the pebble motion.

It is possible to parallelize the movement of agents to decrease the length of the plan. It is important to group (parallelize) only the move actions that do not cause a conflict. Note that the cycle conflict may never arise since it needs agents to move on a fully occupied cycle, however, the `move` action needs at least one `free` node.

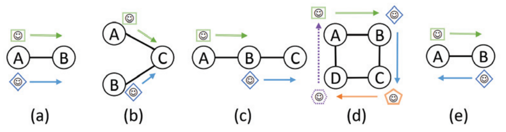

Figure 1: Types of agents' conflicts. From left to right: an edge conflict, a node conflict, a following conflict, a cycle conflict, and a swapping conflict (Stern et al. 2019).

Modeling Cycles

As mentioned above, it is not possible to model movement on cycles with a simple `move` action. We present two approaches to model such movement.

Sequential model Instead of doing a complete move to the next node, the agent opens the move by moving to the edge (disallowing any other agent to use that edge). The agent completes the move to its destination node when the node is empty, which may happen immediately after or later. The danger here is that while an agent stays on the edge, other agents may use the end nodes of that edge, which may give an invalid MAPF plan (Figure 2). Thus a freeze mechanism to disallow usage of these nodes by other agents is used.

For the opening action, we distinguish two situations, the next node is either `free` or some agent is `at` that node.

If the destination node is `free`, the agent can `freeze` it, making it `frozen` and not allowing any other agent to use

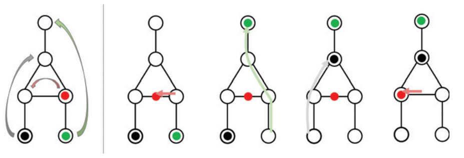

Figure 2: Invalid MAPF plan: red agent stays at the edge while other agents go through the end nodes of that edge.

it. At a later time, the agent can finish the move action by `finishMove` changing its location accordingly.

If, on the other hand, the destination node is occupied, we switch to a `lock` mode, where the agent pushes the other agent out of the destination node (action `require`). When the blocking agent moves away, it makes the node `frozen` for the agent that required the node. ID of agent, who made the request, is stored in the predicate `require` as only one agent is allowed to require a certain node at a time.

The agent that moves out of the required node may also need to move to a node that is occupied. This is done by action `passRequire` that moves the blocking agent to an edge while requiring the destination node from another agent (we pass the request). This action also freezes the origin node for the previous agent so when we leave the `lock` mode, that agent can finish its move.

The `lock` mode is abandoned, when the last agent moves to an edge and its destination node is free (`confirm`). This agent can be the head of a train or it could be the last agent in a cycle. The agent that started the `lock` mode via action `require` actually left its origin node and made that node `free`. This allows the last agent to use it and close the loop. At this time all participating agents are sitting at edges and their destination nodes are frozen so no other agent can use them (see Figure 3). In this situation, other agents can start moving, other loops or trains may be formed or agents close their moves from an edge to the next node.

The outline of the proof that this models a valid MAPF solution is as follows. Each move is represented by a pair of opening (`freeze`, `require`, `passRequire`, or `confirm`) and closing (`finishMove`) actions. Actions happening during the `lock` mode will be grouped together and will be performed in one parallel timestep. All of the `finishMove` actions can be moved right after their respective opening actions. Thus we have correctly paired actions that correspond to one move action and the plan can be parallelized in the same way as for the pebble motion.

Layered model Planners may be forced to do actions in a given order by using a layered structure modeling parallel steps. Again, each move is split into opening and closing actions (`startMove` and `finishMove`), but the order of the agents is forced by predicate `token` that marks the active agent and the predicate `next` that marks the next agent to take an action. As all agents have to perform an action, we need to explicitly model waiting actions (`startWait` and `finishWait`). Since all agents perform exactly one pair of actions at each layer (parallel timestep), the parallelization is straightforward. Note that the freeze mechanism is not required (the example in Figure 2 can not occur).

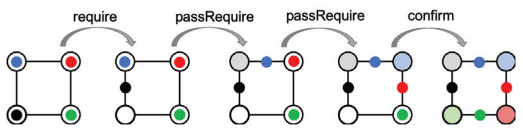

Figure 3: The sequential model of a cycle: frozen nodes are depicted in shadow (the color indicates for which agent).

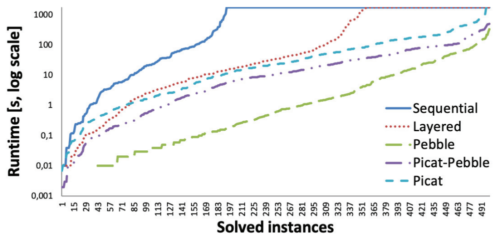

Figure 4: Number of instances solved in a given time limit.

Empirical Evaluation

To compare the proposed models, we used a MAPF benchmark set (Stern et al. 2019). We chose five different maps with dimensions ranging from 8×8 to 32×32. The number of agents increases from 1 to 20. For each of the settings, 5 instances were used, which gives 500 instances in total. All tested models are available on-line (https://github.com/svancaj/MAPF-via-PDDL).

PAR10 score was used to compare the efficiency of the models. This score reflects the runtime and adds an extra penalty for each timeouted instance – the lower the score, the better the model performed. The best model with PAR10 14 and no timeouted instances is the *pebble motion* model, followed by *Layered* with 144 and *Sequential* with 308 timeouted instances and PAR10 5767 and 12226 respectively. A more detailed comparison can be seen in Figure 4, where an ad-hoc planner in Picat was also used. The results indicate that the more complex models that allow cycle conflicts are indeed more challenging to compute. Also, the layered model with a token-passing mechanism is more successful than a carefully crafted sequential model.

Conclusion

In the paper, we proposed sequential planning models to describe MAPF problems. The straightforward pebble-motion model is surprisingly efficient even in comparison with an ad-hoc planner, but it cannot describe plans with cycles. We extended the model to cover cycles, but the performance degraded significantly. By forcing the planner to plan actions for all agents in one layer, we improved the performance.

Note that none of the presented models guarantees an optimal solution. However, the *Layered* model solved 232 instances (out of 356) optimally and those, that were not solved optimally, had a solution on average 6.2% worse than optimum.

Acknowledgment Research is supported by the project P103-19-02183S of the Czech Science Foundation, and by the project 90119 of the Charles University Grant Agency.

References

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Barták, R. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *the International Symposium on Combinatorial Search (SoCS)*.