Multi-Agent Pathfinding

Till Zemann

2022-10-20 15:57:00 +0200

- TOC {:toc}

The Multi-Agent Pathfinding Problem

Multi-Agent Pathfinding (**MAPF**) is the problem of planning paths for multiple agents without colliding.

Applications

It seems to me that MAPF performs better than Multi-agent Reinforcement Learning in restrictive, discrete situations. State-of-the-art MAPF can usually solve up to a low four digit number of agents. The classical MAPF application is logistic warehouses, so we might just be working for free for Amazon. :)

Assumptions

Common assumptions are:

- the environment is **discrete**
- an agent executes one action per timestep
- an agent occupies one vertex/node per timestep

Input

The input to a MAPF problem is a triple $< G, s, t >$ consisting of:

- an undirected graph $G = (V, E)$
- a mapping s to source vertices with $s : [1, \ldots, k] \to V$
- a mapping t to target vertices with $t : [1, \ldots, k] \to V$

Solution

The solution of a MAPF problem is a set π of single-agent plans without conflicts: $\pi = \{\pi_1, \pi_2, \ldots, \pi_k\}$ where π_i denotes the single-agent plan for agent i. A single-agent plan is an action mapping π (careful: notation overload!) that results in the agent being their target state. We can write this constraint as $\pi_i[|\pi|] = t(i)$.

Note, that π does **not** include the starting position $s(i)$. Instead, the first entry in π is the action that performed on the first timestep.

We can also ask, where an agent i is positioned after timestep x (equivalent to asking which node an agent occupies). We would write this as $\pi_i[x]$.

Conflicts

To properly define a MAPF problem, you should cover which of the following conflicts are allowed and which can not appear in a solution π.

Conflict types: a) Vertex conflict b) Swapping conflict c) Following conflict d) Circle conflict

Objectives and Constraints

The two most used objective functions are the **Makespan** and **Sum of costs**.

Makespan The **Makespan** of a MAPF solution is definded as the number of timesteps it takes until all agents reach their goals: $J(\pi) = \max_{1 \leq i \leq k} |\pi_i|$

Sum of costs The **sum of costs** objective function takes the length of all individual agent plans into consideration by summing over all action plan lengths: $J(\pi) = \sum_{1 \leq i \leq k} |\pi_i|$

There is also the not so common **sum of fuel** objective function that counts all non-waiting moves.

An **optimal solution** to our problem is one that **minimizes the chosen objective function** $J(\pi)$ (and satisfies all other given constraints).

Constraints Typical hard constraints that are additionally added are **k-robustness** (an agent can only move to a vertex that hasn't been visited by any agent for k timesteps) and **formation rules**. The **k-robustness** adresses the possiblity of delays that could result in agents colliding at execution. The goal is to be within a probabilistic margin for conflicts or have a policy that can deal with delays at execution time to prevent conflicts. **Formation rules** enforce a specific formation of agents, e.g. to allow communication chains via neighboring agents.

Target behaviors

If an agent that already arrived at its target position doesn't plan on moving away from the target while waiting for other agents to reach their goals it is common to not count the waiting moves towards the sum of cost.

There are two possibilities of handling agents that reach their target. The agent can either **stay at the target** or **disappear at the target**. The stay at target

behavior is more commonly used because it doesn't assume that the environment has a special mechanism for handling the transportation of the agent (e.g. to a fixed starting position) upon arriving at the target.

Special MAPF problems

Weighted Actions MAPF with weighted actions addresses problems, where the assumption of one action per timestep is not useful. The length of an action can be encoded as the weights in an weighted graph, which can be represented as 2^k-grids or in a generalized form as euclidian (2d) space. Note, that diagonal moves in euclidian space are possible and have an execution (time-) cost of $\sqrt{2}$.

Motion-planning This takes the MAPF problem to a **state-based** problem, where the state encodes information like position, orientation and velocity. AN edge between two state configurations can be seen as planning movement (or kinematic motion). If kinematic constraints are added to the MAPF problem, the graph becomes **directed**.

A (not comprehensive) list of other extensions of MAPF includes

- MAPF with large agents
- MAPF with kinematic constraints
- Anonymous MAPF
- Colored MAPF
- Online-MAPF.

References

1. Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., ... & Boyarski, E. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In AAAI/ACM Conference on AI, Mobility, and Autonomous Systems (pp. 75-82). (arXiv)
2. Kaduri, Omri: From A* to MARL (5 part blogpost series)