Multi-Agent Path Finding with Blocking Containers

Bachelor Arbeit

Ben Bausch

June 21, 2019

1 Introduction

2 Background

2.1 Classical Multi-Agent Path-Finding

A classical multi-agent path-finding (MAPF) problem can be defined as a tuple:

$$\Gamma =< G, A, s, t >$$

Where:

- $G =< V, E >$ is a graph, representing the environment
  - V is a set of vertexes $V = \{v_1, v_2, ..., v_n\}$, representing the set of possible positions
  - E is a set of edges $E = \{e_1, e_2, ..., e_m\}$, describing all possible position transitions
- A is the set of k agents $A = \{a_1, a_2, ..., a_k\}$
- $s : A \to V$ is an injective function, that maps each Agent $a \in A$ to a start position $v_s \in V$
- $t : A \to V$ is an injective function, that maps each Agent $a \in A$ to a goal position $v_* \in V$

In [1], an action is a function $a : V \to V$, such that $a(v) = v$. This means, that taking action a in vertex v, results in a position transition of the agent to v'.
The following definition of an action will serve as a more explanatory and planning-like definition, but will not change the meaning of the previous action definition.
Actions consist of a precondition χ, which has to evaluated to true for the action to be applicable, and an effect e, which describes how the world will change performing the action. Actions are a tuple: $a =< \chi, e >$.
The precondition and the effect are logical formulae, possibly containing the predicates:

- $at(a, v)$, true if agent a is at vertex v
- $empty(v)$, true if no agent a is at vertex v
- $edge(v_i, v_j)$, true if there is an edge between the vertices v_i and v_j

Usually, Agents can perform 2 different actions, they can either move or stay:

- the agent moves from vertex v_i to an adjacent vertex v_j:
  $move(a, v_i, v_j) =< (at(a, v_i) \wedge empty(v_j) \wedge edge(v_i, v_j)), (\neg at(a, v_i) \wedge empty(v_i)) >$
- the agent stays at the current position for another time step:
  $wait(a, v_i) =< (at(a, v_i)), (at(a, v_i)) >$

We also have to define a cost-function c, that assigns a cost to each action. Typically, a unit-cost-function, that assigns each action the same cost of 1, is used.
As in [1], we define a sequence of actions of one agent a_i as followed $\pi(a_i) = (a_1, a_2, ..., a_n)$ and let $\pi_i[x]$ be the vertex v after executing the first x actions of the sequence. A single-agent-plan is a sequence of actions $\pi(a_i)$, so that $\pi_i[0] = s(a_i)$ and $\pi_i[|\pi(a_i)|] = t(a_i)$. A solution to the MAPF problem is a set of single-agent-plans for all the k agents, so that no pair of single-agent-plans results in a conflict.
Many different type of conflicts can be defined, but the most spread are the following:

- **vertex conflict**: two agents occupy the same vertex an the same time, $\pi_i[x] = \pi_j[x]$
- **edge conflict**: two agents move from the same vertex v to the same vertex v', $\pi_i[x] = \pi_j[x]$ and $\pi_i[x+1] = \pi_j[x+1]$
- **Following conflict**: an agent move to a vertex, just left by an other agent, $\pi_i[x+1] = \pi_j[x]$
- **Cycle Conflict**: every agent moves to an agent previously occupied by an other agent, $\pi_i[x+1] = \pi_{i+1}[x]$ and $\pi_{i+1}[x+1] = \pi_{i+2}[x]$ and ... and $\pi_{k-1}[x+1] = \pi_k[x]$ and $\pi_k[x+1] = \pi_i[x]$
- **Swapping Conflict**: two agent swap vertices over the same edge, $\pi_i[x+1] = \pi_j[x]$ and $\pi_i[x] = \pi_j[x+1]$

2.2 MAPF with blocking containers

To describe a MAPF with blocking containers (MAPF-BC) problem, we have to change the previously introduced formalization. It can be described as follows.
A MAPF-BC problem:

$$\Gamma =< G, A, C, s, s', t' >$$

Where:

- G, A and s are defined like for the classical MAPF problem
- C is the set of m containers, that need to be transported to a location, $C = \{c_1, c_2, ..., c_m\}$
- $s' : C \to V$ is an injective function, that maps each container $a \in A$ to a start position $v_s \in V$
- $t' : C \to V$ is an injective function, that maps each container $a \in A$ to a goal position $v_* \in V$

Since we introduce, a new type of agent, which change the physics of the environment, we introduce new set of possible predicates for the preconditions and effects of the actions:

- $at(a, v)$, true if agent a is at vertex v
- $at(c, v)$, true if container c is at vertex v
- $empty(v)$, true if no agent/container is at vertex v
- $agent(v)$, true if there exists an agent that occupies vertex v
- $container(v)$, true if there exists a container that occupies vertex v
- $edge(v_i, v_j)$, true if there is an edge between the vertices v_i and v_j

To fit the new dynamics of the environment, we introduce the following actions:

- $move_a(a, v_i, v_j)$ moves agent a from vertex v_i to the adjacent vertex v_j
- $move_c(a, c, v_i, V_j)$ moves the container from vertex v_i to adjacent vertex v_j
- $wait_a(a, v_i)$ agents stay in its current vertex
- $wait_c(c, v_i)$ container stays at current position

The preconditions of the actions are the following:

- $pre(move_a(a, v_i, v_j)) = at(a, vi) \wedge (empty(v) \vee container(v)) \wedge edge(v_i, v_j)$
- $pre(move_c(a, c, v_i, v_j)) = at(a, v_i) \wedge at(c, v_i) \wedge empty(v_j) \wedge edge(v_i, v_j)$
- $pre(wait_a(a, v_i)) = at(a, v_i)$
- $pre(wait_c(c, v_i)) = at(c, v_i)$

The effects of the actions are :

- $eff(move_a(a, v_i, v_j)) = at(a, v_j) \wedge \neg at(a, v_i) \wedge agent(v_j) \wedge \neg agent(v_i) \wedge empty(v_i)$
- $eff(move_c(a, c, v_i, v_j)) = at(a, v_j) \wedge \neg at(a, v_i) \wedge at(c, v_j) \wedge \neg at(c, v_i) \wedge agent(v) \wedge \neg agent(v_i) \wedge container(v_j) \wedge \neg container(v_i) \wedge empty(v_i)$
- $eff(wait_a(a, v_i)) = at(a, v_i)$
- $eff(wait_c(c, v_i)) = at(c, v_i)$

We will still assign a uni-cost of 1 to all the actions, but we will introduce some additional actions each container can choose to execute.

To formalize conflicts, $\pi_i[x]$ is not suited anymore since an action can change the position of two objects at once. To solve this we introduce the function $\gamma_{a,c}[t] = (v_a(t+1), v_c(t+1))$, which returns the position the 2 objects after executing an action at time step t. If only one object is affected, the function will return x for the position of the second object, for example if we execute $move_a(a, v_i, v_j)$ at time step t $\gamma_{a,x}[t] = (v_j, x)$.

In MAPF-BC, we will only be interested in the following conflicts:

- **vertex conflict**: two agents occupy the same vertex an the same time, $\gamma_{a,x}[t] = \gamma_{a',x}[t]$
- **edge conflict**: two agents move from the same vertex v to the same vertex v', $\gamma_{a',x}[t] = \gamma_{a',x}[t]$ and $\gamma_{a,x}[t+1] = \gamma_{a',x}[t+1]$
- **Swapping Conflict**: two agent swap vertices over the same edge, $\gamma_{a,x}[t+1] = \gamma_{a',x}[t]$ and $\gamma_{a,x}[t] = \gamma_{a',x}[t+1]$

The main difference between the classical MAPF and the MAPF-BC is the fact that one Meta-agent can control 2 agents.

2.2.1 Formalization from Agents Perspective

3 References

References

[1] Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks -Stern et al.