Multi-Agent Path Finding with Blocking Containers

Bachelor Arbeit

Ben Bausch

June 20, 2019

1 Introduction

2 Background

Fließtext oder eher Bulletpoint Liste

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 $v_* \in V$

alles in das tupel einfügen?

In the classical MAPF problem, 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'. Usually, Agents can perform 2 different actions, they can either move or stay:

- the agent moves to an adjacent vertex: $move(v_i) = v_j$ if $\exists e_{ij} \in E : e_{ij}$ is edge between v_i and v_j,
- the agent stays at the current position for another time step: $wait(v) = v$

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 $\pi_i[|\pi(a_i)|] = t(a_i)$. hier anstatt π, verwende ich $\pi(a_i)$. 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 problem (MAPF-BC), we have to change the previously introduced formalization. It can be described as follows:

2.2.1 Planning from Containers Perspective

This formalization only works if no blocking container has to be moved in order to make the task solvable.

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

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 $v_* \in V$

We also define a matrix T of form $A \times C$, that agent is coupled to one container. We extend the definition of $\pi_i[x]$, to return the vertex of the container and the assigned agent:

$$\pi_i[x] = (v_c, v_a)$$

To refer to the to containers vertex, we write $\pi_i[x]_c = v_c$. Analog we write $\pi_i[x]_a = v_a$, to refer to agents vertex, after executing the first x actions.
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. Each container can choose to either:

- move its agent to an adjacent vertex:
  $move_a(v_i) = v_j$ if $\exists e_{ij} \in E : e_{ij}$ is edge between v_i and v_j, the container stays at the same position.
- make its agent wait in its current vertex:
  $wait_a(v_i) = v_i$, the container stays at the same position
- move to an adjacent vertex, if its agent is at the same current vertex:
  $move_c(v_i) = v_j$ if ($\exists e_{ij} \in E : e_{ij}$ is edge between v_i and v_j) and (after executing the previous actions x-1, both are at the same vertex: $\pi_i[x-1] = (v_i, v_i)$). Moving the container, induces moving the agent at the same time: $\pi_i[x] = (v_j, v_j)$

It is important to notice, that an action has a precondition, defining when action can be taken:
To define this preconditions, we introduce the state of a vertex v: $state_t[v]$. The state describes by which type of object the vertex is occupied by at time step t. States of a vertex v at time step t:

- empty "e" if ($\nexists c_i \in C : \pi_{c_i}[t]_c = v$) $\wedge$ ($\nexists a_i \in A : \pi_{c_i}[t]_a = v$ with c_i being the container of a_i)
- container "c" if ($\exists c_i \in C : \pi_{c_i}[t]_c = v$) $\wedge$ ($\nexists a_i \in A : \pi_{c_i}[t]_a = v$ with c_i being the container of a_i)
- agent "a" if ($\nexists c_i \in C : \pi_{c_i}[t]_c = v$) $\wedge$ ($\exists a_i \in A : \pi_{c_i}[t]_a = v$ with c_i being the container of a_i)
- both "ac" if ($\exists c_i \in C : \pi_{c_i}[t]_c = v$) $\wedge$ ($\exists a_i \in A : \pi_{c_i}[t]_a = v$ with c_i being the container of a_i)

Preconditions:

- $move_a(v_i) = v_j$ can be taken if $state_t[v_j] = empty$ or $state_t[v_j] = container$
- $wait_a(v_i) = v_i$ can always be executed
- $move_c(v_i) = v_j$ can be taken if $state_t[v_j] = empty$

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

- **vertex conflict**: two agents occupy the same vertex an the same time, $\pi_i[x]_a = \pi_j[x]_a$
- **edge conflict**: two agents move from the same vertex v to the same vertex v', $\pi_i[x]_a = \pi_j[x]_a$ and $\pi_i[x+1]_a = \pi_j[x+1]_a$
- **Swapping Conflict**: two agent swap vertices over the same edge, $\pi_i[x+1]_a = \pi_j[x]_a$ and $\pi_i[x]_a = \pi_j[x+1]_a$

2.2.2 Formalization from Agents Perspective

3 References

References

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