Multi-Agent Path Finding with Blocking Containers

Bachelor Arbeit

Ben Bausch

July 1, 2019

1 Abstract

Over the last years, there has been an ever increasing interest in path finding for multi-agent systems, with applications ranging from automated warehouses to routing autonomous cars optimally through traffic. In the paper, I will formalize a warehouse problem with two different types of Agents, which can influence the behaviour of each other. This requires a new problem formalization MAPF and combines multiple challenges of classical MAPF. I will conduct experiments on the performance of current state-of-the-art algorithms in this new environment.

2 Introduction

3 Background

3.1 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
- $edge(v_i, v_j)$, 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:
  $move(a, v_i, v_j) =< (at(a, v_i) \wedge edge(v_i, v_j)), (\neg at(a, v_i) \wedge at(a, v_j)) >$
- 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 $\pi_i[x + 1] = \pi_j[x + 1]$ implies a vertex conflict after exec of action x
- **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 $\pi_i[x] = \pi_j[x + 1]$

3.2 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:

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

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 $v_* \in V$
- $p : \{A, C\} \to$ active, passive is a function, that determines which Agent set does the planning. At most one set can be active. An agent is active if $e \in EandEisactive$

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
- $edge(v_i, v_j)$, 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:

- $pre(move_a(a, v_i, v_j)) = at(a, vi) \land edge(v_i, v_j)$
- $pre(move_c(a, c, v_i, v_j)) = at(a, v_i) \land at(c, v_i) \land edge(v_i, v_j)$
- $pre(wait_a(a, v_i)) = at(a, v_i)$
- $pre(wait_c(c, v_i)) = at(c, v_i)$

The effects of the actions are :

- $eff(move_a(a, v_i, v_j)) = at(a, v_j) \land \neg at(a, v_i)$
- $eff(move_c(a, c, v_i, v_j)) = at(a, v_j) \land \neg at(a, v_i) \land at(c, v_j) \land \neg at(c, v_i)$
- $eff(wait_a(a, v_i)) = at(a, v_i)$
- $eff(wait_c(c, v_i)) = at(c, v_i)$

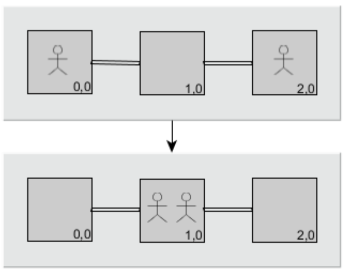
We will still assign a uni-cost of 1 to all the actions, meaning the action will take 1 time step to be executed. This can be changed to better suit real life settings, but has to be considered in the planning algorithms.
To formalize conflicts, we will use the function $\pi_i[t] = v$, the same as in classical MAPF, and the function $\gamma_i[t] = (t, v_t)$ returns a tuple, where the first position indicates the affected agent and its position after agent i executed the action at time step t in the plan of i. Look at figure add figure!, for some examples.
In MAPF-BC, we will only be interested in the following conflicts:

- **agent vertex conflict**: two agents occupy the same vertex an the same time,
  if $\exists a_1, a_2 \in A$ and $c, c' \in C$: $(\pi_{a_1}[t] = \pi_{a_2}[t])$ or $(\gamma_c[t] = (a_1, v_i) \land \gamma_{c'}[t] = (a_2, v_i))$
- **container vertex conflict**: two containers occupy the same vertex,
  if $\exists c_1, c_2 \in C \land \pi_{c_1}[t] = \pi_{c_2}[t]$

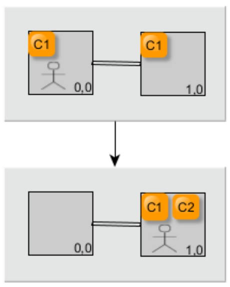
- **agent conflict**: two containers move the same agent to two different positions, if $\exists c_1, c_2 \in C \wedge \exists a \in A \wedge \gamma_{c_1}[t] = (a, v_j) \wedge \gamma_{c_2}[t] = (a, v_k) \wedge v_j \neq v_k$
- **Swapping Conflict**: two agent swap vertices over the same edge, if $\exists a_1, a_2 \in A$ and $c_1, c_2, c_3, c_4 \in C$: $(\pi_{a_1}[t] = \pi_{a_2}[t-1] \wedge \pi_{a_1}[t-1] = \pi_{a_2}[t])$ or $\gamma_{c_1}[t+1] = (a_1, v_i) \wedge \gamma_{c_2}[t] = (a_2, v_i) \wedge (\gamma_{c_3}[t] = (a_1, v_j) \wedge \gamma_{c_4}[t+1] = (a_2, v_j))$

It is important to note, that agent conflicts may only occurs if the set of containers is the active planning set. As in classical MAPF, we can try to optimize the makespan or the sum of cost:

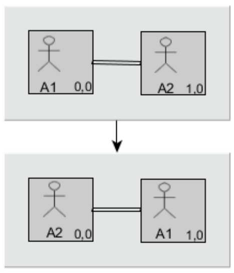
- **Makespan** is defined as the maximum length over all single-agent plans: $max_{1<i<k}(|\pi_i|)$ for all agents i in the active planning set.
- **Sum of costs** is the sum over all the actions for each active agent: $\sum_{n=1}^{k} |\pi_i|$ for all agents i in the active planning set.



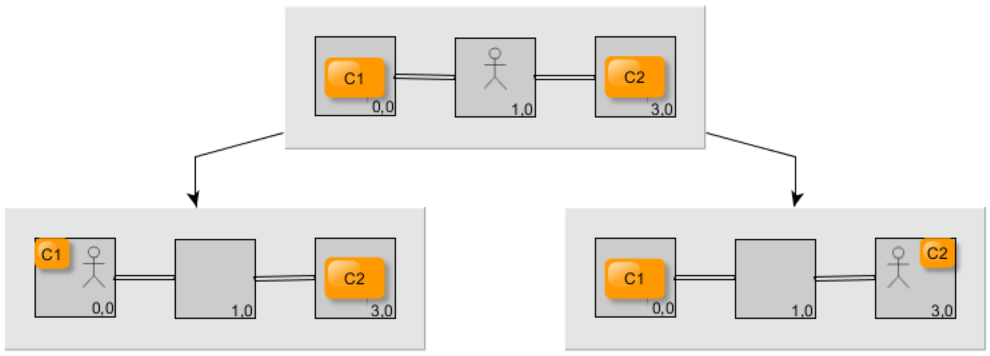
a)



b)



c)



d)

4 References

References

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