From Abstract Models to Executable Models for Multi-Agent Path Finding on Real Robots

Roman Barták
Charles University, Czech Republic

with contributions from Ivan Krasičenko, David Nohejl, Věra Škopková, and Jiří Švancara

Introduction

What is multi-agent path finding (MAPF)?

MAPF problem:
Find a collision-free plan (path) for each agent

Alternative names:
cooperative path finding (CPF), multi-robot path planning, pebble motion
Part I: Introduction to MAPF  
   – Problem formulation, variants and objectives

Part II. Solving MAPF  
   – Reduction-based solvers

Part III. From abstract to executable actions  
   – Translation vs. model modification

Part IV. Demo
Part I: Introduction to MAPF
   – *Problem formulation, variants and objectives*

Part II. Solving MAPF
   – *Reduction-based solvers*

Part III. From abstract to executable actions
   – *Translation vs. model modification*

Part IV. Demo

MAPF formulation

- a **graph** (directed or undirected)
- a set of **agents**, each agent is assigned to two locations (nodes) in the graph (start, destination)
Each agent can perform either **move** (to a neighboring node) or **wait** (in the same node) actions.

*Typical assumption:* all move and wait actions have identical durations (plans for agents are synchronized)

**Plan** is a sequence of actions for the agent leading from its start location to its destination.

The **length of a plan** (for an agent) is defined by the time when the agent reaches its destination and does not leave it anymore.

---

**MAPF task**

Find **plans** for all agents such that the plans **do not collide in time and space** (no two agents are at the same location at the same time).
Some trivial **conditions for plan existence:**

- no two agents are at the same start node
- no two agents share the same destination node (unless an agent disappears when reaching its destination)
- the number of agents is strictly smaller than the number of nodes

---

**No-swap constraint**

Agent at $v_i$ cannot perform move $v_j$ at the same time when agent at $v_j$ performs move $v_i$.

Agents may swap position:

<table>
<thead>
<tr>
<th>time</th>
<th>agent 1</th>
<th>agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>1</td>
<td>move $v_2$</td>
<td>move $v_1$</td>
</tr>
</tbody>
</table>

Swap is not allowed:

<table>
<thead>
<tr>
<th>time</th>
<th>agent 1</th>
<th>agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>1</td>
<td>move $v_2$</td>
<td>move $v_3$</td>
</tr>
<tr>
<td>2</td>
<td>move $v_4$</td>
<td>move $v_2$</td>
</tr>
<tr>
<td>3</td>
<td>move $v_2$</td>
<td>move $v_1$</td>
</tr>
</tbody>
</table>
Agent can approach a node that is currently occupied but will be free before arrival.

Agent at $v_i$ cannot perform \textbf{move $v_j$} if there is another agent at $v_j$.

Trains may be forbidden.

<table>
<thead>
<tr>
<th>time</th>
<th>agent 1</th>
<th>agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>1</td>
<td>wait $v_1$</td>
<td>move $v_3$</td>
</tr>
<tr>
<td>2</td>
<td>move $v_2$</td>
<td>wait $v_3$</td>
</tr>
<tr>
<td>3</td>
<td>move $v_4$</td>
<td>wait $v_3$</td>
</tr>
<tr>
<td>4</td>
<td>wait $v_4$</td>
<td>move $v_2$</td>
</tr>
<tr>
<td>5</td>
<td>wait $v_4$</td>
<td>move $v_1$</td>
</tr>
<tr>
<td>6</td>
<td>move $v_2$</td>
<td>wait $v_1$</td>
</tr>
</tbody>
</table>

Agents form a \textbf{train}.

Train collisions

If any agent is delayed then trains may cause collisions during execution.

To prevent such collisions we may introduce more space between agents.
**k-robustness**

An agent can visit a node, if that node has not been occupied in recent \( k \) steps.

1-robustness covers both no-swap and no-train constraints

---

**Objectives**

How to measure quality of plans?
Two typical criteria (to minimize):

- **Makespan**
  - distance between the start time of the first agent and the completion time of the last agent
  - maximum of lengths of plans (end times)

- **Sum of costs (SOC)**
  - sum of lengths of plans (end times)

<table>
<thead>
<tr>
<th>time</th>
<th>agent 1</th>
<th>agent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
</tr>
<tr>
<td>1</td>
<td>wait ( v_1 )</td>
<td>move ( v_3 )</td>
</tr>
<tr>
<td>2</td>
<td>move ( v_3 )</td>
<td>move ( v_4 )</td>
</tr>
<tr>
<td>3</td>
<td>move ( v_4 )</td>
<td>move ( v_6 )</td>
</tr>
<tr>
<td>4</td>
<td>move ( v_5 )</td>
<td>wait ( v_6 )</td>
</tr>
</tbody>
</table>

Makespan = 4
SOC = 7
Part I: Introduction to MAPF
   – Problem formulation, variants and objectives

Part II. Solving MAPF
   – Reduction-based solvers

Part III. From abstract to executable actions
   – Translation vs. model modification

Part IV. Demo

Search-based techniques
state-space search (A*)
   state = location of agents at nodes
   transition = performing one action for each agent

conflict-based search

Reduction-based techniques
translate the problem to another formalism (SAT/CSP/ASP ...)
Solving approaches

Search-based techniques
state-space search (A*)
state = location of agents at nodes
transition = performing one action for each agent
conflict-based search

Reduction-based techniques
translate the problem to another formalism (SAT/CSP/ASP ...)

Introduction to SAT

Express (model) the problem as a **SAT formula** in a conjunctive normal form (CNF)

Boolean variables (true/false values)
clause = a disjunction of literals (variables and negated variables)
formula = a conjunction of clauses
solution = an instantiation of variables such that the formula is satisfied

**Example:**

(X or Y) and (not X or not Y)
[exactly one of X and Y is true]
SAT model is expressed as a CNF formula

We can go beyond CNF and use **abstract expressions** that are translated to CNF.

<table>
<thead>
<tr>
<th>A =&gt; B</th>
<th>B or not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum(Bs) &gt;= 1</td>
<td>disj(Bs)</td>
</tr>
<tr>
<td>(at-least-one(Bs))</td>
<td></td>
</tr>
<tr>
<td>sum(Bs) = 1</td>
<td>at-most-one(B) and at-least-one(B)</td>
</tr>
</tbody>
</table>

We can even use **numerical variables** (and constraints).

---

**SAT encoding: core idea**

In MAPF, we do not know the lengths of plans (due to possible re-visits of nodes)!

We can encode plans of a known length using a **layered graph** (temporally extended graph).

Each layer corresponds to one time slice and indicates positions of agents at that time.
Using **layered graph** describing agent positions at each time step

\[ B_{tav} : \text{agent } a \text{ occupies vertex } v \text{ at time } t \]

**Constraints:**

- each agent occupies exactly one vertex at each time.
  \[ \sum_{v=1}^{m} B_{tav} = 1 \text{ for } t = 0, \ldots, m, \text{ and } a = 1, \ldots, k. \]
- no two agents occupy the same vertex at any time.
  \[ \sum_{a=1}^{k} B_{tav} \leq 1 \text{ for } t = 0, \ldots, m, \text{ and } v = 1, \ldots, n. \]
- if agent \( a \) occupies vertex \( v \) at time \( t \), then \( a \) occupies a neighboring vertex or stay at \( v \) at time \( t + 1 \).
  \[ B_{tav} = 1 \Rightarrow \sum_{u \in \text{neibs}(v)} (B_{t+1}au) \geq 1 \]

**Preprocessing:**

\( B_{tav} = 0 \) if agent \( a \) cannot reach vertex \( v \) at time \( t \) or \( a \) cannot reach the destination being at \( v \) at time \( t \)

---

**Picat code**

```picat
import data.

path(N, As) =>
  K = len(As),
  lower_upper_bounds(As, LB, UB),
  between(LB, UB, N),
  B = new_array(1..K, N),
  B[1, A, V] = 1,
  B[N+1, A, V] = 1,
  % initialize the first and last states
  foreach (A in 1..K)
    (V, FV) = As[A],
    B[1, A, V] = 1,
    B[N+1, A, FV] = 1,
  end,
  % Each agent occupies exactly one vertex
  foreach (V in 1..N)
    sum([B[T, A, V] : V in 1..N]) #= 1
  end,
  % No two agents occupy the same vertex
  foreach (T in 1..K, A in 1..K, V in 1..N)
    sum([B[T, A, V] : A in 1..K]) #< 1
  end,
  % Every transition is valid
  foreach (T in 1..K, A in 1..K, V in 1..N)
    neibs(V, Neibs),
    foreach (V in 1..N)
      sum([B[T+1, A, U] : U in Neibs]) #= 1
    end,
  end,
  solve(B),
  output_plan(B).
```

- Incremental generation of layers
- Setting the initial and destination locations
- Agent occupies one vertex at any time
- No conflict between agents
- Agent moves to a neighboring vertex

[K-robustness]
Part I: Introduction to MAPF  
– Problem formulation, variants and objectives

Part II. Solving MAPF  
– Reduction-based solvers

Part III. From abstract to executable actions  
– Translation vs. model modification

Part IV. Demo

Turning

6 classical actions needed to go from v1 to v7  
plus 4 turning actions during execution  
turning may take significant time (w.r.t. moving)
Abstract vs. executable actions

Abstract actions:
• move
• wait

Executable actions:
• move forward
• wait
• turn left/right + move
• turn back and move

Times:
\[ t_t \text{ – time to turn left/right} \]
\[ t_f \text{ – time to move forward} \]

<table>
<thead>
<tr>
<th></th>
<th>classic</th>
<th>classic+wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_f )</td>
<td></td>
<td>( t_f + 2*t_t )</td>
</tr>
<tr>
<td>( t_f + t_t/2 )</td>
<td>( t_f + 2*t_t )</td>
<td>( t_f + 2*t_t )</td>
</tr>
<tr>
<td>( t_f + t_t )</td>
<td>( t_f + 2*t_t )</td>
<td>( t_f + 2*t_t )</td>
</tr>
<tr>
<td>( t_f + 2*t_t )</td>
<td>( t_f + 2*t_t )</td>
<td>( t_f + 2*t_t )</td>
</tr>
</tbody>
</table>

Model with turning

It is possible to assume turn actions during path finding by splitting the nodes.

Classical model

Split model
Experiment setting

Some results

<table>
<thead>
<tr>
<th></th>
<th>Computed Makespan</th>
<th>Failed Runs</th>
<th>Number of Collisions</th>
<th>Total Time [s]</th>
<th>Max Δ time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>classic</td>
<td>17</td>
<td>5</td>
<td>4</td>
<td>NA</td>
<td>5</td>
</tr>
<tr>
<td>classic+wait</td>
<td>17</td>
<td>0</td>
<td>4.2</td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>1-robust</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>41</td>
<td>4</td>
</tr>
<tr>
<td>split</td>
<td>27</td>
<td>0</td>
<td>2</td>
<td>36</td>
<td>3</td>
</tr>
<tr>
<td>w-split</td>
<td>45</td>
<td>0</td>
<td>2.6</td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>rw-split</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>39</td>
<td>0</td>
</tr>
</tbody>
</table>
Talk outline

Part I: Introduction to MAPF
   – Problem formulation, variants and objectives

Part II. Solving MAPF
   – Reduction-based solvers

Part III. From abstract to executable actions
   – Translation vs. model modification

Part IV. Demo

