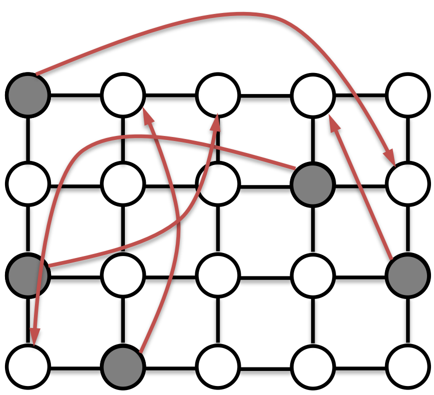
From Abstract Models to Executable Models for Multi-Agent Path Finding on Real Robots

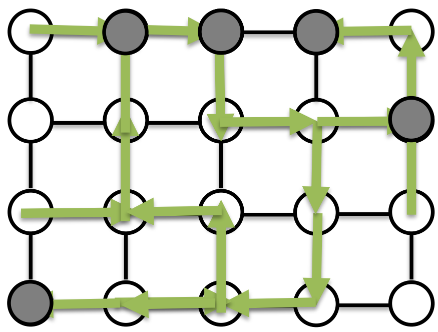
Roman Barták
Charles University, Czech Republic

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

Introduction

What is **multi-agent path finding** (MAPF)?



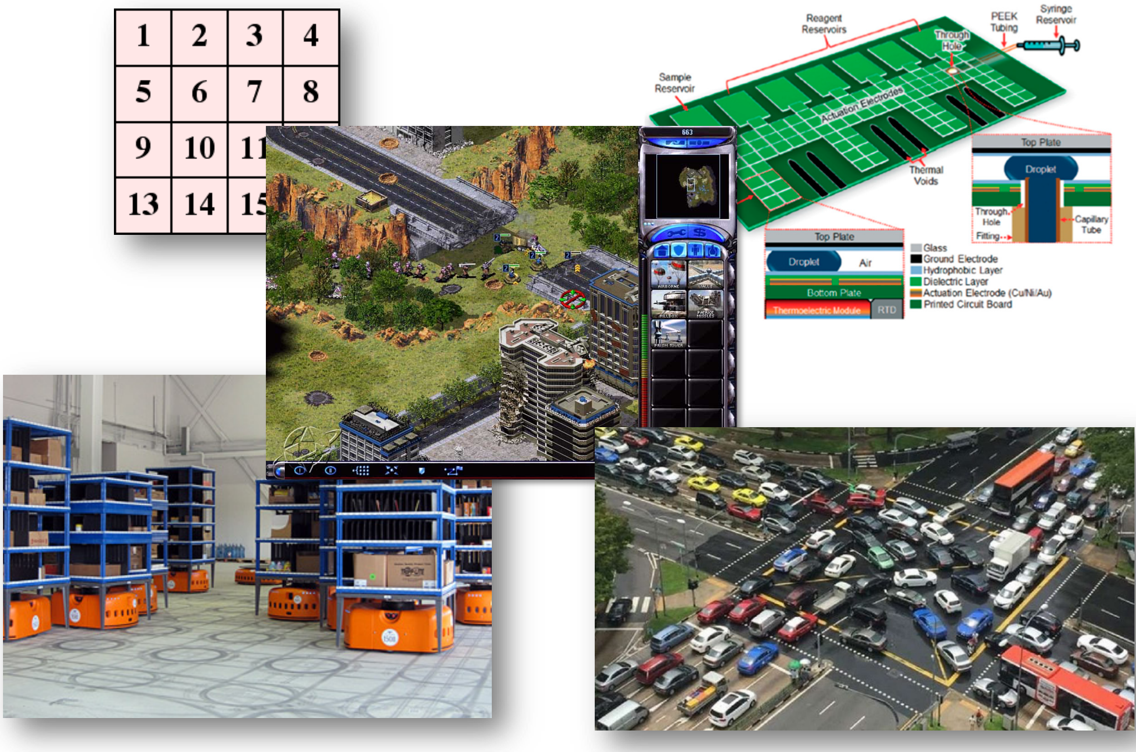


MAPF problem:

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

Talk outline

Part I: Introduction to MAPF

– *Problem formulation, variants and objectives*

Part II. Solving MAPF

– *Reduction-based solvers*

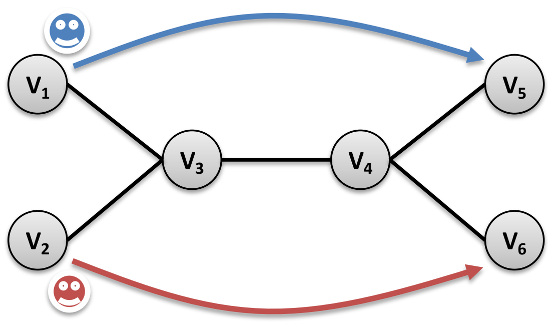
Part III. From abstract to executable actions

– *Translation vs. model modification*

Part IV. Demo

MAPF formulation

- a **graph** (directed or undirected)
- a set of **agents**, each agent is assigned to two locations (nodes) in the graph (start, destination)



Plans

Each agent can perform either **move** (to a neighboring node) or **wait** (in the same node) actions.

Typical assumption:

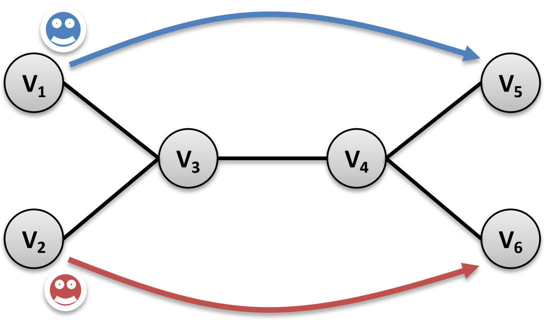
all move and wait actions have identical durations (plans for agents are synchronized)

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

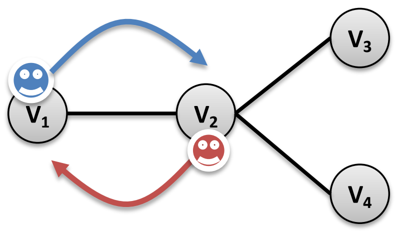
Find **plans** for all agents such that the plans **do not collide in time and space** (no two agents are at the same location at the same time).



time	agent 1	agent 2
0	v_1	v_2
1	wait v_1	move v_3
2	move v_3	move v_4
3	move v_4	move v_6
4	move v_5	wait v_6

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



Agent at $\mathbf{v_i}$ cannot perform **move** $\mathbf{v_j}$ at the same time when agent at $\mathbf{v_j}$ performs **move** $\mathbf{v_i}$

Agents may swap position

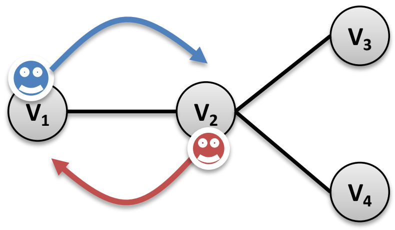
time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents use the same edge at the same time!

Swap is not allowed.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

No-train constraint



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

Agent can approach a node that is currently occupied but will be free before arrival.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	move $\mathbf{v_2}$	move $\mathbf{v_3}$
2	move $\mathbf{v_4}$	move $\mathbf{v_2}$
3	move $\mathbf{v_2}$	move $\mathbf{v_1}$

Agents form a **train**.

Trains may be forbidden.

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	*wait* $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_2}$	*wait* $\mathbf{v_3}$
3	move $\mathbf{v_4}$	*wait* $\mathbf{v_3}$
4	*wait* $\mathbf{v_4}$	move $\mathbf{v_2}$
5	*wait* $\mathbf{v_4}$	move $\mathbf{v_1}$
6	move $\mathbf{v_2}$	*wait* $\mathbf{v_1}$

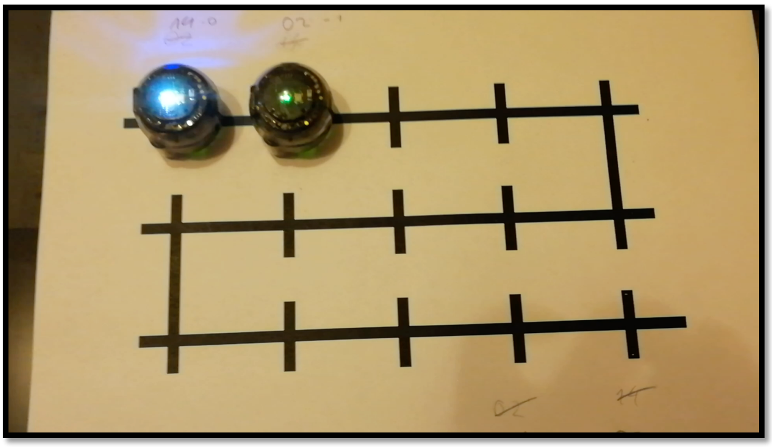
Train collisions

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

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)

time	agent 1	agent 2
0	$\mathbf{v_1}$	$\mathbf{v_2}$
1	wait $\mathbf{v_1}$	move $\mathbf{v_3}$
2	move $\mathbf{v_3}$	move $\mathbf{v_4}$
3	move $\mathbf{v_4}$	move $\mathbf{v_6}$
4	move $\mathbf{v_5}$	wait $\mathbf{v_6}$

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 …)

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 …)

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.

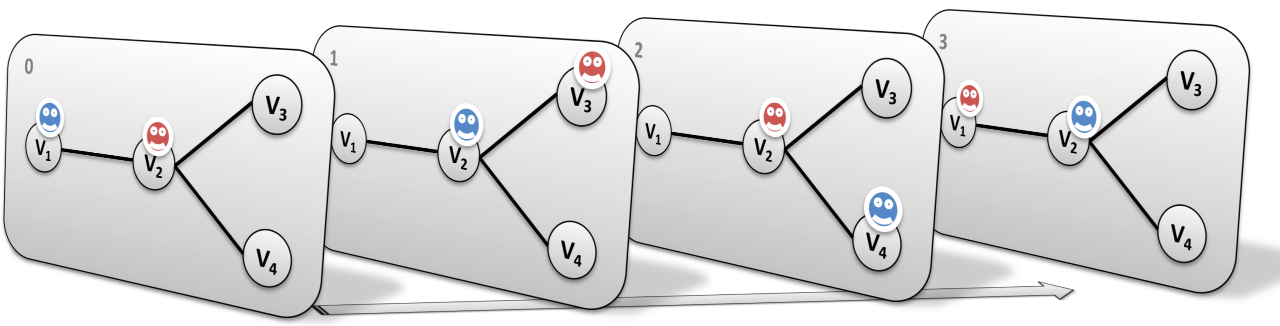
A => B	B *or not* A
sum(Bs) >= 1 (at-least-one(Bs))	disj(Bs)
sum(Bs) = 1	at-most-one(B) *and* at-least-one(B)

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

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.



Classical model

Using **layered graph** describing agent positions at each time step

B_{tav} : agent *a* occupies vertex *v* at time *t*

Constraints:

- each agent occupies exactly one vertex at each time.

  $$\Sigma_{v=1}^{n} B_{tav} = 1 \text{ for } t = 0, \ldots, m, \text{ and } a = 1, \ldots, k.$$

- no two agents occupy the same vertex at any time.

  $$\Sigma_{a=1}^{k} B_{tav} \leq 1 \text{ for } t = 0, \ldots, m, \text{ and } v = 1, \ldots, n.$$

- if agent *a* occupies vertex *v* at time *t*, then *a* occupies a neighboring vertex or stay at *v* at time *t* + 1.

  $$B_{tav} = 1 \Rightarrow \Sigma_{u \in neibs(v)} (B_{(t+1)au}) \geq 1$$

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

```
import sat.

path(N,As) =>
    K = len(As),
    lower_upper_bounds(As,LB,UB),
    between(LB,UB,M),
    B = new_array(M+1,K,N),
    B :: 0..1,

    % Initialize the first and last states
    foreach (A in 1..K)
        (V,FV) = As[A],
        B[1,A,V] = 1,
        B[M+1,A,FV] = 1
    end,

    % Each agent occupies exactly one vertex
    foreach (T in 1..M+1, A in 1..K)
        sum([B[T,A,V] : V in 1..N]) #= 1
    end,

    % No two agents occupy the same vertex
    foreach (T in 1..M+1, V in 1..N)
        sum([B[T,A,V] : A in 1..K]) #=< 1
    end,

    % Every transition is valid
    foreach (T in 1..M, A in 1..K, V in 1..N)
        neibs(V,Neibs),
        B[T,A,V] #=>
        sum([B[T+1,A,U] : U in Neibs]) #>= 1
    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

```
foreach(T in 1..M1, A in 1..K, V in 1..N)
    B[T,A,V] #=> sum([B[Prev,A2,V] :
                  A2 in 1..K, A2!=A,
                  Prev in max(1,T-L)..T]) #= 0
end
```

K-robustness

Talk outline

Part I: Introduction to MAPF

– *Problem formulation, variants and objectives*

Part II. Solving MAPF

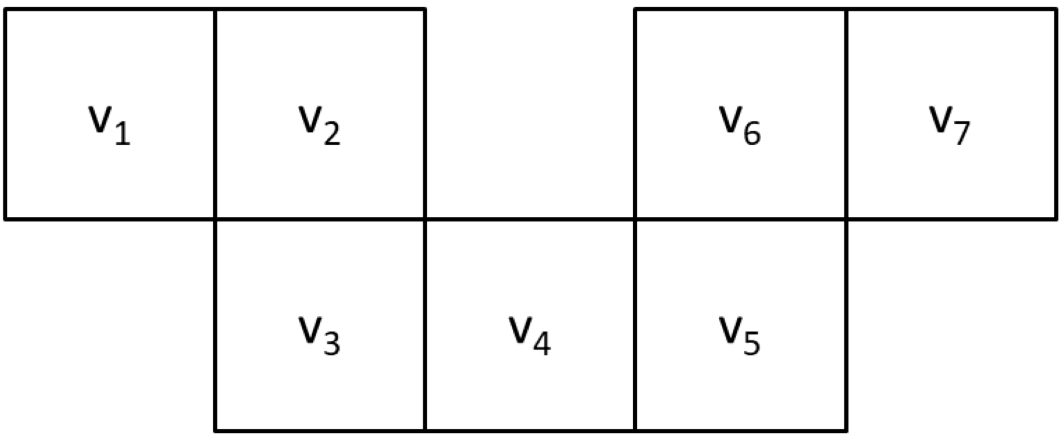
– *Reduction-based solvers*

Part III. From abstract to executable actions

– *Translation vs. model modification*

Part IV. Demo

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 actions:

- move
- wait

Times:

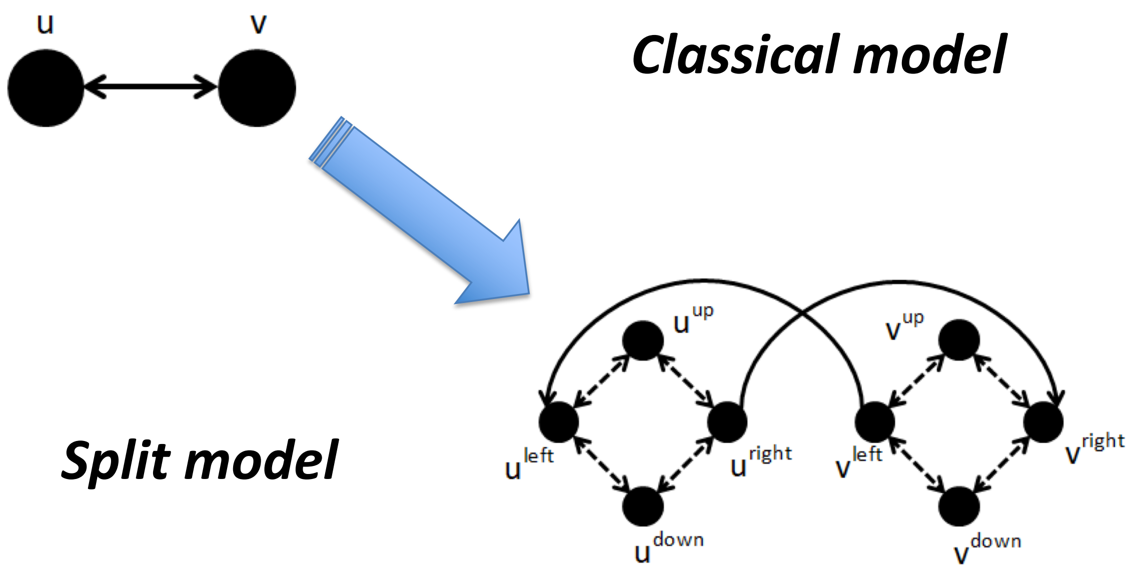
t_t – time to turn left/right
t_f – time to move forward

Executable actions:

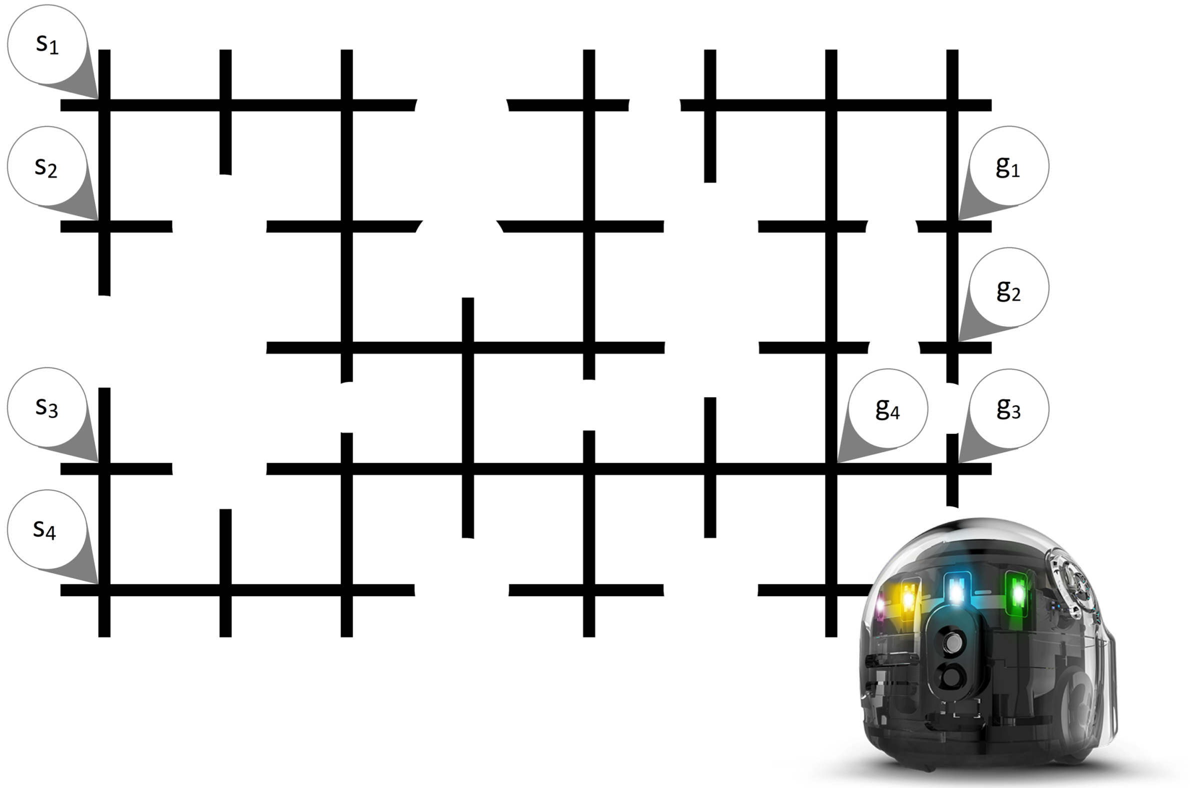
- move forward
- wait
- turn left/right + move
- turn back and move

classic	classic+wait
t_f	$t_f + 2*t_t$
$t_f + t_t/2$	$t_f + 2*t_t$
$t_f + t_t$	$t_f + 2*t_t$
$t_f + 2*t_t$	$t_f + 2*t_t$

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



Experiment setting



Some results

	Computed Makespan	Failed Runs	Number of Collisions	Total Time [s]	Max Δ time [s]
classic	17	5	4	NA	5
classic+wait	17	0	4.2	53	0
1-robust	19	0	0	41	4
split	27	0	2	36	3
w-split	45	0	2.6	39	0
rw-split	47	0	0	39	0

Part I: Introduction to MAPF

– *Problem formulation, variants and objectives*

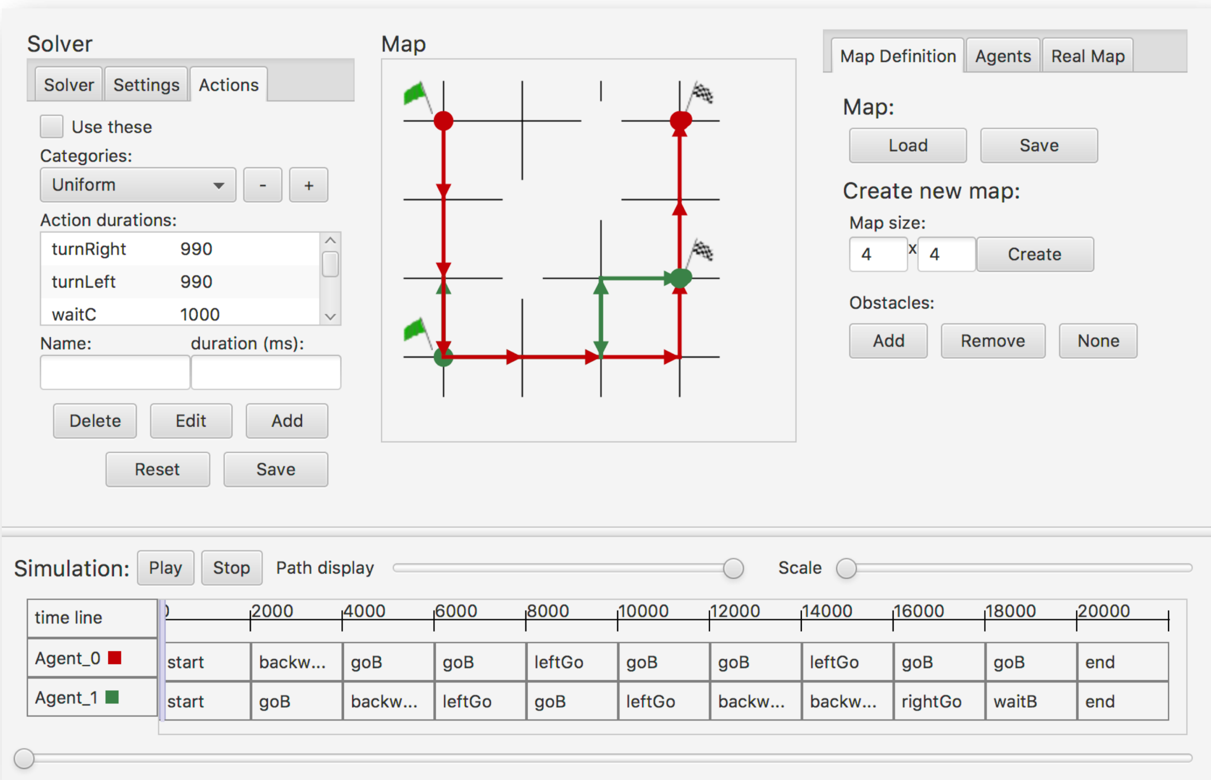
Part II. Solving MAPF

– *Reduction-based solvers*

Part III. From abstract to executable actions

– *Translation vs. model modification*

Part IV. Demo



Atzmon, D.; Felner, A.; Stern, R.; Wagner, G.; Barták, R.; and Zhou, N. 2017. **k-robust multi-agent path finding**. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS)*, 157–158.

Barták, R., Švancara, J., Vlk, M. 2018. **A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs**. In Proceedings of AAMAS 2018, Stockholm, Sweden, July 11-13.

Barták, R.; Zhou, N.-F.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. **Modeling and solving the multi-agent pathfinding problem in Picat**. In *29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 959–966. IEEE Computer Society.

Barták, R.; Švancara, J.; Škopková, V.; and Nohejl, D. 2018. **Multi-agent path finding on real robots: First experience with ozobots**. In *Advances in Artificial Intelligence – IBERAMIA 2018*. Springer.

Roman Barták
Charles University, Faculty of Mathematics and Physics
bartak@ktiml.mff.cuni.cz