Multi-Agent Systems

(Classical) Multi-Agent Path Finding

Albert-Ludwigs-Universität Freiburg

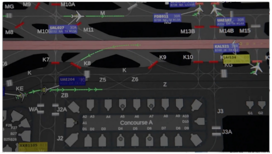
Bernhard Nebel, Rolf Bergdoll, and Thorsten Engesser

Winter Term 2019/20

Agents moving in a spatial environment

A central problem in many applications is the coordinated movement of agents/robots/vehicles in a given spatial environment.

Logistic robots (KARIS)



Airport ground traffic control (atrics)

Multi-agent path finding

Definition (Multi-agent path finding (MAPF) problem)

Given a set of *agents* A, a (perhaps directed) *graph* $G = (V, E)$, an *initial state* modelled by an injective function $\alpha_0 : A \to V$, and a *goal state* modelled by another injective function α_*, can α_0 be *transformed* into α_* by *movements of single agents* without collisions?

Multi-agent path finding

Definition (Multi-agent path finding (MAPF) problem)

Given a set of *agents* A, a (perhaps directed) *graph* $G = (V, E)$, an *initial state* modelled by an injective function $\alpha_0 : A \to V$, and a *goal state* modelled by another injective function α_*, can α_0 be *transformed* into α_* by *movements of single agents* without collisions?

- *Existence problem*: Does there exist a successful sequence of movements (= *plan*)?

Multi-agent path finding

Definition (Multi-agent path finding (MAPF) problem)

Given a set of *agents* A, a (perhaps directed) *graph* $G = (V, E)$, an *initial state* modelled by an injective function $\alpha_0 : A \rightarrow V$, and a *goal state* modelled by another injective function α_*, can α_0 be *transformed* into α_* by *movements of single agents* without collisions?

- *Existence problem*: Does there exist a successful sequence of movements (= *plan*)?
- *Bounded existence problem*: Does there exist a plan of a given *length k* or less?

Multi-agent path finding

Definition (Multi-agent path finding (MAPF) problem)

Given a set of *agents* A, a (perhaps directed) *graph* $G = (V, E)$, an *initial state* modelled by an injective function $\alpha_0 : A \to V$, and a *goal state* modelled by another injective function α_*, can α_0 be *transformed* into α_* by *movements of single agents* without collisions?

- *Existence problem*: Does there exist a successful sequence of movements (= *plan*)?
- *Bounded existence problem*: Does there exist a plan of a given *length k* or less?
- *Plan generation problem*: Generate a plan.

Multi-agent path finding

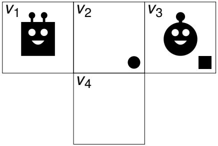
Definition (Multi-agent path finding (MAPF) problem)

Given a set of *agents* A, a (perhaps directed) *graph* $G = (V, E)$, an *initial state* modelled by an injective function $\alpha_0 : A \to V$, and a *goal state* modelled by another injective function α_*, can α_0 be *transformed* into α_* by *movements of single agents* without collisions?

- *Existence problem*: Does there exist a successful sequence of movements (= *plan*)?
- *Bounded existence problem*: Does there exist a plan of a given *length k* or less?
- *Plan generation problem*: Generate a plan.
- *Optimal plan generation problem*: Generate a shortest plan.

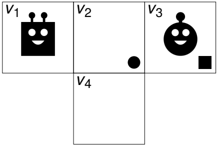
Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?



Example

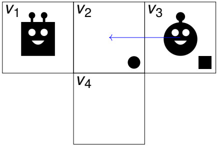
Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?



$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

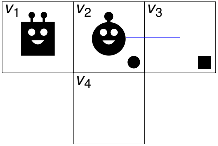


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan:

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

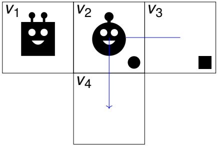


$G = (V,E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

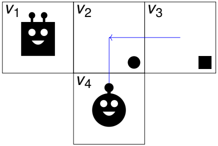


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

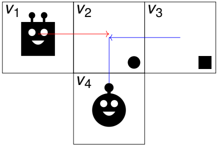


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

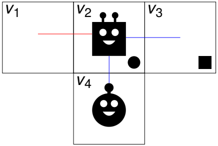


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

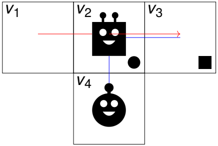


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

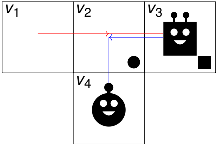


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

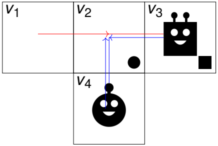


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2), (S, v_2, v_3),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?

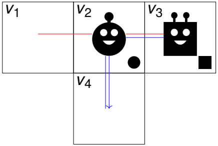


$G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2), (S, v_2, v_3),

Example

Can we find a (central) plan to move the square robot S to v_3 and the circle robot C to v_2?



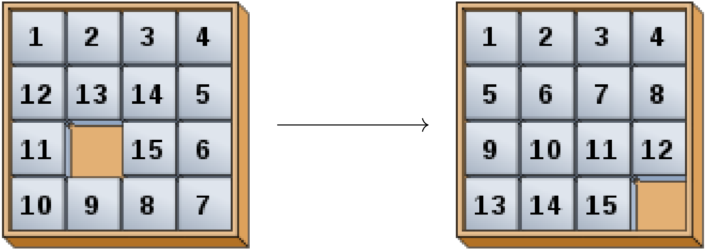
$G = (V,E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \Big\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_4\}\Big\}$
$A = \{S, C\}$ and $\alpha_0(S) = v_1, \alpha_0(C) = v_3, \alpha_*(S) = v_3, \alpha_*(C) = v_2$

Plan: (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2), (S, v_2, v_3),(C, v_4, v_2).

A special case: 15-puzzle

Pictures from Wikipedia article on 15-Puzzle

A special case: 15-puzzle



Pictures from Wikipedia article on 15-Puzzle

Lecture plan

- MAPF: variations, algorithms, complexity
- Distributed MAPF (each agent plans on it own): DMAPF
- Distributed MAPF with destination uncertainty: MAPF/DU

Sequential MAPF

- *Sequential MAPF* (or pebble motion on a graph) allows only one agent to move per time step.
- An agent $a \in A$ can move in one step from $s \in V$ to $t \in V$ transforming α to α', if
  - $\alpha(a) = s$,
  - $\langle s,t \rangle \in E$,
  - there is no agent b such that $\alpha(b) = t$.
- In this case, α' is determined as follows:
  - $\alpha'(a) = t$,
  - for all agents $b \neq a : \alpha(b) = \alpha'(b)$,
- One usually wants to minimize the number of single movements (= *sum-of-cost* over all agents)

Parallel MAPF

- *Parallel MAPF* allows many agents to move in parallel, provided they do not collide.
- Two models:
  - *Parallel*: A chain of agents can move provided the first agent can move on a an unoccupied vertex.
  - *Parallel with rotations*: A closed cycle in move synchronously.
- In both cases, one is usually interested in the number of parallel steps (= *make-span*).
- However, also the sum-of-cost is sometimes considered.

Anonymous MAPF

- There is a set of agents and a set of targets (of the same cardinality as the agent set).
- Each target must be reached by one agent.
- This means one first has to assign a target and then to solve the original MAPF problem.
- Interestingly, the problem as a whole is easier to solve (using flow-based techniques).

Types of MAPF algorithms

- **A^*-based** algorithm (optimal)

Types of MAPF algorithms

- **A*-based** algorithm (optimal)
- *Conflict-based search* (optimal)

Types of MAPF algorithms

- **A*-based** algorithm (optimal)
- *Conflict-based search* (optimal)
- *Reduction-based approaches*: Translate MAPF to *SAT*, *ASP* or to a *CSP* (usually optimal)

Types of MAPF algorithms

- **A*-based** algorithm (optimal)
- *Conflict-based search* (optimal)
- *Reduction-based approaches*: Translate MAPF to *SAT*, *ASP* or to a *CSP* (usually optimal)
- *Suboptimal search-based algorithms* (may even be incomplete): **Cooperative A*** (CA*), *Hierarchical Cooperative A** (HCA*) and *Windowed HCA** (WHCA*).

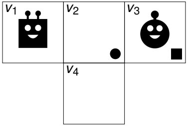
Types of MAPF algorithms

- **A^*-based** algorithm (optimal)
- *Conflict-based search* (optimal)
- *Reduction-based approaches*: Translate MAPF to *SAT*, *ASP* or to a *CSP* (usually optimal)
- *Suboptimal search-based algorithms* (may even be incomplete): **Cooperative A^*** (CA*), *Hierarchical Cooperative A^** (HCA*) and *Windowed HCA** (WHCA*).
- *Rule-based algorithms*: *Kornhauser's algorithm*, *Push-and-Rotate*, **BIBOX**, …(complete on a given class of graphs, but suboptimal)

A*-based algorithm

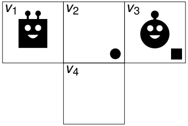
- Define state space:
  - A state is an assignment of agents to vertices (modelled by a function α)
  - There is a transition from one state α to α' iff there is a legal move from α to α' according to the appropriate semantics (sequential, parallel, or parallel with rotations)
- Search in this state space using the A* algorithm.
- Possible *heuristic estimator*: Sum or maximum over the length of the individual movement plans (ignoring other agents).
- **Problem**: Large *branching factor* because of many agents that can move.

Example: State space for A* algorithm



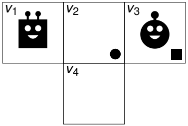
Convention: Function α is represented by $\langle \alpha(S), \alpha(C) \rangle$

Example: State space for A* algorithm



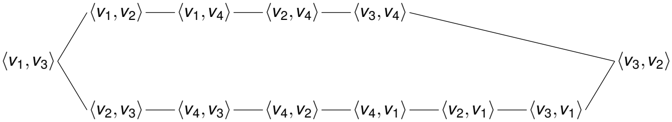
Convention: Function α is represented by $\langle \alpha(S), \alpha(C) \rangle$
Question: How many states?

Example: State space for A* algorithm

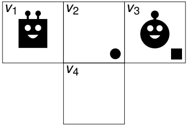


Convention: Function α is represented by $\langle \alpha(S), \alpha(C) \rangle$

Question: How many states?

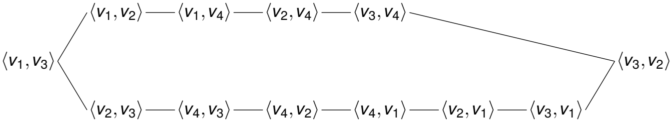


Example: State space for A^* algorithm



Convention: Function α is represented by $\langle \alpha(S), \alpha(C) \rangle$

Question: How many states?



Question: Heuristic value for states $\langle v_1, v_2 \rangle$ and $\langle v_2, v_3 \rangle$ under the sum-aggregation?

CA*

- Problems with A^* on MAPF state space:

CA*

- Problems with A^* on MAPF state space:
  - exponential state space

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- CA^*: Decoupled planning in space & time

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- CA^*: Decoupled planning in space & time
  - Order agents linearly and then plan for each agent separately a (shortest) path.

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- *CA**: Decoupled planning in space & time
  - Order agents linearly and then plan for each agent separately a (shortest) path.
  - Store each path in a *reservation table*, which stores for each node at which time point it is occupied.

CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- CA^*: Decoupled planning in space & time
  - Order agents linearly and then plan for each agent separately a (shortest) path.
  - Store each path in a *reservation table*, which stores for each node at which time point it is occupied.
  - When planning, take the reservation table into account and avoid nodes at time points, when they are reserved for other agents; wait action is possible.

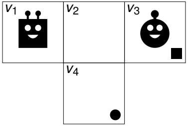
CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- CA^*: Decoupled planning in space & time
  - Order agents linearly and then plan for each agent separately a (shortest) path.
  - Store each path in a *reservation table*, which stores for each node at which time point it is occupied.
  - When planning, take the reservation table into account and avoid nodes at time points, when they are reserved for other agents; wait action is possible.
  - Solvability depends on chosen order.

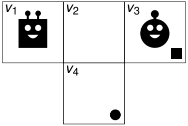
CA*

- Problems with A^* on MAPF state space:
  - exponential state space, i.e., $m!/(m-n)!$ with m nodes and n agents;
  - huge branching factor: $n \times d$ for sequential and d^n for parallel MAPF for graphs with maximal degree d.
- *CA**: Decoupled planning in space & time
  - Order agents linearly and then plan for each agent separately a (shortest) path.
  - Store each path in a *reservation table*, which stores for each node at which time point it is occupied.
  - When planning, take the reservation table into account and avoid nodes at time points, when they are reserved for other agents; wait action is possible.
  - Solvability depends on chosen order.
  - Our small example is not solvable (shortest paths lead to head on collision), but small modification works.

Example CA* run

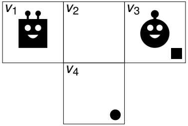


Example CA* run



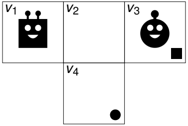
- Linear order: $\langle C, S \rangle$

Example CA* run



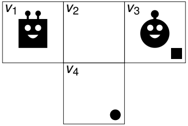
- Linear order: $\langle C, S \rangle$
- Plan for C:
- Reservation table: $(0 : v_3)$

Example CA* run



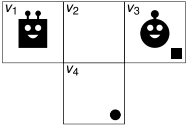
- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2)
- Reservation table: $(0 : v_3)$

Example CA* run



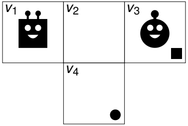
- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$

Example CA* run



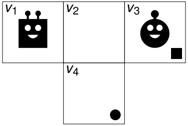
- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$

Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$

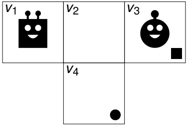
Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S:

- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$

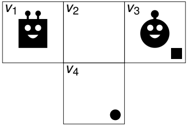
Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1)

- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$

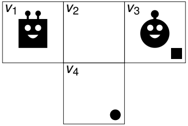
Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1)

- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$

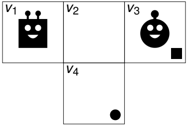
Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1), (S, v_1, v_2)

- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$

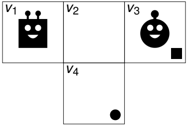
Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1), (S, v_1, v_2)

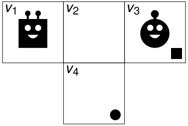
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$, $(2 : v_2)$

Example CA* run



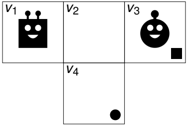
- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1), (S, v_1, v_2), (S, v_2, v_3)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$, $(2 : v_2)$

Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1), (S, v_1, v_2), (S, v_2, v_3)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$, $(2 : v_2)$, $(3 - n : v_3)$

Example CA* run



- Linear order: $\langle C, S \rangle$
- Plan for C: (C, v_3, v_2), (C, v_2, v_4)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_4)$
- Plan for S: *wait* (because v_2 occupied at time 1), (S, v_1, v_2), (S, v_2, v_3)
- Reservation table: $(0 : v_3)$, $(1 : v_2)$, $(2 - n : v_5)$, $(0 : v_1), (1 : v_1)$, $(2 : v_2)$, $(3 - n : v_3)$
- Not solvable with different order!

BIBOX

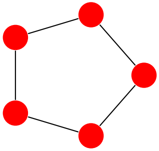
BIBOX is a rule-based algorithm that is complete on all *bi-connected* graphs with at least two unoccupied nodes in the graph.

Definition

A graph $G = (V, E)$ is *connected* iff $|V| \geq 2$ and there is *path* between each pair of nodes $s, t \in V$. A graph is *bi-connected* iff $|V| \geq 3$ and for each $v \in V$, the graph $(V - \{v\}, E')$ with $E' = \left\{ \{x, y\} \in E \,|\, x, y \neq v \right\}$ is connected.

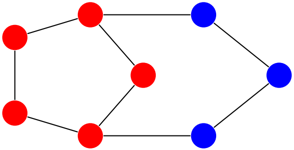
Loop decomposition

Every bi-connected graph can be constructed from a *cycle* by adding *loops* iteratively.



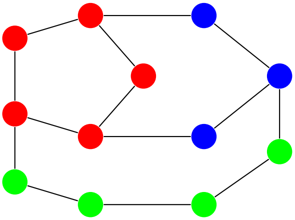
Loop decomposition

Every bi-connected graph can be constructed from a *cycle* by adding *loops* iteratively.



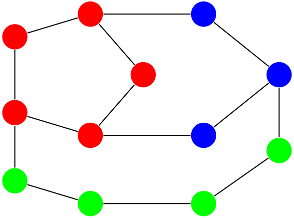
Loop decomposition

Every bi-connected graph can be constructed from a *cycle* by adding *loops* iteratively.



Loop decomposition

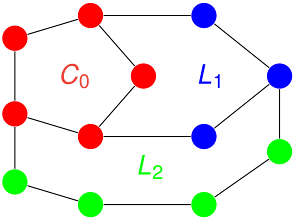
Every bi-connected graph can be constructed from a *cycle* by adding *loops* iteratively.



A *loop decomposition* into a basic cycle and additional loops can be done in time $O(|V|^2)$.

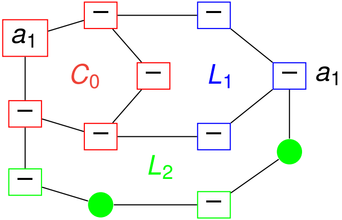
Loop decomposition

Every bi-connected graph can be constructed from a *cycle* by adding *loops* iteratively.

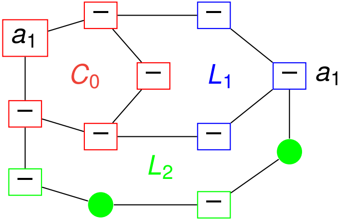


A *loop decomposition* into a basic cycle and additional loops can be done in time $O(|V|^2)$.
Let us name them C_0, L_1, L_2, ..., where the index depends on the time when the loop is added.

Moving unoccupied nodes and agents around

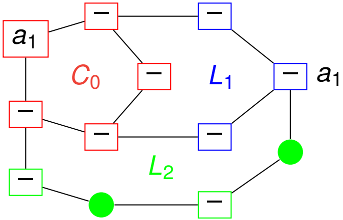


Moving unoccupied nodes and agents around



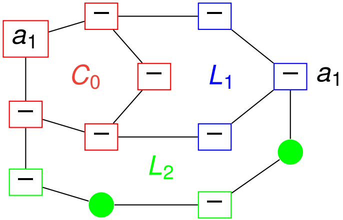
- An unoccupied place can be sent to any node.

Moving unoccupied nodes and agents around



- An unoccupied place can be sent to any node.
- Any agent can be sent to any node by rotating the agents in a cycle or in the loop.

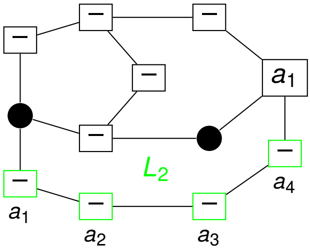
Moving unoccupied nodes and agents around



- An unoccupied place can be sent to any node.
- Any agent can be sent to any node by rotating the agents in a cycle or in the loop.
- This can be done without disturbing loops with a higher index than the one the agent starts and finishes in.

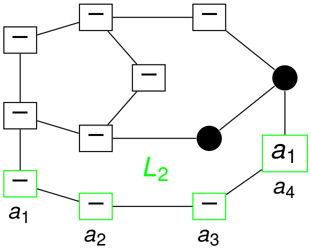
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



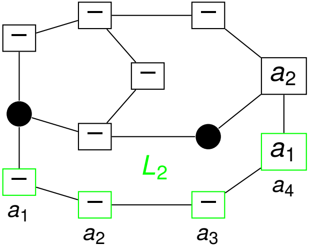
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



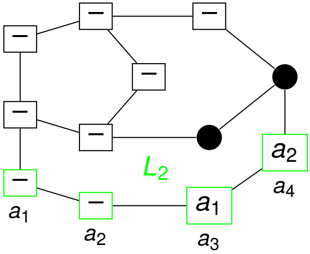
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



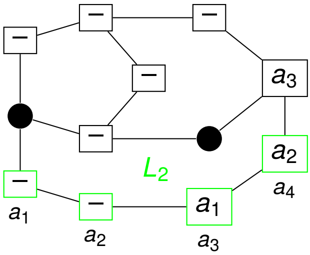
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



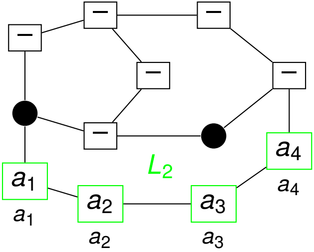
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



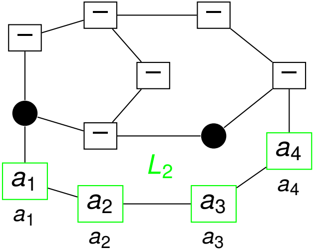
Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



Filling loops

- Starting with highest-index loop: Move agents to destination loop, then shift agents to their destinations.
- Special case: When agents are already in the destination loop, they have to be rotated out of the loop.



- When done with one loop, repeat for next one with next lower index.

Reordering agents in the cycle

- Assumption: The destinations for the empty places are in the cycle C_0 (can be relaxed).
- If the agents are in the right order, just rotate them to their destinations.
- Otherwise reorder by successively take one out and re-insert.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.
- Moving one agent to an arbitrary position can be done in $O(|V|^2)$ steps.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.
- Moving one agent to an arbitrary position can be done in $O(|V|^2)$ steps.
- Moving one agent to its final destination in a loop needs $O(|V|^2)$.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.
- Moving one agent to an arbitrary position can be done in $O(|V|^2)$ steps.
- Moving one agent to its final destination in a loop needs $O(|V|^2)$.
- Since this has to be done $O(|V|)$ times, we need overall $O(|V|^3)$ steps.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.
- Moving one agent to an arbitrary position can be done in $O(|V|^2)$ steps.
- Moving one agent to its final destination in a loop needs $O(|V|^2)$.
- Since this has to be done $O(|V|)$ times, we need overall $O(|V|^3)$ steps.
- Reordering in the final cycle is also bounded by $O(|V|^3)$.

Runtime and plan length estimation

- Moving an empty place around is in $O(|V|)$ steps.
- Moving one agent to an arbitrary position can be done in $O(|V|^2)$ steps.
- Moving one agent to its final destination in a loop needs $O(|V|^2)$.
- Since this has to be done $O(|V|)$ times, we need overall $O(|V|^3)$ steps.
- Reordering in the final cycle is also bounded by $O(|V|^3)$.

→ Runtime and number of steps is bounded by $O(|V|^3)$.

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.
- Generation: $O(|V|^3)$, generating the same number of steps, again using Kornhauser's algorithm or BIBOX (on a smaller instance set).

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.
- Generation: $O(|V|^3)$, generating the same number of steps, again using Kornhauser's algorithm or BIBOX (on a smaller instance set).
- Bounded existence: Is definitely in NP

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.
- Generation: $O(|V|^3)$, generating the same number of steps, again using Kornhauser's algorithm or BIBOX (on a smaller instance set).
- Bounded existence: Is definitely in NP
  - If there exists a solution, then it is polynomially bounded.

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.
- Generation: $O(|V|^3)$, generating the same number of steps, again using Kornhauser's algorithm or BIBOX (on a smaller instance set).
- Bounded existence: Is definitely in NP
  - If there exists a solution, then it is polynomially bounded.
  - A solution candidate can be checked in polynomial time for satisfying the conditions of being a movement plan with k of steps or less.

Computational Complexity of MAPF

- Existence: For arbitrary graphs with at least one empty place, the problem is polynomial ($O(|V|^3)$ using Kornhauser's algorithm). For BIBOX on bi-connected with at least two empty places also cubic, but smaller constant.
- Generation: $O(|V|^3)$, generating the same number of steps, again using Kornhauser's algorithm or BIBOX (on a smaller instance set).
- Bounded existence: Is definitely in NP
  - If there exists a solution, then it is polynomially bounded.
  - A solution candidate can be checked in polynomial time for satisfying the conditions of being a movement plan with k of steps or less.
- Question: Is the problem also NP-hard?

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

Example

$U = \{1, 2, 3, 4, 5, 6\}$

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

Example

$U = \{1,2,3,4,5,6\}$

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

Example

$U = \{1,2,3,4,5,6\}$
$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$
$C'_1 = \{\{1,2,3\},\{2,3,4\}\}$ is not a cover.

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

Example

$U = \{1,2,3,4,5,6\}$

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$

$C'_1 = \{\{1,2,3\},\{2,3,4\}\}$ is not a cover.

$C'_2 = \{\{1,2,3\},\{2,3,4\},\{1,5,6\}\}$ is not an exact cover.

The Exact Cover By 3-Sets (X3C) Problem

Definition (Exact Cover By 3-Sets (X3C) Problem)

Given a set of elements U and a collection of subsets $C = \{s_j\}$ with $s_j \subseteq U$ and $|s_j| = 3$. Is there a sub-collection of subsets $C' \subseteq C$ such that $\bigcup_{s \in C'} s = U$ and all subsets in C' are pairwise disjoint, i.e., $s_a \cap s_b = \emptyset$ for each $s_a, s_b \in C'$ with $s_a \neq s_b$?

X3C is NP-complete.

Example

$U = \{1,2,3,4,5,6\}$

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$

$C'_1 = \{\{1,2,3\},\{2,3,4\}\}$ is not a cover.

$C'_2 = \{\{1,2,3\},\{2,3,4\},\{1,5,6\}\}$ is not an exact cover.

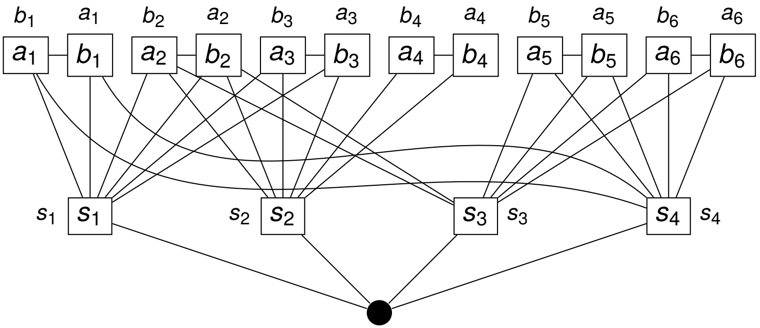
$C'_3 = \{\{2,3,4\},\{1,5,6\}\}$ is an exact cover.

NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$

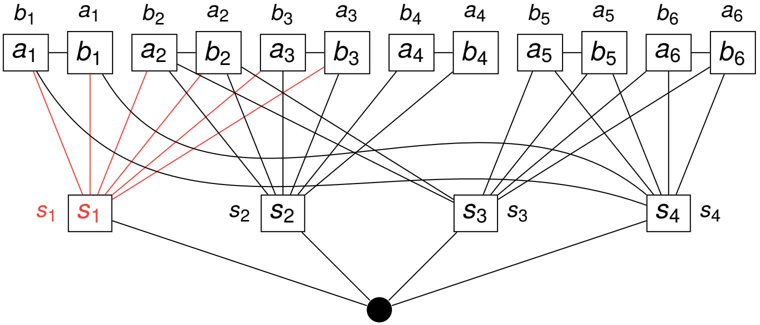
NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$



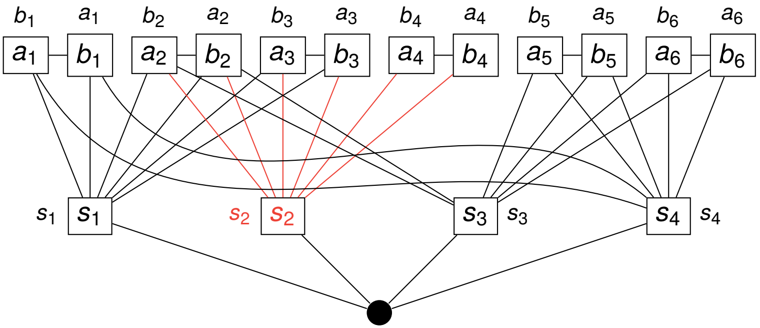
NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$



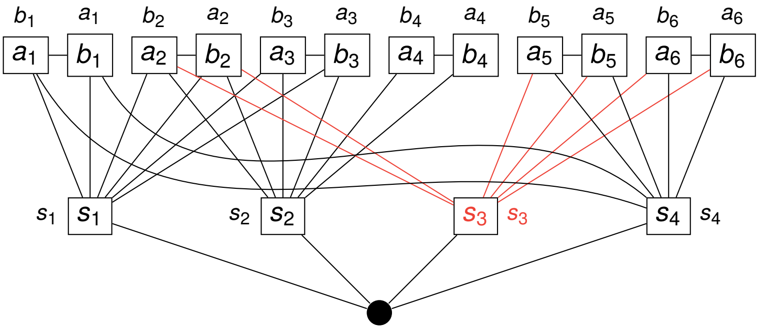
NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\}, \{2,3,4\}, \{2,5,6\}, \{1,5,6\}\}$



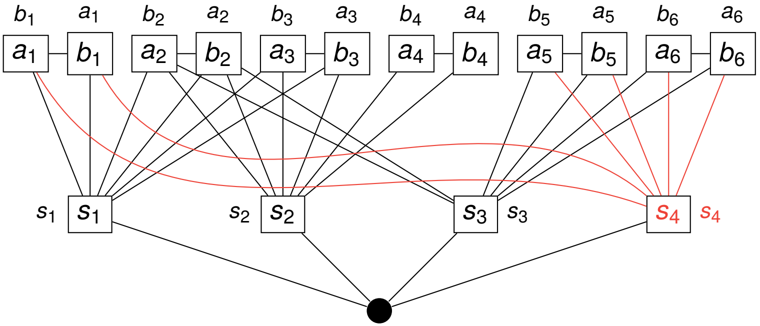
NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$



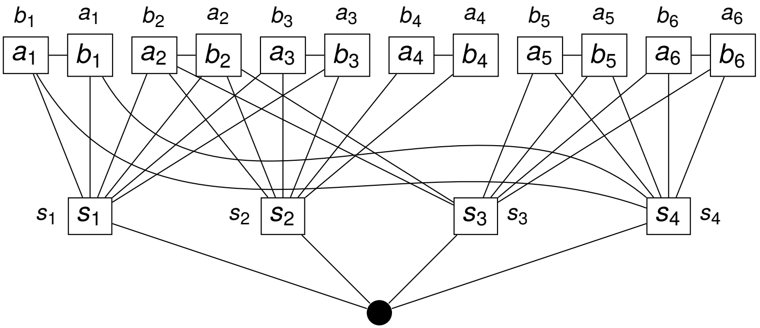
NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$



NP-hardness of MAPF: Reduction from X3C

$C = \{\{1,2,3\},\{2,3,4\},\{2,5,6\},\{1,5,6\}\}$



Claim: There is an exact cover by 3-sets iff the constructed MAPF instance can be solved in at most $k = 11/3|U|$ moves.

Literature (1)

D. Kornhauser, G. L. Miller, and P. G. Spirakis.
Coordinating pebble motion on graphs, the diameter of permutation groups, and applications.
In *25th Annual Symposium on Foundations of Computer Science (FOCS-84)*, pages 241–250, 1984.

O. Goldreich.
Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard.
In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 1–5. 2011.

Literature (2)

H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. HÃűnig, T. K. Satish Kumar, T. Uras, H. Xu, C. A. Tovey, G. Sharon:
Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios.
CoRR abs/1702.05515, 2017.

A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek.
Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges.
In *Proceedings of the Tenth International Symposium on Combinatorial Search (SOCS-17)*, pages 29–37, 2017.

P Surynek.
A novel approach to path planning for multiple robots in bi-connected graphs.
In *Proc. 2009 IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 3613–3619, 2009.