Non-optimal Multi-agent Pathfinding is Solved (Since 1984)

Gabriele Röger Malte Helmert

University of Basel, Switzerland

Workshop on Multi-agent Pathfinding
AAAI 2012

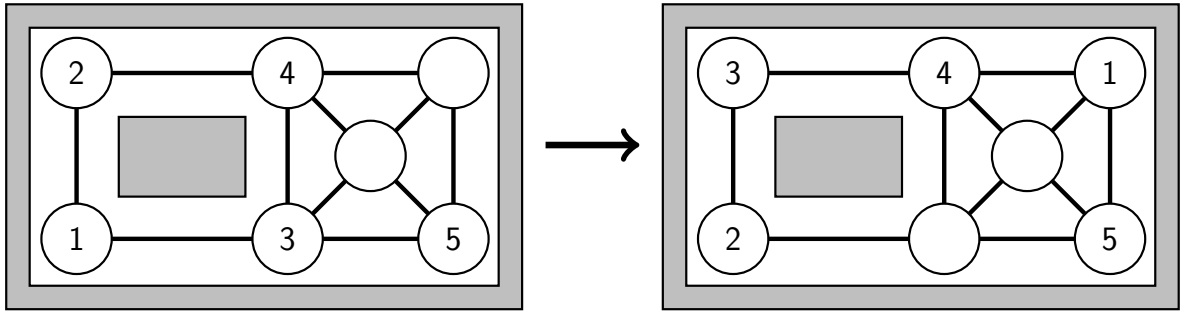
Explaining the title

Multi-agent pathfinding

Non-optimal multi-agent pathfinding is solved (since 1984)

Multi-agent pathfinding

Non-optimal multi-agent pathfinding is solved (since 1984)



Given

- undirected graph
- each node either contains an agent or is unoccupied
- goal node for each agent
- agents can move to adjacent node if unoccupied

Find sequence of moves which brings each agent to its goal.

Non-optimal multi-agent pathfinding

Non-optimal multi-agent pathfinding is solved (since 1984)

Non-optimal multi-agent pathfinding

Non-optimal multi-agent pathfinding is solved (since 1984)

Setting of this paper:

- no optimality requirements
- no attempt to keep number of solution steps low (but some words on quality guarantees later)
- require polynomial-time algorithms

What can we do in polynomial time?

optimal solutions:

- Ratner & Warmuth, AAAI 1986:
  Finding a Shortest Solution for the $N \times N$ Extension of the 15-PUZZLE Is Intractable.

suboptimal solutions:

- SoCS 2011
- IJCAI 2011
- IROS 2011
- JAIR 2011

$\rightsquigarrow$ polynomial algorithms for different problem fragments

- e.g.: trees, SLIDABLE instances, ...

Quotes

Quote #1 (SoCS 2011)

"[Our] work is just one step in classifying problems that can be solved in polynomial time."

Quote #2 (JAIR 2011)

"[Our approach] identifies classes of multi-agent planning problems that can be solved in polynomial time."

Quote #3 (IJCAI 2011)

"In comparison to existing alternatives that provide completeness guarantees for certain problem subclasses, the proposed method provides similar guarantees for a much wider problem class."

Non-optimal multi-agent pathfinding is solved

Non-optimal multi-agent pathfinding is solved (since 1984)

Non-optimal multi-agent pathfinding is solved

Non-optimal multi-agent pathfinding is solved (since 1984)

What do we mean by solved?

- arbitrary instances: determine solution existence in $O(n)$ (?)
- solvable instances: produce solutions in $O(n^3)$
- some instances require $\Theta(n^3)$ steps for optimal solutions

Non-optimal multi-agent pathfinding is solved

Non-optimal multi-agent pathfinding is solved (since 1984)

What do we mean by solved?

- arbitrary instances: determine solution existence in $O(n)$ (?)
- solvable instances: produce solutions in $O(n^3)$
- some instances require $\Theta(n^3)$ steps for optimal solutions

Of course this leaves many practical issues open!

This is (only) about the quest for tractable fragments.

Non-optimal multi-agent pathfinding is solved (since 1984)

Non-optimal multi-agent pathfinding is solved (since 1984)

Non-optimal multi-agent pathfinding is solved (since 1984)

Non-optimal multi-agent pathfinding is solved (since 1984)

The work we present here is not ours:

- Wilson (1974): one blank position, biconnected graphs
- Kornhauser, Miller and Spirakis (1984): general case

The Kornhauser et al. algorithm

Reading Wilson (1974) and Kornhauser et al. (1984)

- Wilson and Kornhauser et al. papers are no easy reads
- not written from an algorithmic perspective
- not quite your typical AI venues:
  - Wilson (1974): Journal of Combinatorial Theory (Series B)
  - Kornhauser et al. (1984): Foundations of Computer Science
- Kornhauser et al.: non-existent "final version" for details
- details can be found in Kornhauser's M.Sc. thesis
- (to our knowledge) never implemented

rest of the presentation:
brief introduction to the Kornhauser et al. algorithm

Kornhauser et al.: overview

Overview:

1. Move blanks in goal configuration to initial blank positions. (This will form end of the solution.)
2. Derive subproblems, each defined by a set of agents that can reach the same set of nodes.
3. Solve each subproblem using permutation group theory.

First step

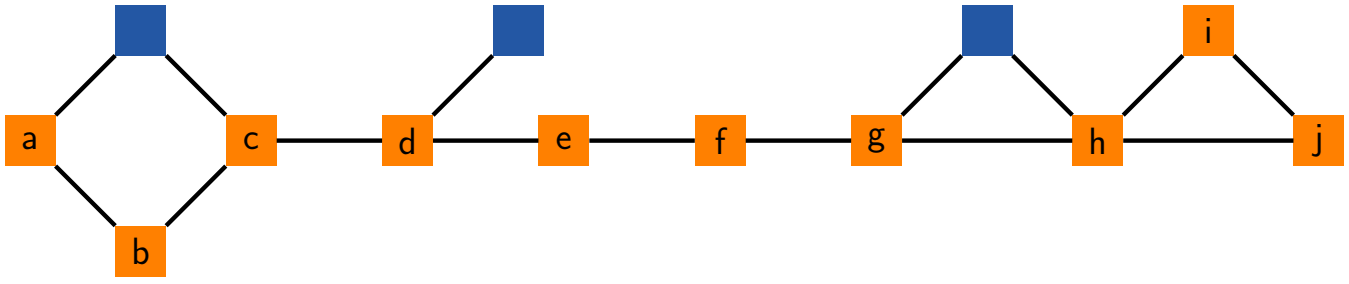
Kornhauser et al.: first step

Overview:

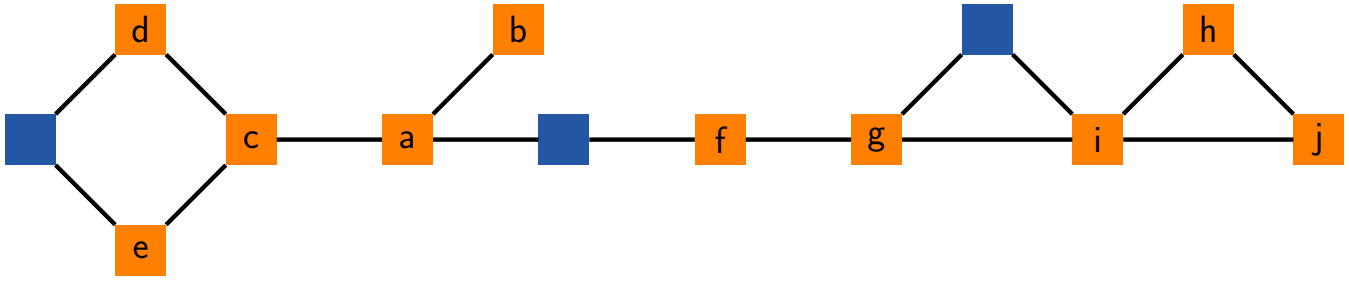
1. Move blanks in goal configuration to initial blank positions. (This will form end of the solution.)
2. Derive subproblems, each defined by a set of agents that can reach the same set of nodes.
3. Solve each subproblem using permutation group theory.

First step: fix the blank positions

Initial configuration:



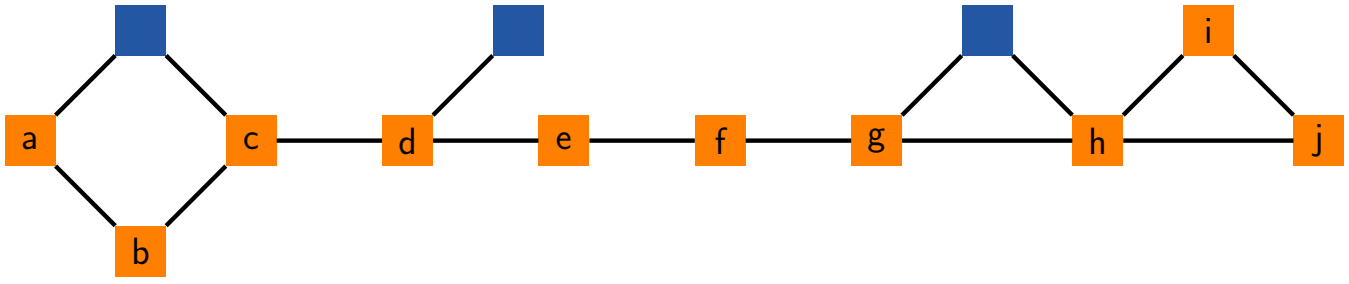
Goal configuration:



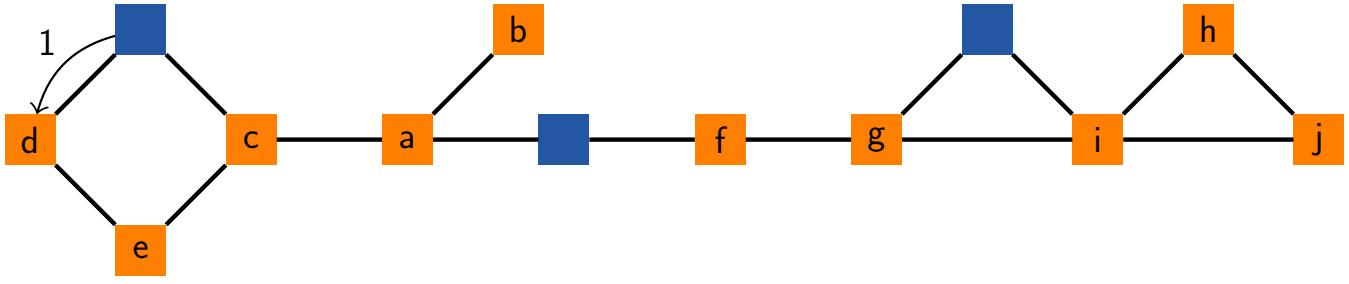
can produce $O(n^2)$ moves; goal configuration computable in $O(n)$

First step: fix the blank positions

Initial configuration:



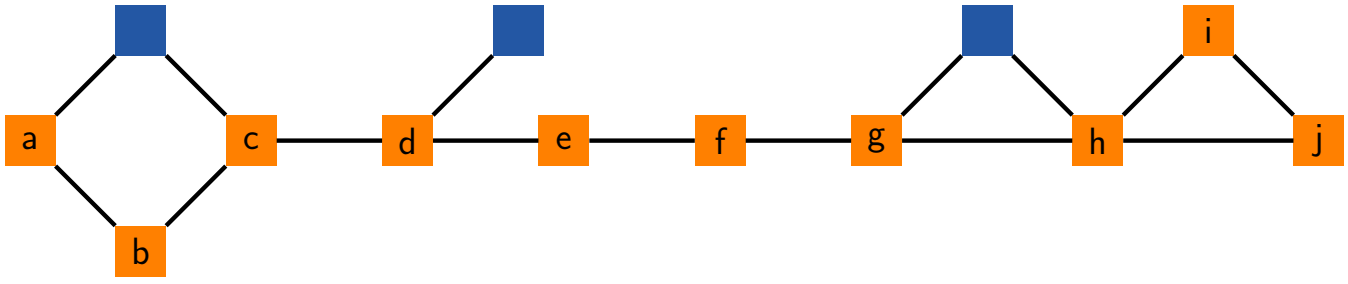
Goal configuration:



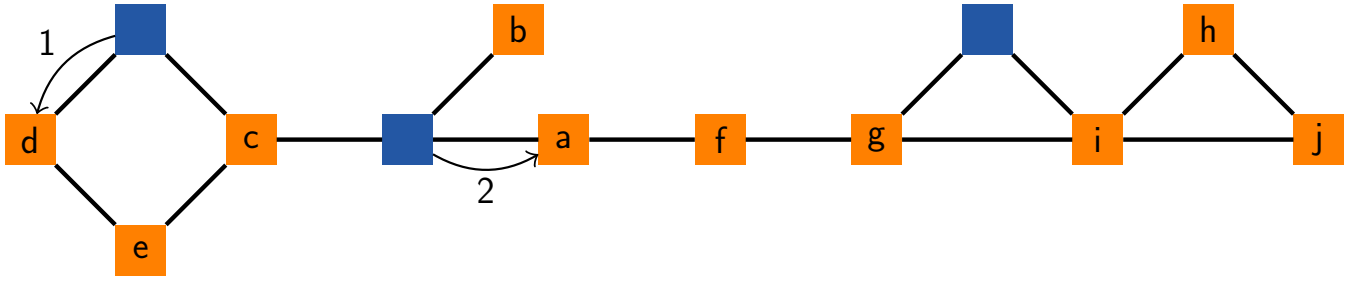
can produce $O(n^2)$ moves; goal configuration computable in $O(n)$

First step: fix the blank positions

Initial configuration:



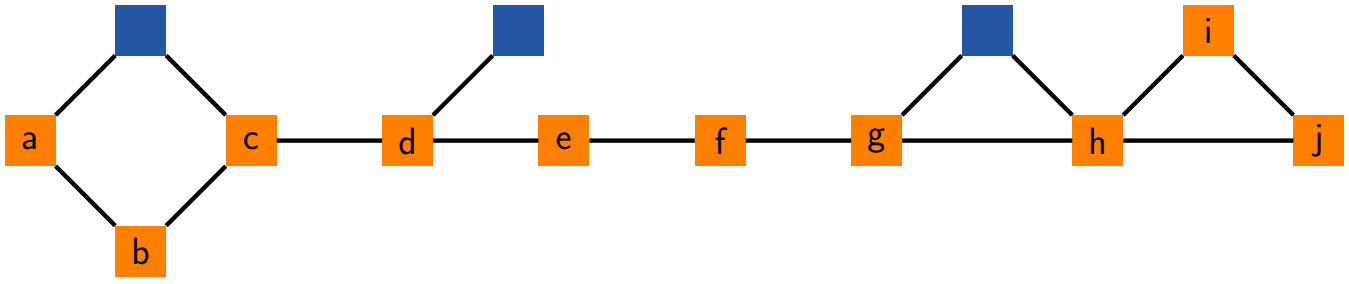
Goal configuration:



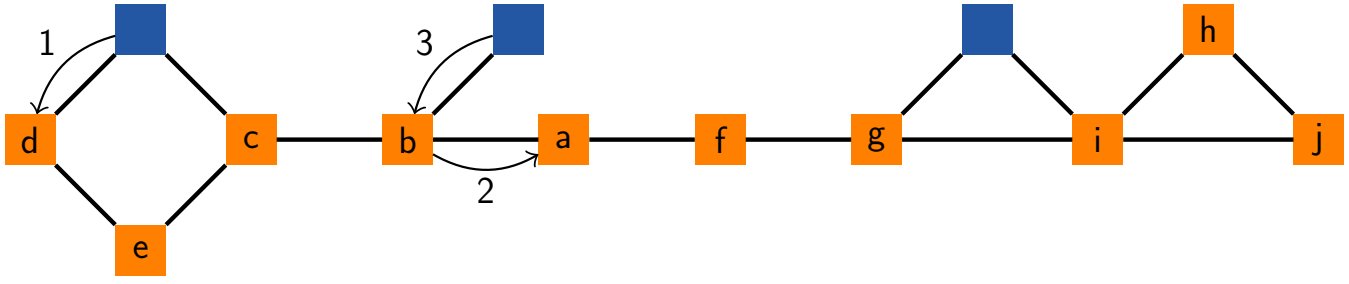
can produce $O(n^2)$ moves; goal configuration computable in $O(n)$

First step: fix the blank positions

Initial configuration:



New goal configuration:



can produce $O(n^2)$ moves; goal configuration computable in $O(n)$

Second step

Kornhauser et al.: second step

Overview:

1. Move blanks in goal configuration to initial blank positions. (This will form end of the solution.)
2. Derive subproblems, each defined by a set of agents that can reach the same set of nodes.
3. Solve each subproblem using permutation group theory.

Second step: derive the subproblems

Case of 1 blank:

- Compute nontrivial maximal biconnected components.
- Agents can never leave their biconnected component.
- Overall problem partitions into subproblems with one blank and biconnected graph ($\rightsquigarrow$ Wilson's case)

Second step: derive the subproblems

Case of 1 blank:

- Compute nontrivial maximal biconnected components.
- Agents can never leave their biconnected component.
- Overall problem partitions into subproblems with one blank and biconnected graph ($\rightsquigarrow$ Wilson's case)

$O(n)$ with standard graph algorithms (Hopcroft and Tarjan, 1973)

Second step: derive the subproblems

Case of 1 blank:

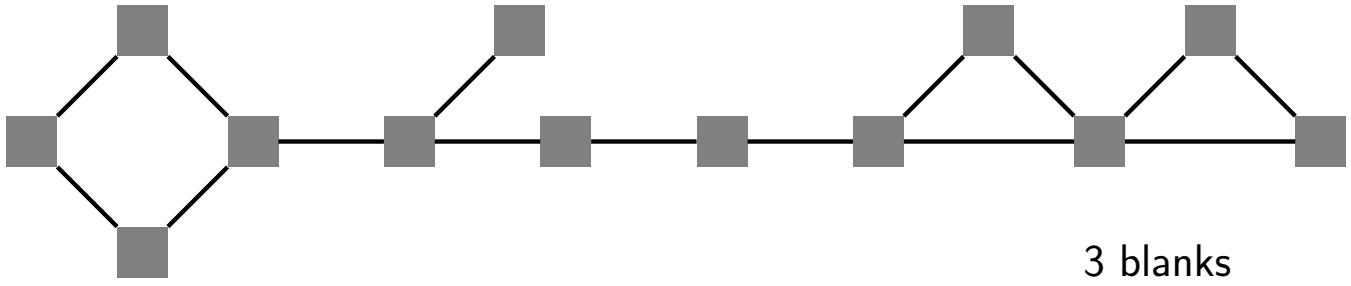
- Compute nontrivial maximal biconnected components.
- Agents can never leave their biconnected component.
- Overall problem partitions into subproblems with one blank and biconnected graph ($\rightsquigarrow$ Wilson's case)

$O(n)$ with standard graph algorithms (Hopcroft and Tarjan, 1973)

Case of ≥ 2 blanks:

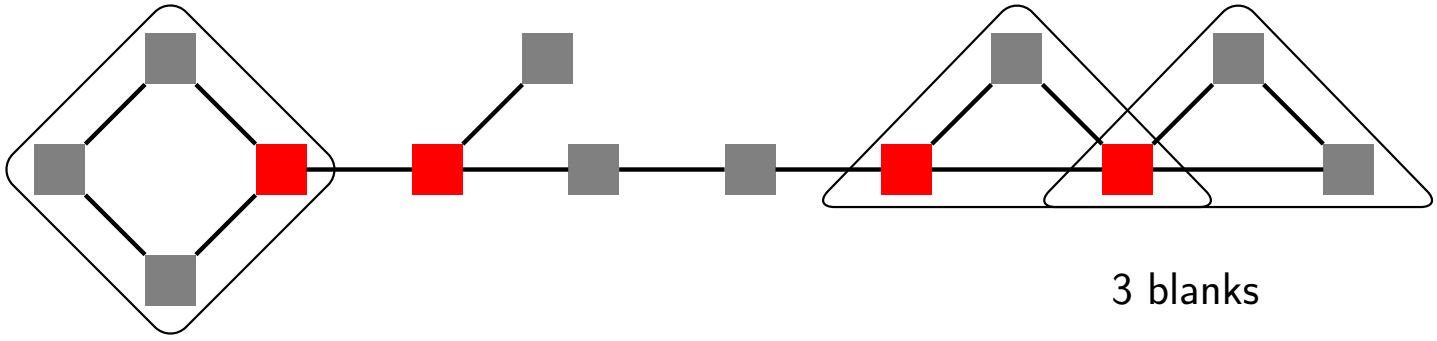
$\rightsquigarrow$ next slides

Second step: derive the subproblems for ≥ 2 blanks



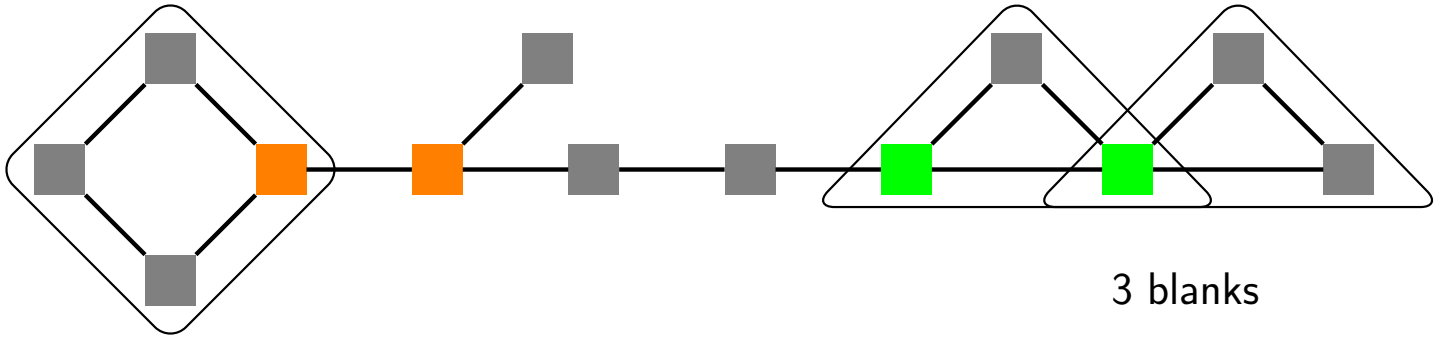
3 blanks

Second step: derive the subproblems for ≥ 2 blanks



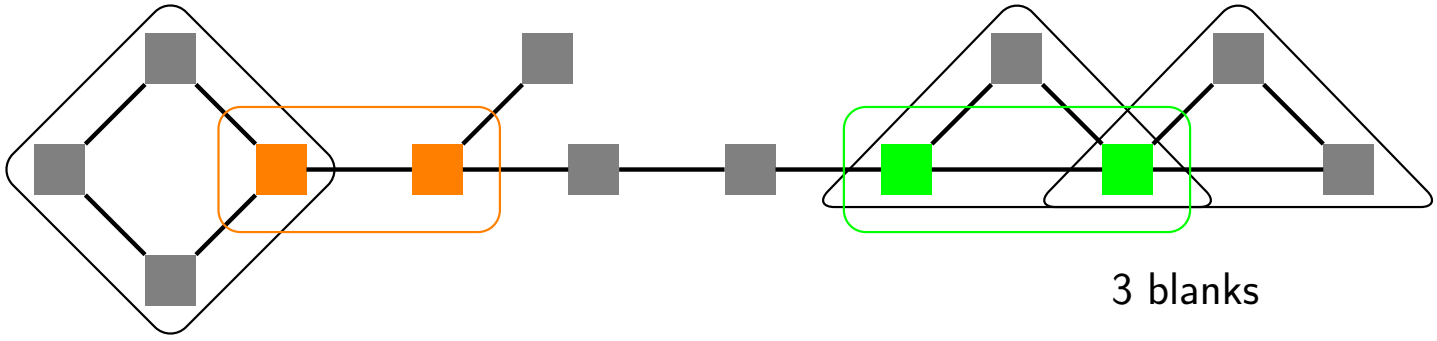
- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.

Second step: derive the subproblems for ≥ 2 blanks



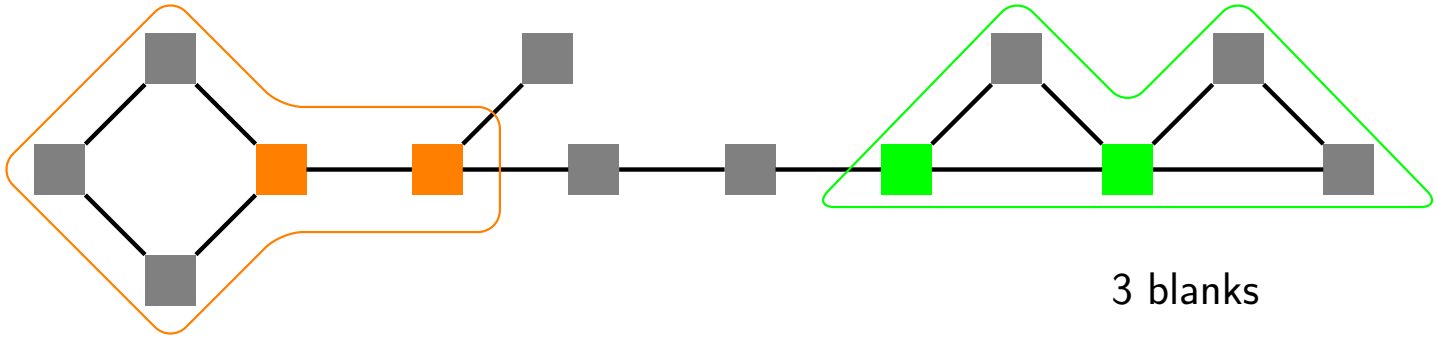
- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.
- Compute equivalence classes of articulation points.
  - equivalent if in same nontrivial component
  - equivalent if reachable on path of length $\leq \#blanks - 2$
  - build closure

Second step: derive the subproblems for ≥ 2 blanks



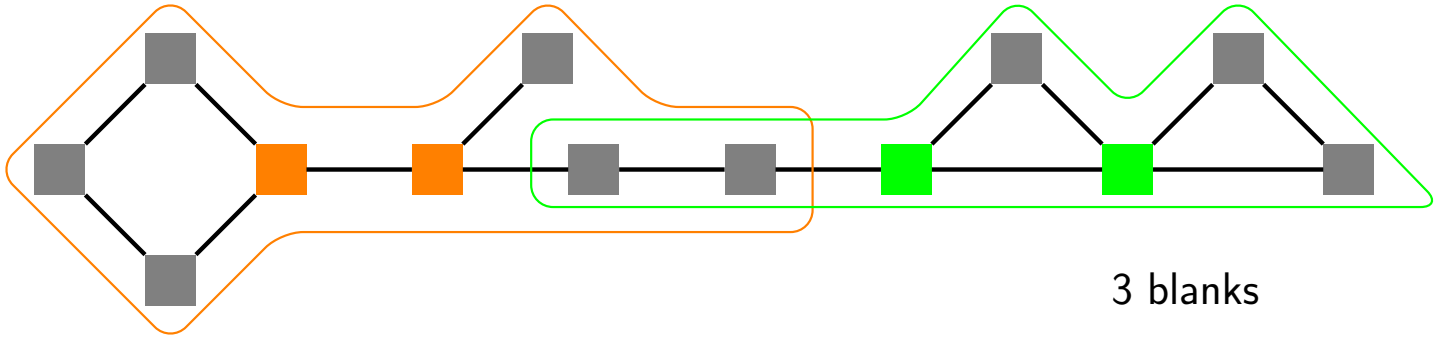
- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.
- Compute equivalence classes of articulation points.
  - equivalent if in same nontrivial component
  - equivalent if reachable on path of length $\leq \#blanks - 2$
  - build closure
- one subgraph for each equivalence class

Second step: derive the subproblems for ≥ 2 blanks



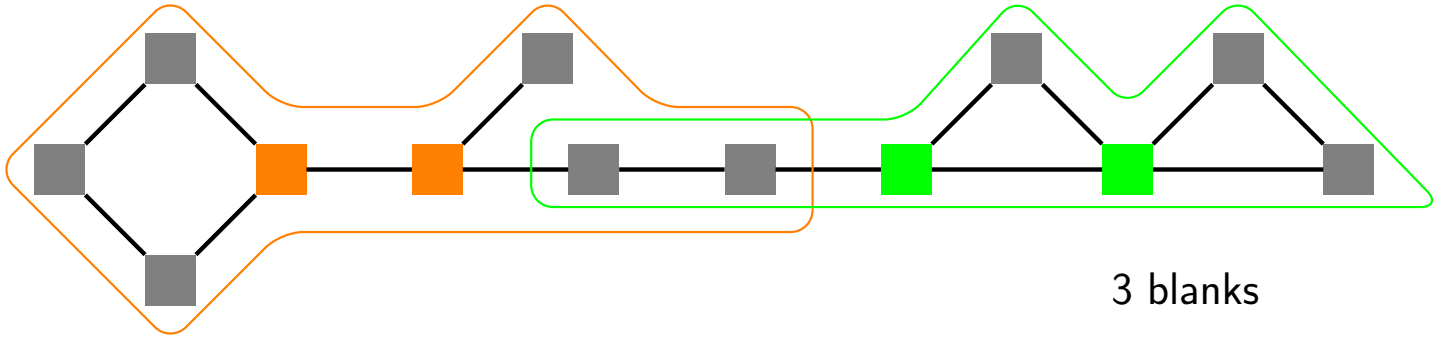
- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.
- Compute equivalence classes of articulation points.
  - equivalent if in same nontrivial component
  - equivalent if reachable on path of length $\leq \#blanks - 2$
  - build closure
- one subgraph for each equivalence class
  - extend by nontrivial biconnected components sharing a node

Second step: derive the subproblems for ≥ 2 blanks



- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.
- Compute equivalence classes of articulation points.
  - equivalent if in same nontrivial component
  - equivalent if reachable on path of length $\leq \#\mathit{blanks} - 2$
  - build closure
- one subgraph for each equivalence class
  - extend by nontrivial biconnected components sharing a node
  - extend by nodes reachable on path of length $\leq \#\mathit{blanks} - 1$

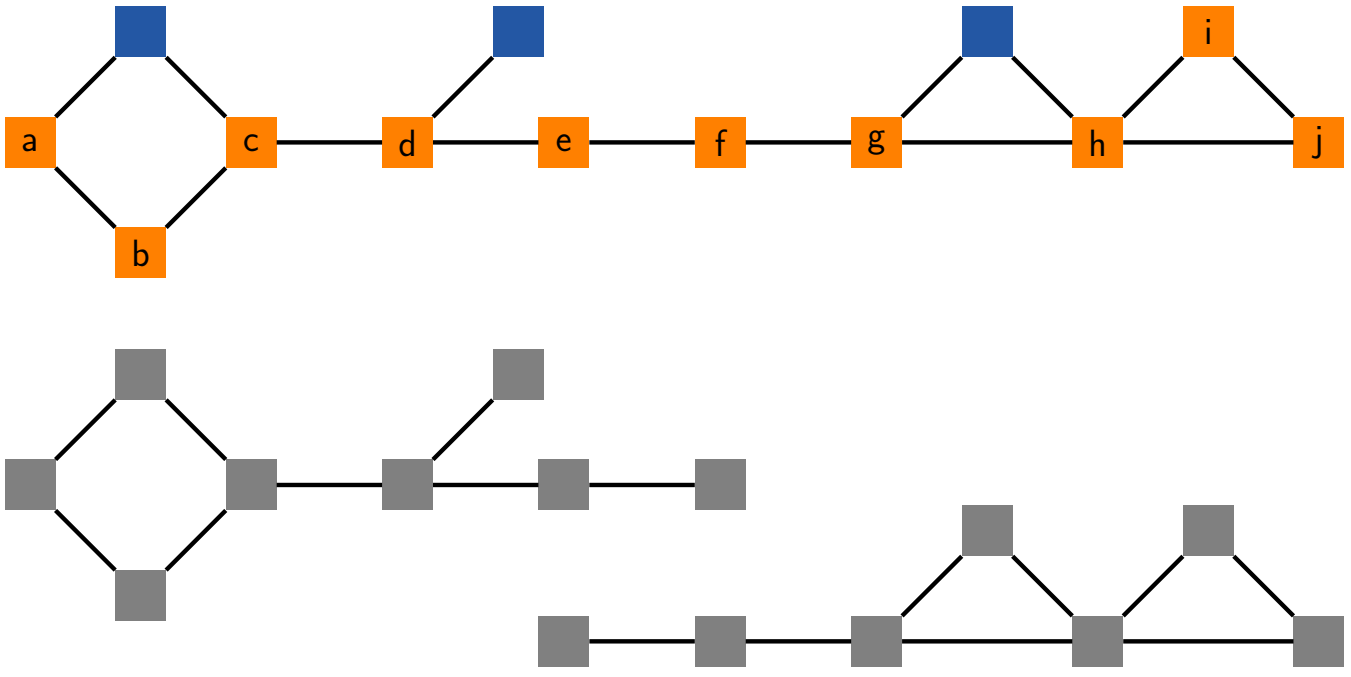
Second step: derive the subproblems for ≥ 2 blanks



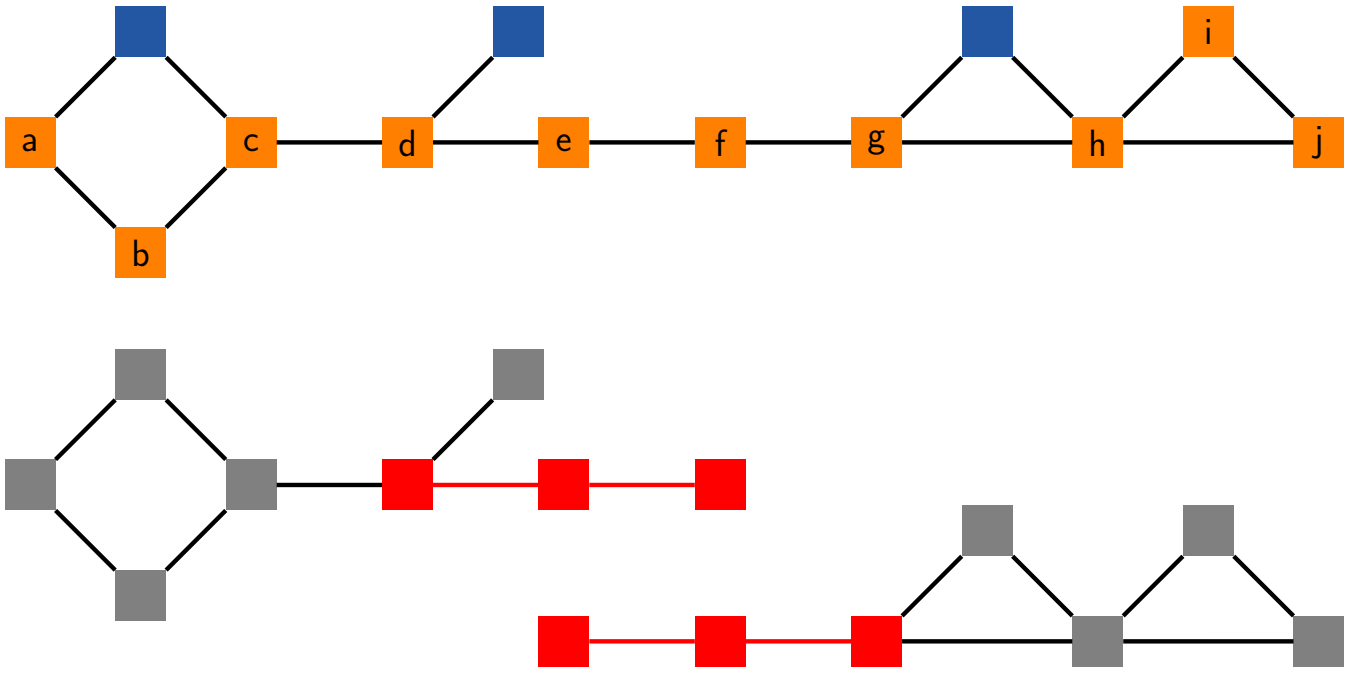
- Compute nontrivial maximal biconnected components and articulation points of degree ≥ 3.
- Compute equivalence classes of articulation points.
  - equivalent if in same nontrivial component
  - equivalent if reachable on path of length $\leq \#\mathit{blanks} - 2$
  - build closure
- one subgraph for each equivalence class
  - extend by nontrivial biconnected components sharing a node
  - extend by nodes reachable on path of length $\leq \#\mathit{blanks} - 1$

implementable in $O(n)$

Second step: assign agents to subproblems

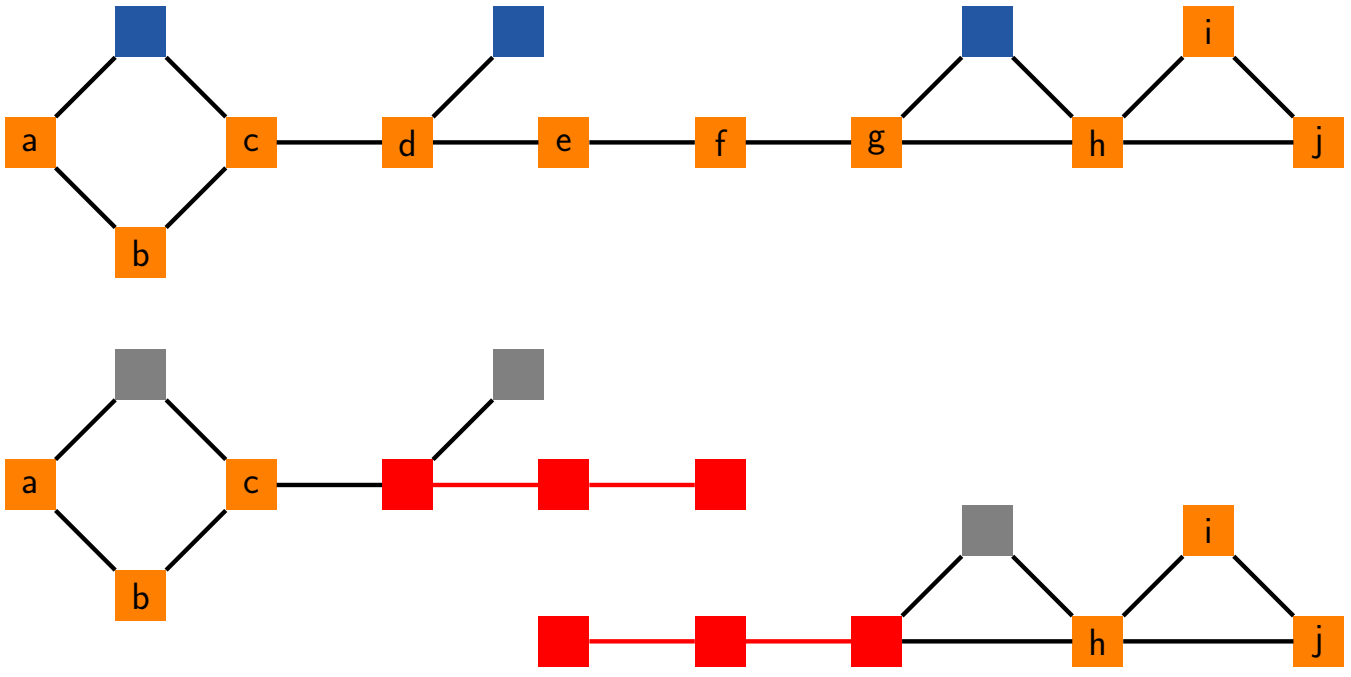


Second step: assign agents to subproblems



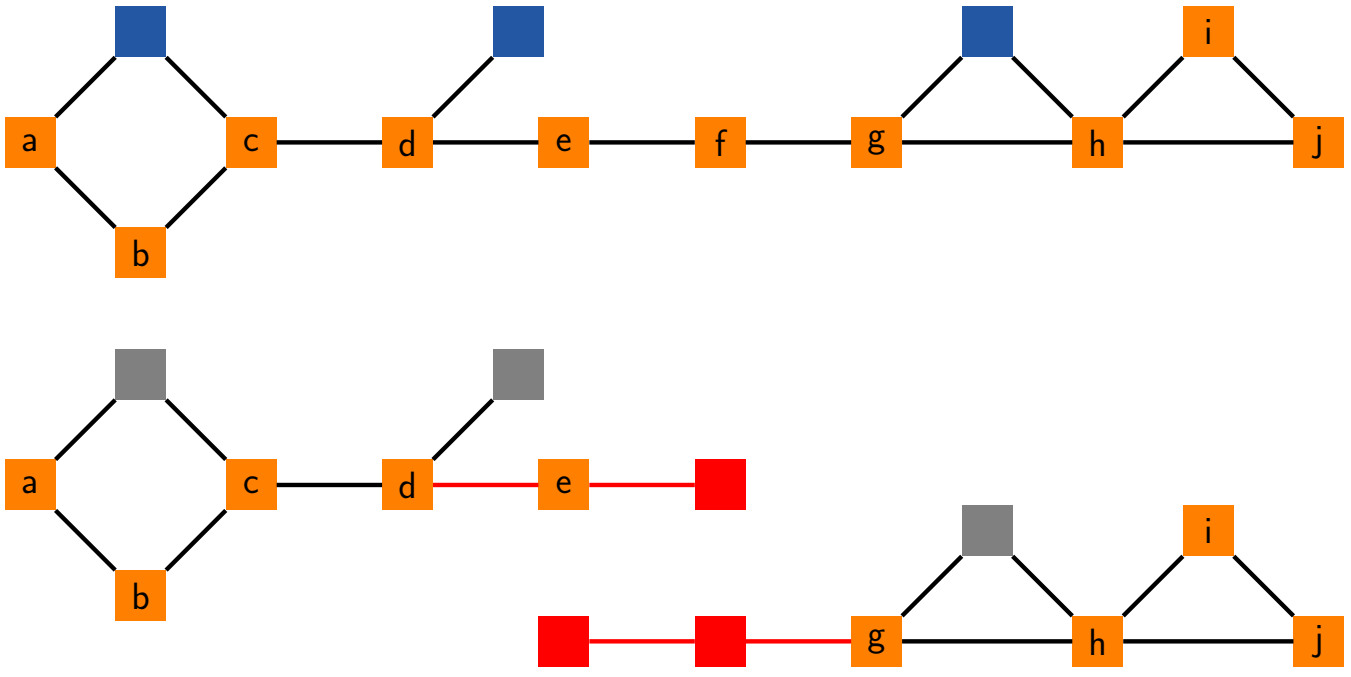
Subgraphs connect to nodes outside the subgraph
via lines of length $\#blanks - 1$: planks of the subgraph.

Second step: assign agents to subproblems



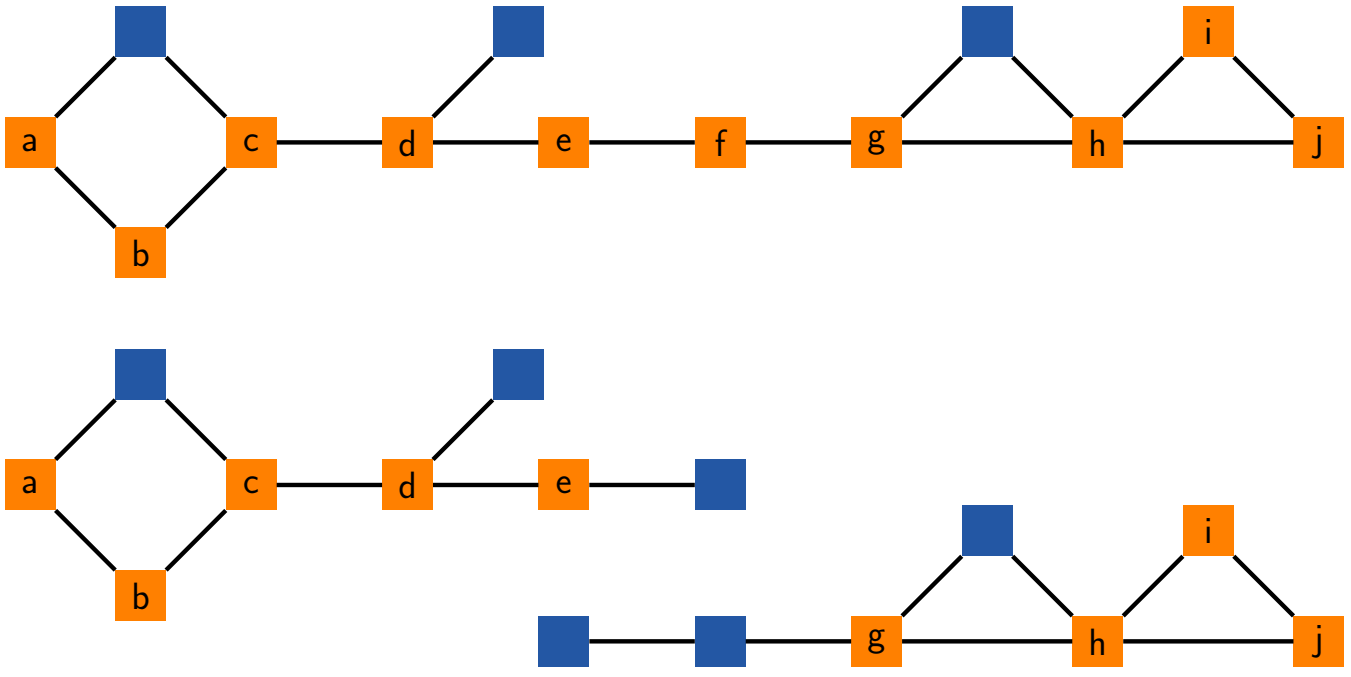
Assign all agents on non-plank nodes.

Second step: assign agents to subproblems



If an agent can leave a plank towards a subgraph, assign it to that subproblem (count blanks on that side of overall graph).

Second step: assign agents to subproblems



All other nodes of the subproblems are unoccupied.

$\rightsquigarrow$ assignment possible in $O(n)$

Second step: solvability

The overall problem is solvable iff

- each subproblem contains same agents in initial and goal state
- all agents not assigned to a subproblem are initially on same node as in the goal, and
- all subproblems are solvable $\rightsquigarrow$ next step

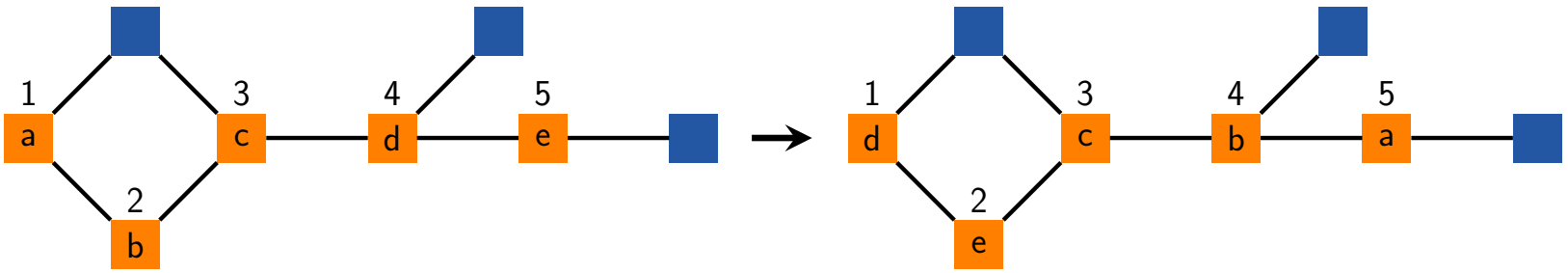
Third step

Kornhauser et al.: third step

Overview:

1. Move blanks in goal configuration to initial blank positions. (This will form end of the solution.)
2. Derive subproblems, each defined by a set of agents that can reach the same set of nodes.
3. Solve each subproblem using permutation group theory.

Permutation of a (normalized) MAPF problem

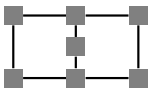


Permutation on occupied nodes:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 1 & 2 \end{pmatrix}$$

Solvability of the subproblems

1 blank (Wilson): solvable iff one of three cases holds

- graph is polygon: agents initially in same order as in goal
- graph is  : decide using precomputed look-up table
- other graph: graph is not bipartite or permutation is even

≥ 2 blanks: solvable iff one of two cases holds

- graph is polygon: agents initially in same order as in goal
- graph is not a polygon: always

All conditions testable in $O(n)$.

Some permutation group theory (1)

- k-cycle $(l_0 \ldots l_{k-1})$: permutation mapping l_i to $l_{(i+1) \bmod k}$, leaving everything else stationary
- transposition, swap: 2-cycle
- every permutation on n elements expressible as product of at most $n-1$ swaps

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 1 & 2 \end{pmatrix} = (1524) = (15)(12)(14)$$

$\rightsquigarrow$ How do we get all swaps?

Some permutation group theory (2)

- permutation even (odd) if it is the product of an even (odd) number of swaps
- every even permutation can be expressed as product of at most $n-2$ 3-cycles
- every odd permutation can be expressed as product of at most $n-2$ 3-cycles and any fixed odd permutation

$\rightsquigarrow$ How do we get all 3-cycles?

Some permutation group theory (3)

k-transitivity (in MAPF terminology):

- any k agents can be moved to any k-tuple of target locations
- it is OK to disturb all other agents arbitrarily while doing this

Some permutation group theory (4)

- Key fact: If we know one k-cycle c and have k-transitivity, we can generate all k-cycles.
- Idea:
  1. Move the agents we want to cycle to the position of c.
  2. Perform the cycle c.
  3. Repeat movements from part 1. in reverse.
- Formally: Given cycles $c = (l_1, \ldots, l_k)$ and $c' = (l'_1, \ldots, l'_k)$, find permutation t that maps each l_i to l'_i.
  Then $c' = t^{-1}ct$.

Solving the subproblems

1. Express permutation as product $c_1 \dots c_n$ of k-cycles.
2. Derive move sequence m for one arbitrary k-cycle.
3. For each c_i find move sequence m_i for a suitable k-transitivity permutation.
4. Then $m_1^{-1} m m_1 \dots m_n^{-1} m m_n$ solves the subproblem.

1 blank: use 3-cycle and 3-transitivity

≥ 2 blanks: use 2-cycle (transposition) and 2-transitivity

Solving the subproblems

1. Express permutation as product $c_1 \dots c_n$ of k-cycles.
2. Derive move sequence m for one arbitrary k-cycle.
3. For each c_i find move sequence m_i for a suitable k-transitivity permutation.
4. Then $m_1^{-1} m m_1 \dots m_n^{-1} m m_n$ solves the subproblem.

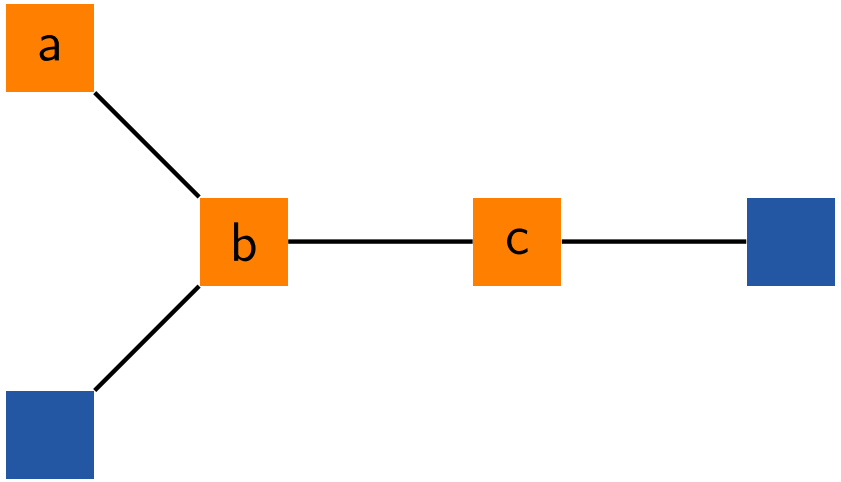
1 blank: use 3-cycle and 3-transitivity

≥ 2 blanks: use 2-cycle (transposition) and 2-transitivity

$\rightsquigarrow$ next slide (other cases omitted)

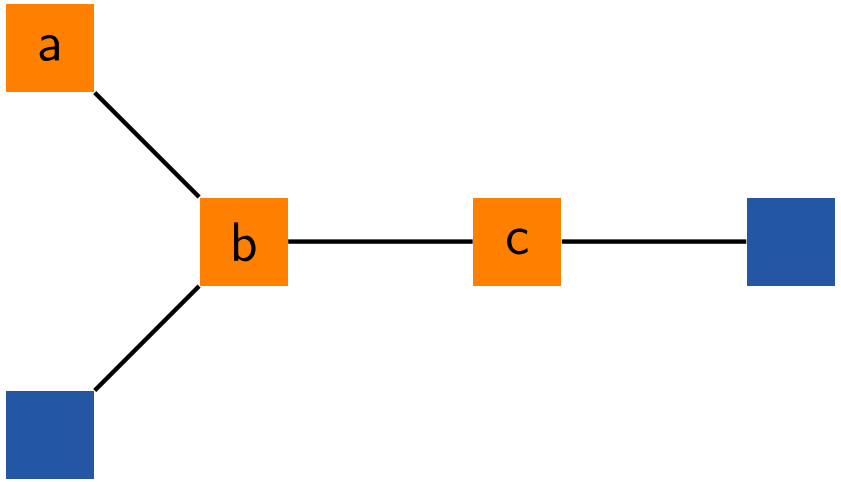
Finding one transposition (≥ 2 blanks)

can swap at any node of degree ≥ 3:



Finding one transposition (≥ 2 blanks)

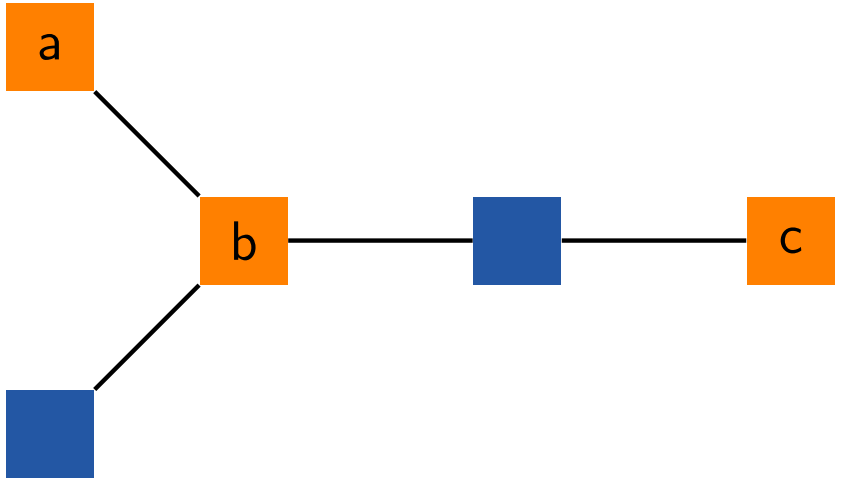
can swap at any node of degree ≥ 3:



Move two blanks next to node of degree ≥ 3.

Finding one transposition (≥ 2 blanks)

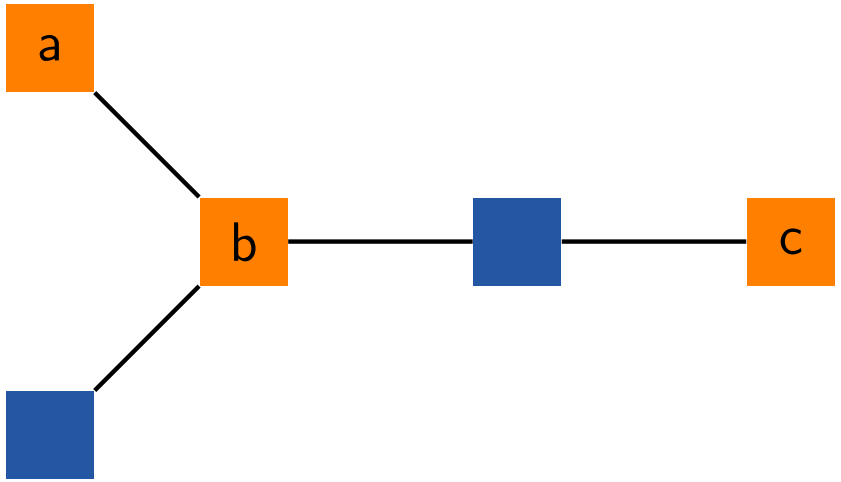
can swap at any node of degree ≥ 3:



Move two blanks next to node of degree ≥ 3.

Finding one transposition (≥ 2 blanks)

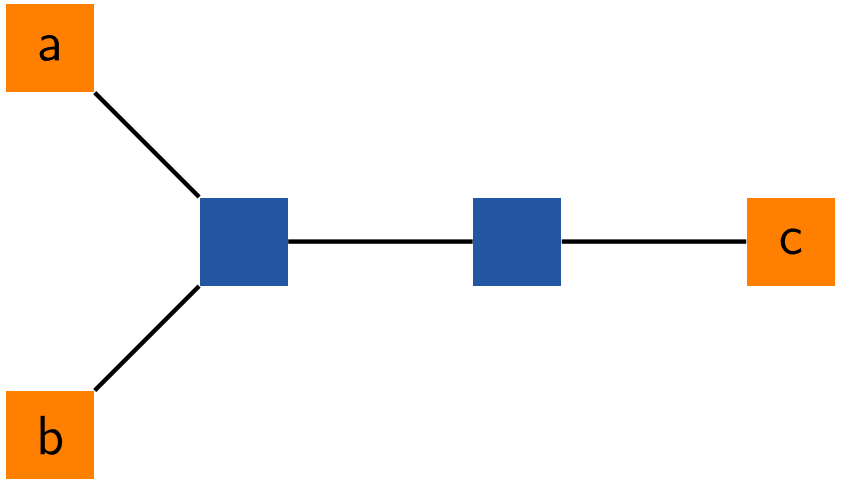
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

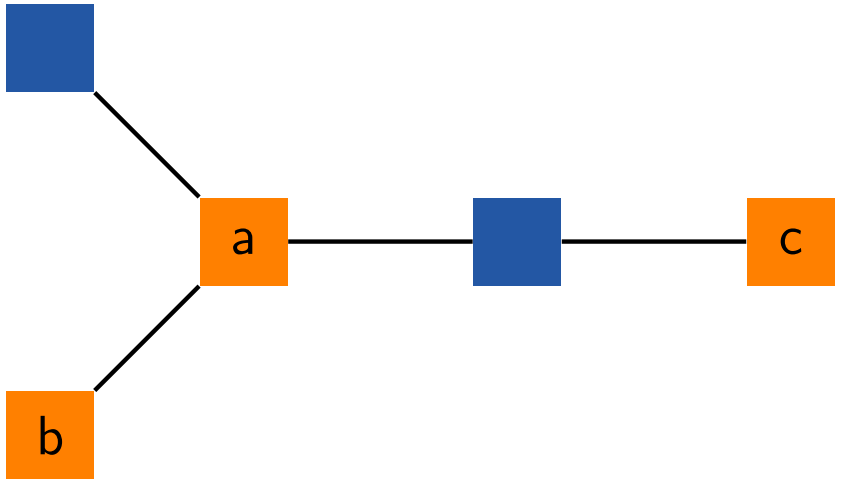
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

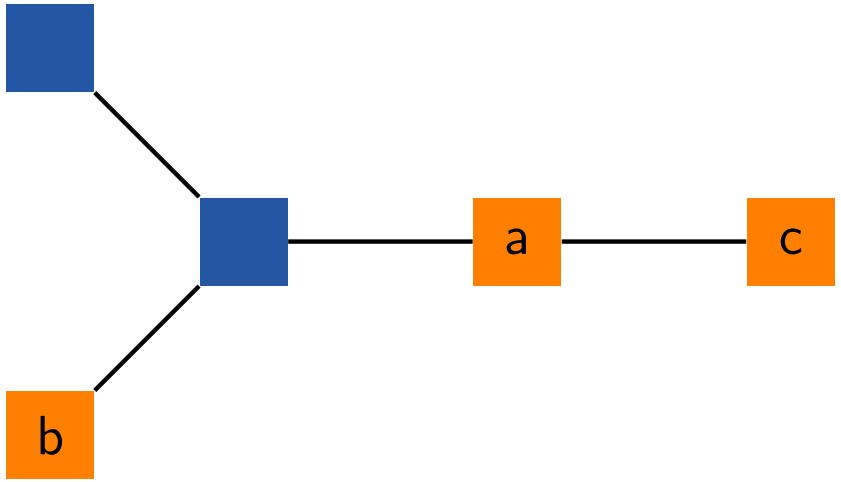
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

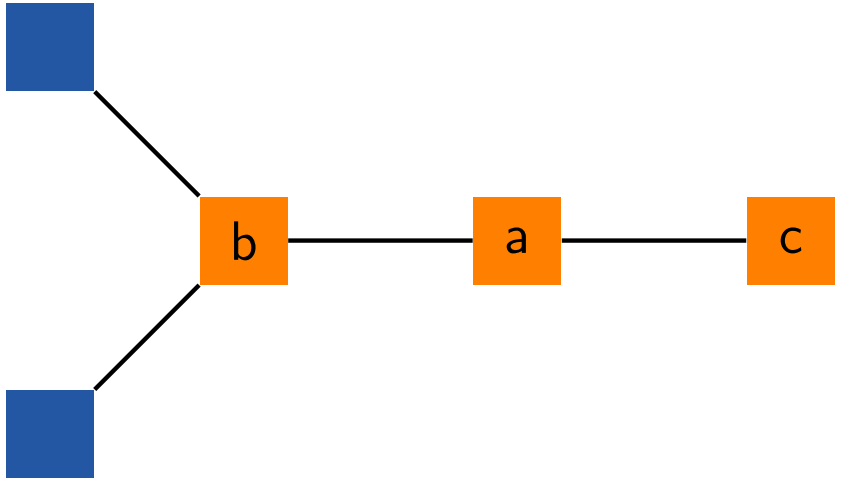
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

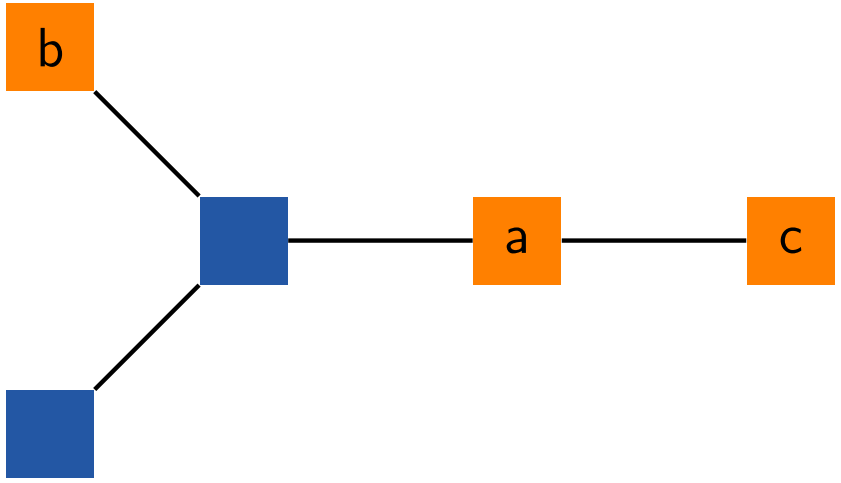
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

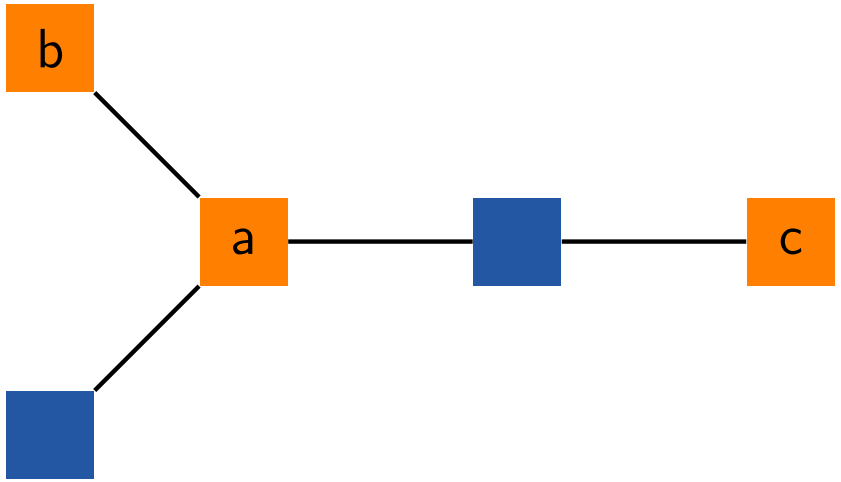
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

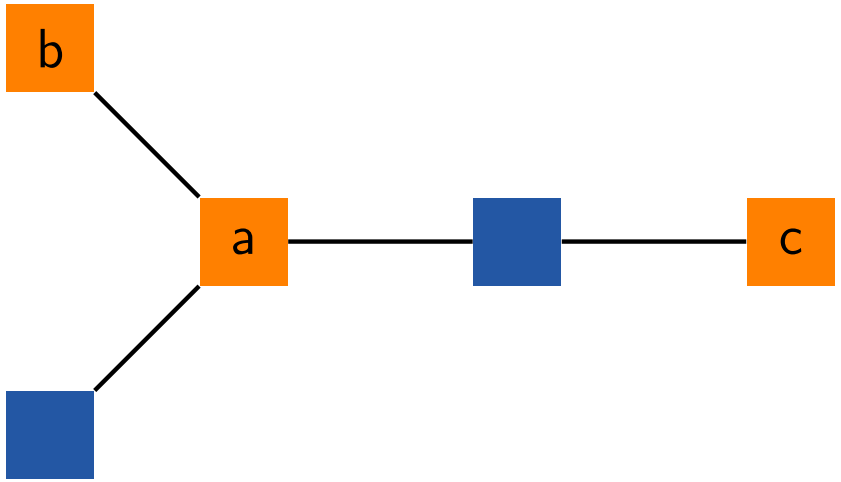
can swap at any node of degree ≥ 3:



Swap two agents which are closest to the center.

Finding one transposition (≥ 2 blanks)

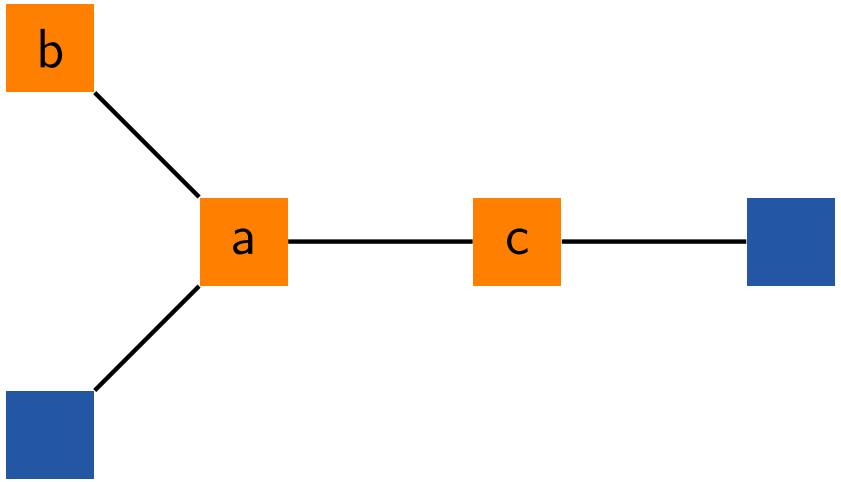
can swap at any node of degree ≥ 3:



Move blanks back to initial position.

Finding one transposition (≥ 2 blanks)

can swap at any node of degree ≥ 3:



Move blanks back to initial position.

Conclusion

Here:

- We have sketched major ideas and ingredients of the Kornhauser et al. algorithm.
- Our aim: facilitate understanding of work in combinatorial theory community, help absorbing it into our community.

Ongoing and future work:

- complete pseudo-code description
- implementation
- experimental comparison to other approaches
- instantiation/improvements to give reasonable solution lengths

The end

Thank you for your attention!