Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Solving Multi-Agent Path Finding on Strongly Biconnected Digraphs* (Extended Abstract)

Adi Botea,[1] **Davide Bonusi**[2] and **Pavel Surynek**[3]
[1] IBM Research, Dublin, Ireland
[2] Raffmetal S.p.a, Brescia, Italy
[3] Czech Technical University, Faculty of Information Technology
adibotea@ie.ibm.com, davidebonusi@gmail.com, pavel.surynek@fit.cvut.cz

Abstract

We present and evaluate diBOX, an algorithm for multi-agent path finding on strongly biconnected directed graphs. diBOX runs in polynomial time, computes suboptimal solutions and is complete for instances on strongly biconnected digraphs with at least two unoccupied positions. A detailed empirical analysis shows a good scalability for diBOX.

1 Introduction

In multi-agent path finding (MAPF), a group of agents have to navigate from their original locations to their target locations. This is a challenging computational problem, as agents need to avoid collisions and deadlocks. We focus on *cooperative path finding* [Silver, 2005], where each agent is interested in reaching its target, as opposed to an adversarial framework, where agents could block each other on purpose. The navigation environment is often represented as a grid map or, more generally, as an undirected graph [Ryan, 2008].

MAPF has many applications in areas such as robotics, traffic and computer games. Work on multi-agent path finding spans across several computer science fields, such as artificial intelligence, robotics and graph theory.

Suboptimal rule-based algorithms achieve a good scalability. However, such approaches make the assumption that the graph representing the navigation environment is *undirected*. Consider, for example, the swap primitive implemented in algorithms such as Push and Swap [Luna and Bekris, 2011] and Push and Rotate [de Wilde *et al.*, 2013]. This is a macro-action that swaps the positions of two agents. As part of the process, an agent moves to an adjacent vertex, to allow another agent to pass through. Then the first agent comes back to its former location, traversing the corresponding edge in the opposite direction. The MAPP algorithm [Wang and Botea, 2011] often requires reverting part of recent moves, after an agent has reached its target.

Using edges in both directions is also employed in algorithms such as TASS [Khorshid *et al.*, 2011] and BIBOX [Surynek, 2009]. In the latter, taking an agent out of a cycle involves the following steps: rotate the agents in the cycle until the corresponding agent can step out at the desired location; and rotate the remaining agents back in the opposite direction.

Many real-life scenarios however can be better modeled using directed edges that can be traversed in one direction only. For example, some environments may exhibit features such as one-way entrances, exits, escalators, bridges and roads. Agents capable to travel down the hill may not necessarily be able to climb up. Furthermore, algorithms such as FAR impose unidirectional traffic on purpose, to avoid head-to-head collisions [Wang and Botea, 2008]. Motion in directed graphs is also important in asymmetric communication networks [Marina and Das, 2002; Jetcheva and Johnson, 2006; Wu and Grumbach, 2010].

We focus on multi-agent path finding on strongly biconnected directed graphs (digraphs). These are strongly connected digraphs where the undirected graphs obtained by ignoring the edge orientations have no cut vertices. We found that all instances on strongly biconnected digraphs with at least two unoccupied vertices (blanks) can be solved suboptimally in polynomial time, except for the particular case of digraphs with a cyclic shape, where instances may or may not have a solution [Botea and Surynek, 2015; Botea *et al.*, 2018].

In this extended abstract, we overview and evaluate diBOX, a suboptimal algorithm that is complete on instances with at least two blanks on strongly biconnected digraphs. As mentioned later in this extended abstract, diBOX makes use of a decomposition of the input digraph into simpler structures called ears. In a detailed empirical analysis we show that the decomposition strategy plays an important role in the performance of the algorithm, and present a strategy that leads to an effective performance. We further show that diBOX scales convincingly beyond the capabilities of an existing optimal solver.

2 Related Work

Previous formal studies of multi-agent pathfinding appear to be focused on undirected graphs [Wilson, 1974; Kornhauser *et al.*, 1984]. For instance, Wilson [1974] explicitly requires that the adjacency relation between agent configura-

*This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Botea, Bonusi and Surynek, 2018]

tions (called "labellings") is symmetric, which is equivalent to stating that the graph is undirected. All graphs considered in these two works appear to have undirected edges, with no mention about if or how parts of the study, such as the considered permutation groups, would be applicable to directed graphs. Our work is complementary to previous work on undirected graphs. It is a step towards achieving a similar level of understanding for directed graphs.

Algorithms like BIBOX [Surynek, 2009], Push and Swap [Luna and Bekris, 2011], TASS [Khorshid *et al.*, 2011], and Push and Rotate [de Wilde *et al.*, 2014] build on above-mentioned formal studies and typically use small sets of movement rules to achieve the final configuration of agents. A key assumption in these rules is that the underlying graph is undirected.

Among existing rule-based, suboptimal, polynomial-time algorithms for undirected graphs, we view Surynek's [2009; 2014] algorithm BIBOX as the most related to our work. The main difference is that BIBOX explicitly relies on the fact that edges allow travel in both directions. Both BIBOX and diBOX employ an ear decomposition of the input graph, and solve the ears one by one in a sequence. However, the technical details related to solving individual ears are quite different in the cases of undirected graphs and directed graphs respectively.

Although rule-based algorithms are scalable and provide completeness guarantees, their solutions may suffer quality-wise. Therefore multiple search-based algorithms, both optimal and bounded suboptimal, have been developed. Examples include OD+ID [Standley, 2010], M* [Wagner and Choset, 2015], CBS [Sharon *et al.*, 2015], ICTS [Sharon *et al.*, 2013] and SAT compilation [Surynek *et al.*, 2016].

3 Background

In this section, we define the problem formally and introduce a few relevant graph theoretical concepts.

Definition 1 *Given a directed graph (digraph) $D = (V, E)$ and a set of agents A, a configuration of agents over D is a placement of agents in vertices of the graph, with at most one agent in each vertex. Formally, the configuration is a uniquely invertible assignment of agents to vertices $\alpha : A \longrightarrow V$.*

If we need to know what agent is placed in a given vertex we may use the inverse configuration $\alpha^{-1} : V \longrightarrow A \cup \{\text{blank}\}$ which is well defined due to the unique invertibility of α (the blank is used for unoccupied vertices).

A configuration can be transformed into another one using *moves*. A move involves changing the position of one agent to a neighboring vertex, provided that the target vertex is blank. Moves are possible only along the positive orientation of edges.

Definition 2 *An instance of multi-agent path finding over directed graphs consists of a digraph $D = (V, E)$, a set of agents A, an initial configuration $\alpha_0 : A \longrightarrow V$, and a goal configuration $\alpha_+ : A \longrightarrow V$. The task is to find a sequence of moves over D that transform α_0 to α_+.*

An undirected graph G is biconnected if G is connected and there are no cut vertices in G. In other words, removing any single vertex would keep the graph connected. A digraph D is strongly connected if, for any two distinct vertices v and w, both a directed path from v to w and a directed path from w to v exist in D. Given a digraph D, $\mathcal{G}(\mathcal{D})$ is the underlying undirected graph of D, i.e., the undirected graph obtained by ignoring the orientation of the edges.

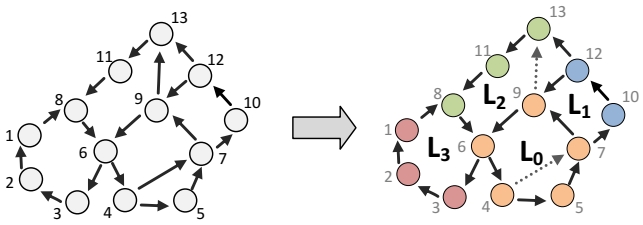

Figure 1: An example of a strongly biconnected digraph (left) and an ear decomposition (right). Dashed edges are trivial derived ears (not explicitly labelled to avoid clutter).

Definition 3 (Wu and Grumbach 2010) *Let D be a digraph. D is said to be strongly biconnected if D is strongly connected and $\mathcal{G}(\mathcal{D})$ is biconnected.*

Ear decompositions have been defined for both directed and undirected graphs. They allow decomposition of certain graphs into a cycle and a collection of chains called ears. Ear decompositions corresponding to undirected graphs have been used in BIBOX [Surynek, 2014]. We will make use of ear decompositions in strongly biconnected digraphs.

Definition 4 *An* ear decomposition *of a digraph $D = (V, E)$ is an ordered sequence of subgraphs of D, say $[L_0, L_1, \ldots, L_r]$, such that:*

- *L_0 is a cycle; and*
- *$\forall i \in \{1 \ldots r\}$, L_i is a path (chain) whose two endpoints belong to the subgraph*

$$D_{i-1} = \bigcup_{j=0}^{i-1} L_j,$$

but no other vertices or edges of L_i belong to D_{i-1}.

Figure 1 shows an example. Each subgraph L_i is called an ear. We say that L_0 is the *basic cycle* and all other ears are *derived ears*. In the example, L_0 contains the vertices $4, 5, 7, 9, 6$ and the corresponding edges $(4, 5)$, $(5, 7)$, $(7, 9)$, $(9, 6)$, $(6, 4)$.

An ear is *cyclic* if its endpoints are represented by a single vertex. Our example has no derived cyclic ears.

Given an ear decomposition $O = [L_0, L_1, \ldots, L_r]$, we call the *$i$-prefix decomposition* the decomposition O_i restricted to the first i ears: $O_i = [L_0, L_1, \ldots, L_i]$. Notice that this is a proper ear decomposition on its own, corresponding to a subgraph D_i of D, whose vertices and edges are precisely the vertices and edges contained in the first i ears. We call D_i the i-prefix subgraph.

In the example, O_1 contains the vertices $4, 5, 7, 9, 6, 10, 12$ and all the edges involved in L_0 and L_1: $(4, 5)$, $(5, 7)$, $(7, 9)$, $(9, 6)$, $(6, 4)$, $(7, 10)$, $(10, 12)$, and $(12, 9)$. Notice that the trivial ear $(4, 7)$ is not part of O_1.

Definition 5 *An* open ear decomposition *of a digraph D is an ear decomposition with no cyclic derived ears.*

The decomposition shown in Figure 1 is open, as each derived ear has its entrance vertex distinct from its exit vertex (i.e., no derived ear is cyclic).

Theorem 1 (Wu and Grumbach 2010) *Let D be a non-trivial digraph, i.e. containing at least two vertices. D is strongly biconnected if and only if D has an open ear decomposition. Moreover, any cycle can be the starting point of an open ear decomposition.*

Corollary 1 *Let D be a strongly biconnected digraph, and let $O = [L_0, L_1, \ldots, L_r]$ be an open ear decomposition. For every i with $0 \leq i \leq r$, the i-prefix subgraph D_i is strongly biconnected.*

4 The diBOX Algorithm

Before invoking diBOX on a given MAPF instance, we assume that when the underlying digraph is strongly biconnected, a regular open ear decomposition is available. These are properties that depend on the input digraph, but not on the initial and the goal positions of the agents. Thus, preprocessing can be performed once and re-used in solving many MAPF instances on a given digraph.

Algorithm 1: diBOX in pseudocode.

Input: a digraph $D = (V, E)$; a set of agents A; an initial configuration α_0 of agents over D; a goal configuration α_+ of agents over D
Output: sequence of moves transforming α_0 into α_+

```
1 diBOX (D, A, α_0, α_+)
2 let [L_0, L_1, ..., L_r] be a regular open ear decomp.
    with non-trivial ear L_1 attached to L_0;
3 α ← α_0;
4 for i = r down to 1 do
5 |   SolveDerivedEar (D, L_i, α_+);
6 SolveBasicCycle (L_0, L_1, α_+);
```

Algorithm 1 shows diBOX in pseudocode, simplified (to save room) as follows. Firstly, we assume that, in the goal configuration, the basic cycle has at least two blanks. This assumption is safe to make, with no loss of generality. Secondly, we skip a particular, degenerate case when the input digraph has a cyclic shape. See the long article for details [Botea *et al.*, 2018].

Ears are solved in reverse order, leaving the basic cycle to the end. In Figure 1, ears will be solved in the order L_3, L_2, L_1, L_0. After solving an ear L_r, with $r \geq 2$, we never have to touch that ear again. According to Corollary 1, after solving ear L_r, we are left with a problem of the same type, only strictly smaller. The process continues recursively, until we need to solve the basic cycle.

Intuitively, solving an ear means to bring all corresponding agents to goal vertices located inside that ear. For instance, for ear L_3 in Figure 1, push inside the agent whose goal is vertex 1, followed by agent whose goal is vertex 2, followed by the agent whose goal is vertex 3. Agents are brought one by one to the entrance of the ear at hand, and then pushed inside the ear. Solving the basic cycle L_0 makes use of the ear L_1, whose both ends are attached to L_0. L_1's goal configuration is temporarily destroyed and restored in the process. Detailed movement rules, as well as correctness and complexity proofs can be found in the long article [Botea *et al.*, 2018]. Both the worst-case time complexity and the number of moves for diBOX are within $\mathcal{O}(|V|^3)$.

Digraph size (# vertices)	80		100		120	
Type of ear decomp.	S	L	S	L	S	L
Number of ears	19	12	20	13	22	15
Max ear size	11	32	17	38	22	49
Min ear size	3	3	3	3	3	3
Digraph size (# vertices)	160		180		200	
Type of ear decomp.	S	L	S	L	S	L
Number of ears	26	-	31	-	32	-
Max ear size	26	-	29	-	31	-
Min ear size	3	-	3	-	3	-

Table 1: Average key statistics on the digraph decomposition strategy. S = short-ear version; L = long-ear version.

5 Experimental Evaluation

We focus our analysis onto the scalability in densely populated instances, and a comparison to an optimal solver.

5.1 Scalability in Densely Populated Instances

We evaluate the diBOX algorithm on difficult instances with only two blanks available in the input digraph. Our digraphs range in size from 80 to 200 vertices. For each digraph size, we generate three digraphs, and for each digraph we generate 10 MAPF instances with the initial and the goal configurations set at random. In all cases, there are only two empty vertices (blanks), and all other vertices are occupied by one agent each. Such instances are too hard for optimal solvers.

Our evaluation presented in this section focuses on two major aspects: how the algorithm scales as the digraph size increases, and determining the impact of open ear decomposition on the performance. For the latter purpose, we have developed two decomposition strategies, one that aims at decompositions with long ears (version L), and one that aims at decompositions with short ears (version S) [Botea *et al.*, 2018].

Table 1 shows average key statistics about the digraph decomposition, such as the number of ears, the number of vertices of the biggest ear and the number of vertices of the smallest ear.

Version L is significantly more costly as compared to version S. In fact, version L does not scale beyond digraphs with 120 vertices, given a timeout of 30 minutes per instance.

Figure 2 and Figure 3 show average solution length data and average CPU time data, respectively. Version S performs better, both in terms of solution quality and CPU time. It generates significantly fewer moves than version L. The explanation stems from the fact that, in diBOX, agents move

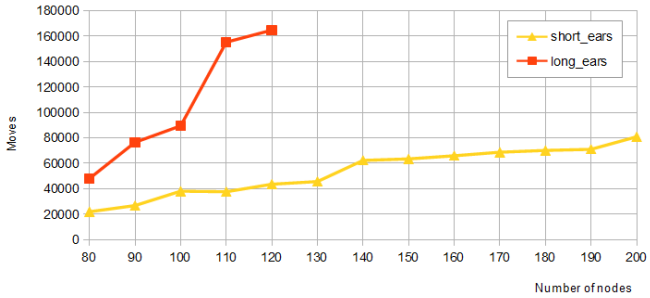

Figure 2: Average number of moves per instance.

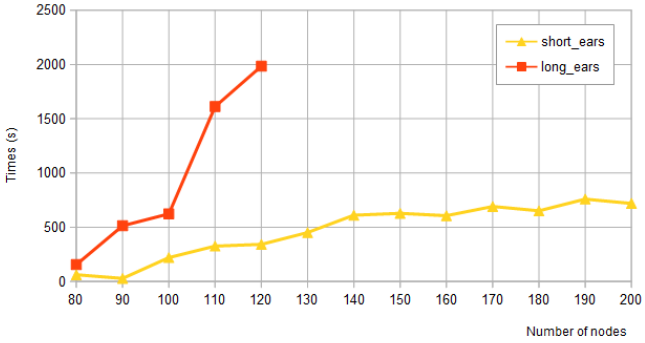

Figure 3: Average CPU time per instance.

only along ears [Botea *et al.*, 2018]. A digraph with shorter ears gives more opportunities to find shorter paths between a given pair of vertices.

For the version S, all figures show a good scalability of the algorithm in practice. Specifically, as the digraph size increases, the increase in the solving time and the solution length is much slower than the cubic upper bounds shown in our worst-case formal analysis.

Solving the Derived Ears and the Basic Cycle. In addition to the overall performance measurements presented earlier, we evaluated the performance broken down into solving the derived ears and solving the basic cycle. Figures 4 and 5 show the number of moves produced when solving derived ears and the basic cycle respectively. The charts demonstrate that using short ears is important when solving both types of ears.

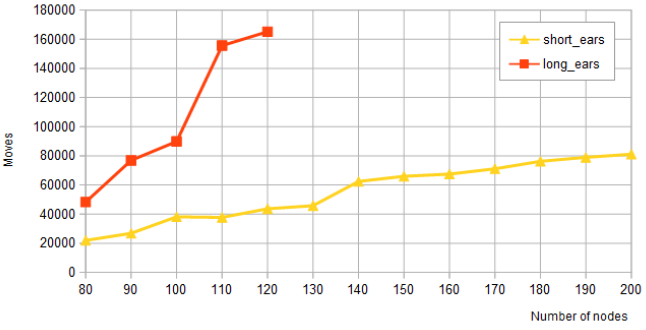

Figure 4: Average number of moves per instance to solve all derived ears, but not the basic cycle.

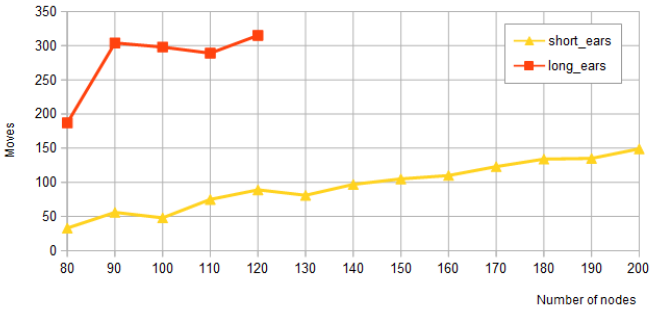

Figure 5: Average number of moves per instance to solve the basic cycle.

5.2 Comparison to Optimal Solutions

We tested diBOX against MDD-SAT [Surynek *et al.*, 2016], an existing optimal solver based on propositional satisfiability (SAT). Originally, MDD-SAT produces sum-of-costs optimal solutions, counting each move and each waiting action at a position different from the goal of the agent at hand. In our evaluation, MDD-SAT has been modified to generate optimal solutions in terms of the number of moves.

We report results using a digraph with 20 vertices. Instances are relatively small, so that an optimal solver could succeed at least in part of the instances. The basic cycle has 5 vertices. Derived ears vary in size from 2 to 5 internal vertices. The number of agents was gradually increased. For each number of agents we generated 10 instances with the initial and the goal configurations set at random.

On instances with 11 agents or more, MDD-SAT reached a timeout of 300 seconds. On the other hand, diBOX scaled much better, solving all instances in less than 1 second per instance. Solutions computed with diBOX range from nearly optimal to a deviation from optimal values by at most a factor of 4.5.

6 Conclusion

In this extended abstract we have focused on multi-agent path finding on strongly biconnected digraphs. We have presented a suboptimal rule-based algorithm, called diBOX, for solving instances with at least two blanks. Its worst-case time complexity and solution length are both within $\mathcal{O}(|V|^3)$, where V is the set of vertices.

We implemented the diBOX algorithm and evaluated its performance. Experiments show that ear decompositions have a great impact on the performance of the algorithm. One ear decomposition strategy aims at computing short ears and the other prefers long ones. Results clearly indicate that the preference of short ears is the better option, as it leads to better solutions in term of number of moves and CPU times. Overall, the results show a good scalability of the diBOX algorithm, in combination with short ear decompositions, successfully solving instances of an increasing size and difficulty. diBOX scales significantly better than an optimal SAT-based approach.

In future work, we plan to extend diBOX to a more general class of directed graphs.

References

[Botea and Surynek, 2015] Adi Botea and Pavel Surynek. Multi-agent path finding on strongly biconnected digraphs. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2024–2030. AAAI Press, 2015.

[Botea *et al.*, 2018] Adi Botea, Davide Bonusi, and Pavel Surynek. Solving multi-agent path finding on strongly biconnected digraphs. AAAI Press, in press, 2018.

[de Wilde *et al.*, 2013] Boris de Wilde, Adriaan W. ter Mors, and Cees Witteveen. Push and rotate: Cooperative multi-agent path planning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS-13)*, pages 87–94. IFAAMAS, 2013.

[de Wilde *et al.*, 2014] Boris de Wilde, Adriaan ter Mors, and Cees Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51:443–492, 2014.

[Jetcheva and Johnson, 2006] Jorjeta G. Jetcheva and David B. Johnson. Routing characteristics of ad hoc networks with unidirectional links. *Ad Hoc Networks*, 4(3):303–325, 2006.

[Khorshid *et al.*, 2011] Mokhtar M. Khorshid, Robert C. Holte, and Nathan R. Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Proceedings of the 4th Annual Symposium on Combinatorial Search, SOCS 2011*. AAAI Press, 2011.

[Kornhauser *et al.*, 1984] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–250. IEEE Computer Society, 1984.

[Luna and Bekris, 2011] Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*, pages 294–300. IJCAI/AAAI, 2011.

[Marina and Das, 2002] Mahesh K. Marina and Samir R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *Proceedings of ACM MobiHoc*, pages 12–23. ACM Press, 2002.

[Ryan, 2008] M. R. K. Ryan. Exploiting Subgraph Structure in Multi-Robot Path Planning. *Journal of Artificial Intelligence Research*, 31:497–542, 2008.

[Sharon *et al.*, 2013] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005*, pages 117–122. AAAI Press, 2005.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

[Surynek *et al.*, 2016] Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, pages 810–818. IOS Press, 2016.

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

[Surynek, 2014] Pavel Surynek. Solving abstract cooperative path-finding in densely populated environments. *Computational Intelligence*, 30(2):402–450, 2014.

[Wagner and Choset, 2015] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[Wang and Botea, 2008] K.-H. C. Wang and A. Botea. Fast and Memory-Efficient Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-08)*, pages 380–387. AAAI Press, 2008.

[Wang and Botea, 2011] K.-H. C. Wang and A. Botea. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[Wilson, 1974] Richard M Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974.

[Wu and Grumbach, 2010] Zhilin Wu and Stéphane Grumbach. Feasibility of motion planning on acyclic and strongly connected directed graphs. *Discrete Applied Mathematics*, 158(9):1017 – 1028, 2010.