Multi-Agent Path Finding on Strongly Biconnected Digraphs

Adi Botea
IBM Research
Dublin, Ireland
adibotea@ie.ibm.com

Pavel Surynek
Faculty of Mathematics and Physics
Charles University Prague
Malostranské náměstí 25, 11 800, Praha 1
pavel.surynek@mff.cuni.cz

Abstract

Much of the literature on multi-agent path finding focuses on undirected graphs, where motion is permitted in both directions along a graph edge. Despite this, travelling on directed graphs is relevant in navigation domains, such as pathfinding in games, and asymmetric communication networks.

We consider multi-agent path finding on strongly biconnected directed graphs. We show that all instances with at least two unoccupied positions can be solved or proven unsolvable. We present a polynomial-time algorithm for this class of problems, and analyze its complexity. Our work may be the first formal study of multi-agent path finding on directed graphs.

Introduction

Multi-agent path finding (MAPF) is an important computational problem, with applications in areas such as robotics and computer games. Not surprisingly, the problem has received a considerable attention in computer science areas such as artificial intelligence, robotics and graph theory.

In a common problem formulation, coined as *cooperative path finding*, the purpose is to navigate every agent from its original location to its target location, while avoiding collisions and deadlocks. The navigation environment is typically represented as a graph.

Current sub-optimal, scalable approaches work under the key assumption that the underlying graph is *undirected*. For instance, algorithms Push and Swap (Luna and Bekris 2011) and Push and Rotate (de Wilde, ter Mors, and Witteveen 2013) implement a core primitive, called swap, where agents must move in both directions along some graph edges. Among other moves, the swap primitive involves moving an agent to an adjacent node, to allow another agent to pass through, after which the first agents comes back to its former location, traversing the graph edge in the opposite direction. The MAPP algorithm (Wang and Botea 2011) relies on reverting part of recently performed moves, after an agent reaches its target. The similar use of edges in both directions appears in the BIBOX algorithm (Surynek 2009) where robots in cycles are rotated in order to relocate a selected robot and eventually rotated back after the robot gets out of the cycle to restore the situation.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

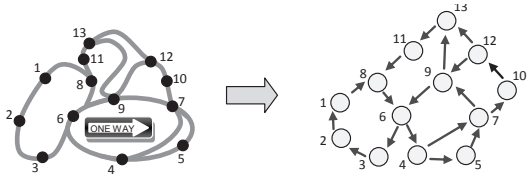

Figure 1: An example of uni-directional road network and its abstraction as a digraph. The elliptic road is uni-directional as a rule. Other roads are uni-directional, as the centrifugal force does not allow making sharp turns.

In navigation domains, uni-directional edges can arise from the properties of the environment, such as the existence of one-way entrances, exits, escalators, bridges and roads. Some agents could be able to travel down the hill or float down the river, but not the other way around. Furthermore, the literature shows approaches where uni-directional traffic is imposed on purpose on a game map, with the goal of avoiding head-to-head collisions between mobile agents (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).

Uni-directional networks also appear in traffic domains such as highways and railways. Connection lanes in highway system impose one way routing, as vehicles cannot make sharp turns when traveling at high speeds (see Figure 1). Similar situations arise at railway switches, which can switch a train to another track in one direction only (Bauer and Delling 2008).

We contribute the first tractability analysis of multi-agent path finding on directed graphs (digraphs). We focus on strongly biconnected digraphs, i.e., strongly connected digraphs where the undirected graphs obtained by ignoring the edge orientations have no cutting vertices. We demonstrate that all instances with at least two unoccuppied vertices can be solved or proven unsolvable in polynomial time. We introduce diBOX, a sub-optimal algorithm, and formally discuss its completeness, correctness, and complexity.

Previous formal studies of multi-agent pathfinding, sometimes also called coordinated pebble motion in graphs, ap-

pear to be focused on undirected graphs (Wilson 1974; Kornhauser, Miller, and Spirakis 1984). For instance, Wilson (1974) explicitly requires that the adjacency relation between agent configurations (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/how parts of the study, such as the considered permutation groups, would be applicable to directed graphs. We see our work as complementary to previous work on undirected graphs, being a step towards achieving a similar level of understanding for directed graphs.

Related Work

Motion planning problems related to MAPF assume the existence of one mobile robot and several mobile obstacles (Papadimitriou et al. 1994). This could be seen as a multi-agent pathfinding problem where only one agent has a specific target to achieve. Wu and Grumbach (2010) studied motion planning with one agent and several mobile obstacles on directed graphs, showing cases where the feasibility can be decided in linear time.

Much of the MAPF literature, including algorithms named in the introduction, focuses on undirected graphs. Part of the existing methods, however, can be applied to both directed and undirected graphs, even though they are typically evaluated on undirected graphs, such as grid maps. Examples include sub-optimal, incomplete methods (Silver 2006), as well as optimal algorithms (Standley 2010; Sharon et al. 2013). The latter are less scalable, which is not surprising, given that solving MAPF optimally has been shown to be NP-hard (Ratner and Warmuth 1986; Surynek 2010). A major focus in these contributions (Standley 2010; Sharon et al. 2013) is an improved speed performance in practice. Our contributions are substantially different, focusing on a theoretical analysis of MAPF on directed graphs, as explained in the last part of the introduction.

Background

In this section we overview a few concepts and results from the literature that form a starting point to our approach.

Definition 1. *Having a 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 so that at most one agent is placed in each vertex. The configuration is formally represented by 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 inverse configuration $\alpha^{-1} : V \longrightarrow A \cup \{\text{blank}\}$ which is well defined due to unique invertibility of α (*blank* is used for unoccupied vertices).

A configuration can be transformed to another by a move of agent. An agent can be moved in discrete time steps to the neighboring vertex provided the target vertex is unoccupied. The move is possible only along positive orientation of edges. For simplicity, in this theoretical study we assume that agents move one at a time.

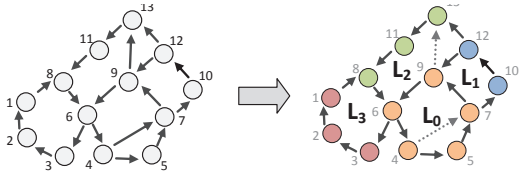

Figure 2: An example of a strongly biconnected digraph and its ear decomposition. Dashed edges are trivial derived ears (not explicitly labelled to avoid clutter).

Definition 2. *The instance of multi-agent path finding over directed graphs (dMAPF) 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_+.*

Definition 3. *An* ear decomposition *of a digraph $D = (V, E)$ is an ordered sequence of sub-digraphs 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 whose two endpoints belong to $D_i = \cup_{0 \leq j < i} L_j$, but no other vertices or edges of L_i belong to D_i.*

Each sub-digraphs L_i is called an ear (see Figure 2). An ear is *trivial* if it has exactly one edge (Bang-Jensen and Gutin 2008). An ear is *cyclic* if its endpoints are represented by a single vertex. We say that L_0 is the *basic cycle* and all other ears are *derived ears*. Given an ear L, $|L|$ denotes its number of vertices. The *interior* of a derived ear L, denoted as $\text{int}(L)$, refers to the contained vertices different from its endpoints. Its number of vertices is denoted as $|\text{int}(L)|$. The endpoints are called the entrance and the exit point, respectively. A digraph is said to be trivial iff it has only one vertex. In this paper, we work with non-trivial digraphs.

Given an ear decomposition $O = [L_0, L_1, \ldots, L_r]$, we call the k-prefix subgraph the subgraph restricted to the first k ears, $O_k = [L_0, L_1, \ldots, L_k]$.

Definition 4. *An* open ear decomposition *of a digraph D is an ear decomposition with no cyclic derived ears.*

An undirected graph G is biconnected if G is connected and there are no cut vertices in G. A digraph D is strongly connected if for any two distinct vertices v and w, there are both a path from v to w and a path from w to v in D. Given a digraph D, $\mathcal{G}(\mathcal{D})$ is the underlying graph of D, i.e., the undirected graph obtained by ignoring the orientation of the arcs (Wu and Grumbach 2010).

Definition 5 (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.*

Theorem 1 (Wu and Grumbach 2010)**.** *Let D be a non-trivial digraph. D is strongly biconnected iff D has an open ear decomposition. Moreover, any cycle can be the starting point of an open ear decomposition.*

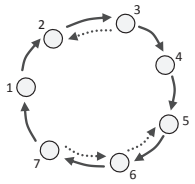

Figure 3: An example of bidirectional cycle over 7 vertices.

Solving Strongly Biconnected Digraphs

In this section we focus on solving multi-agent path finding instances on strongly biconnected digraphs. We will show that all instances with two or more unoccupied vertices (blanks) can be solved (or proven unsolvable) in polynomial time, and will present algorithms suitable for this problem.

Definition 6. *A digraph is a* partially-bidirectional cycle *if it consists of a simple cycle C, plus zero or more edges of the type (u, v), where $(v, u) \in C$ (i.e., edges obtained by swapping the direction of an edge from C - see Figure 3).*

Definition 7. *We say that an open ear decomposition of a strongly biconnected digraph is* regular *if the basic cycle L_0 has three or more vertices, and there exists a non-trivial derived ear with both ends attached to the basic cycle.*

Proposition 1. *For every strongly biconnected digraph $D = (V, E)$, exactly one of the following two cases holds:*

1. *D has a regular open ear decomposition; or*
2. *D is a partially-bidirectional cycle.*

Proof. Part A: Showing that at least one case holds. We start by noting that any strongly biconnected digraph with three or fewer vertices satisfies case 2. Thus, in the rest of Part A, we assume that $|V| \geq 4$.

Let $[L_0, \ldots, L_r]$ be an open ear decomposition for D. Unless L_0 already has three or more vertices, we modify the open ear decomposition slightly to obtain a basic cycle with that property. Indeed, when L_0 has two vertices, there must exist a derived non-trivial ear L, since the graph has 4 or more vertices. Ear L, together with one edge from L_0, become the new basic cycle in the decomposition. The other edge from L_0 becomes a trivial derived ear. In the rest of the analysis in Part A we can assume that $|L_0| \geq 3$.

At this point we distinguish between two scenarios. First, we consider the case when L_0 contains every vertex in V. If there exists an edge $e \in E \setminus L_0$ that connects two vertices p and q that are not next to each other in the basic cycle L_0, then the edge $e = (p, q)$ and part of L_0's edges create a new basic cycle L_0'. The remaining edges of L_0 create a non-trivial derived ear. We are now in case 1 of the proposition. If no such an edge e exists, we are in case 2.

Consider now the scenario when L_0 contains only a subset of the vertices in V. It remains to show that a non-trivial derived ear exists. Consider a vertex $w \notin L_0$ and a path from some vertex $p \in L_0$ to w. (Such a path always exists, as the digraph is strongly biconnected.) The first edge (r, s) on that path that does not belong to L_0 is the beginning of an ear with at least one interior vertex, namely s. It follows that one or more non-trivial derived ears exist. The first one in the ordering of the decomposition has both its endpoints on L_0. This corresponds to case 1 of the proposition.

Part B: Showing that at most one case holds. This can be proven by contradiction. Assume a given strongly biconnected digraph D is a partially-bidirectional cycle, and it has a regular open ear decomposition. As D is a partially-bidirectional cycle, the only options for the basic cycle L_0 that could occur in an open ear decomposition are: i) L_0 has two vertices; or ii) L_0 contains all vertices. Option i) contradicts the requirement that L_0 has at least three vertices. Option ii) contradicts the requirement that there exists at least one non-trivial derived ear. □

Partially-bidirectional cycles

In discussing multi-agent path finding on strongly biconnected digraphs, we start with the easy case of partially-bidirectional cycles. Then, in the next section, we discuss the complementary, more involved case of strongly biconnected digraphs with regular open ear decompositions.

Proposition 2. *An instance on a partially-bidirectional cycle, with at least one blank, has a solution if and only if the ordering of the agents in the initial state is identical with the ordering in the goal state.*

Proof. The proof is obvious, as no swapping between agents is possible: an instance is solvable if and only if the agents come in the right order in the first place. □

Digraphs with regular open ear decompositions

In this section we show that, on digraphs with a regular open ear decomposition, every instance with two or more blanks has a solution. For simplicity, assume that exactly two blanks are available. We present an algorithm capable of computing solutions in polynomial time.

Similarly to Surynek's (2009; 2014) strategy for undirected graphs, our algorithm solves the ears one by one, in the reverse order, solving L_0 at the end. When solving a derived ear, we never touch the interior of previously solved ears. A derived ear is solved if all *interior* positions labelled as goals are occupied by the corresponding agents, and all other interior positions (if any) are blank.

Borrowing blanks for L_0. With no generality loss, we assume that, in the goal configuration, the two blanks are inside the basic cycle L_0. When this doesn't hold, the instance is converted into another instance as follows. Identify a node g that must be blank in the goal, and a directed path from g to a location $n \in L_0$. Shift all goal positions backwards by one step along this path, resulting in a goal state where the blank is at $n \in L_0$. We call this procedure *BorrowBlanks*. At the end, agents are pushed along this path by one step each, reaching the original goal (method *ReturnBlanks*).

Solving derived ears. Solving a derived ear $L_i, i > 0$, with a goal configuration $q_1, \ldots q_z$, pushes agents inside the

ear in order, starting with q_z, the agent whose goal is the farthest away from the ear's entrance.

Assume that the agents $q_{l+1}, \ldots q_z$ have been pushed inside, on the first $z-l$ interior positions of L_i, and we need to insert the next agent q_l. For simplicity, and without any generality loss, assume that all interior positions of L_i are occupied, and the two available blanks are located elsewhere.

We distinguish between two cases. In case 1, the next agent q_l to insert is outside the ear L_i. First, we bring one blank to the last interior position of L_i. As blanks travel backwards in a directed graph, the positions of $q_{l+1}, \ldots q_z$ are not impacted. Then, agent q_l is brought to the entrance of the ear, without touching $\bigcup_{j \geq i} \text{int}(L_j)$. This is possible with one remaining blank in use, according to Theorem 14 in (Wu and Grumbach 2010). We call the method that can move an agent within a k-prefix subgraph *MoveAgentInSubgraph*. Then agent q_l is pushed inside L_i, reaching a configuration where the first $z - l + 1$ interior positions of L_i are occupied with agents $q_l, q_{l+1}, \ldots, q_z$. This completes case 1. Let us recall that agent relocation as suggested in (Wu and Grumbach 2010) can be implemented with the worst case time complexity $\mathcal{O}(|V|^2)$ and it produces $\mathcal{O}(|V|^2)$ moves.

Definition 8. *Let π be a simple path (chain) and let n be node in the path. An edge (n, m) is an* escape door *for π if either m does not belong to π, or m is in π, being at least two positions earlier than n in π.*

Intuitively, an escape door allows an agent to avoid moving forward on a given path π.

Proposition 3. *Let $O = [L_0, L_1, \ldots, L_r]$ be a regular open ear decomposition of a strongly biconnected digraph. For any derived ear $L_i, i > 0$, there exist a path π in O_{i-1}, from the exit to the entrance of L_i, such that π has an escape door (n, m).*

Proof sketch. Let a and b be the entrance and the exit of L_i. As O_{i-1} is strongly biconnected, it has a path from a to b and a path π from b to a. Assuming there is no escape door along π, it follows that O_i is in fact a chain of nodes (between a and b) with every two adjacent nodes linked in both directions. This contradicts the requirement that the basic cycle has 3 or more nodes. □

In case 2, the agent q_l is inside the ear. It has to be brought out first, after which the agents $q_{l+1}, \ldots q_z$, if any, will be reinserted. This method is called *TakeAgentOutsideEar*.

Here it is how it works. We build a simple cycle C' by putting together L_i and π, and bring two blanks to nodes m and n mentioned in Proposition 3. Agents are pushed forward (rotated) repeatedly, in the cycle C', until 1) all agents $q_{l+1}, \ldots q_z$ (if any) reach again their positions inside L_i; and 2) q_l is outside the interior of L_i. Besides forward moves along C', there is one exception that applies whenever q_l is at node n. Instead of pushing q_l forward along C', q_l is pushed through the escape door to m.

Even if $m \in C'$, recall that m is not the node right before n in the cycle. This ensures that pushing q_l from n to m does not cause a deadlock (i.e., there is no repetition of the previous global agent configuration).

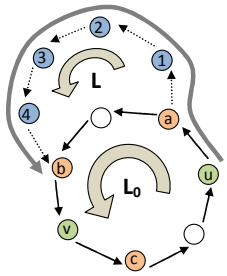

Figure 4: Bringing u (Mickey) next to v (Minnie) as part of solving the basic cycle. Mickey travels along L in such a way that, at the end, L preserves its goal configuration. Apart from relocating u, all other ordering relations in L_0 are preserved.

After applying method *TakeAgentOutsideEar*, q_l is outside $\text{int}(L_i)$ and agents $q_{l+1}, \ldots q_z$, if any, are inside L_i. This fits case 1 and the processing continues as in that case.

Solving the basic cycle. When ordering agents inside the cycle L_0, we make use of a non-trivial derived ear L with both ends connected to L_0. Such an ear L always exists in a regular decomposition. All agents belonging to interior positions of L are already solved, as described earlier. Let $q_1, \ldots q_z$ be these agents in the order they are arranged on their goal positions in the ear L. Recall that there are two blanks available in L_0.

Assume that two agents u (Mickey) and v (Minnie) need to be brought next to each other in L_0, in the order u, v in the direction L_0. This process, described in detail below, is referred to as method *BringAgentsNextToEachOther*. A high-level overview of the process is shown in Figure 4. As highlighted in Figure 5, this is split into several stages:

- Mickey's departure: Rotate agents in the basic cycle L_0 until u is at the entrance of the ear L, and a blank is available at the exit of the ear.
- Mickey's admission into the bubble ride: Push all agents in L one step further, so that agent u is on the first interior position of L, and q_z is in L_0.
- Mickey's bubble ride: Bring u along the ear L, until u reaches the last interior position of L. The bubble ride needs to be more sophisticated than simply pushing all agents forward along L until agent u is on the last position. The reason is that L's internal configuration has to be kept in such a way that, at the end of the bubble ride, L's goal configuration is very easy to restore.

  A bubble ride is a series of macro-steps. Each macro step progresses u along L by one position, and also re-inserts back into L an agent temporarily removed from L. Agents are re-inserted in the same order in which they left the ear L, reconstructing L's goal configuration step by step.

  The dashed box in Figure 5 illustrates one macro step, changing the configuration of $\text{int}(L)$ from $u, 1, 2, 3$ into $4, u, 1, 2$ (listing agents from the first to the last interior

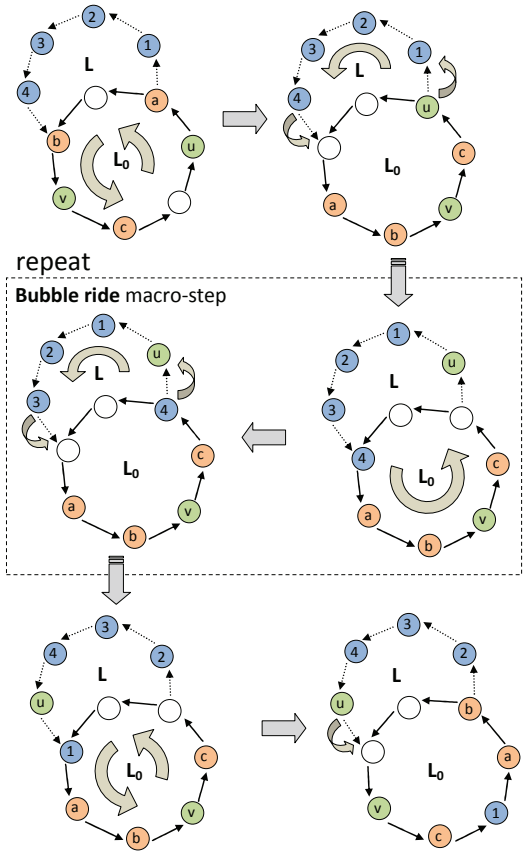

Figure 5: Top: Mickey's departure (left) and admission into the bubble ride (right). In the dashed box, one macro-step as part of a bubble ride: rotation inside L_0 (right), followed by progress along L (left). Bottom: Minnie's trip (left) and reunification (right). Clean-up not shown.

vertex, in the direction of travel along L). More generally, the k-th macro step starts from a configuration of int(L) such as $q_{z-k+2}, \ldots, q_z, u, q_1, \ldots, q_{z-k}$.[1] Agent q_{z-k+1} is inside L_0.

A macro-step has two "sub-macros". First, agents are rotated inside L_0 until a blank is available at the exit from L, and q_{z-k+1} is at the entrace of L. Then, all agents along L are pushed one step forward, pushing q_{z-k} into L_0, and pushing agent q_{z-k+1} onto the first interior position of L.

At the end of the k-th macro step, the configuration of int(L) is $q_{z-k+1}, \ldots, q_z, u, q_1, \ldots, q_{z-k-1}$. Agent q_{z-k} is inside L_0. All other agents inside L_0 preserve their ordering as it existed before applying the macro at hand. At the end of the current bubble ride, after applying the last macro, the configuration of int(L) is $q_2, \ldots, q_z, u$. Agent q_1 is inside L_0. All other agents in L_0 preserve their relative ordering that existed before starting the bubble ride. In the running example, the resulting configuration of int(L) is $2, 3, 4, u$ (Figure 5, bottom left).

- Minnie's trip (Figure 5, bottom left): Rotate agents in the basic cycle L_0 until v is placed right after the exit from L, and the exit is empty.
- Reunification (Figure 5, bottom right): Advance agents inside L one step further, placing u adjacent to v, in the desired order, inside L_0, and leaving a blank at the first interior position of L.
- Clean up: Rotate agents inside L_0 until q_1 is at the entrance of L, and then push q_1 inside L.

After applying method *BringAgentsNextToEachOther*, L has preserved its goal configuration. Agents u and v are next to each other in the desired order. Apart from repositioning u, no other ordering relationships were affected inside L_0. The process is repeated until the basic cycle L_0 is solved, which completes solving the entire instance.

The solving strategy outlined in the previous paragraphs allows us to formulate the following result.

Proposition 4. *On a strongly biconnected digraph with a regular open ear decomposition, all multi-agent path finding instances with two or more blanks have a solution.*

In summary, a strongly biconnected digraph either is a partially bidirectional cycle, or it admits a regular open ear decomposition. On partially bidirectional cycles, all instances with at least one blank can be solved or proven unsolvable. On digraphs with a regular open ear decomposition, all instances with at least two blanks have a solution.

Algorithm Analysis

We put together all pieces described in the previous section, obtaining an algorithm that we call diBOX. We show its pseudo-code in Algorithm 1 and analyze its complexity.

Rotate is a one-step rotation of all the agents contained within a cycle C with at least one blank. It consumes $|C|$ steps and produces $|C|$ moves.

Given a cycle C, a goal configuration α_+ and an agent u, the agent that should be next to u, in the positive orientation of C, is provided in constant time by the function *NextAgent*.

As the configuration in the basic cycle may be shifted with respect to the required goal configuration after putting agents into the right order, several final rotations of the basic cycle may be necessary. This final correction is done with method *ShiftAgentsInCycle*. The operation consumes a time of $\mathcal{O}(|C|^2)$, where C is the cycle at hand, and produces the same number of moves. Indeed, $\mathcal{O}(|C|)$ rotations may be necessary, while one rotation needs $|C|$ steps and moves.

If the goal configuration does not have two blanks in the basic cycle, the instance is transformed with procedure *BorrowBlanks*. At the end, the original goal configuration is restored with procedure *ReturnBlanks*. These two methods can run in a time of $\mathcal{O}(|E|)$, producing $\mathcal{O}(|V|)$ moves.

Proposition 5. *The worst-case time complexity of diBOX is within $\mathcal{O}(|V|^3)$ and so is the number of moves.*

Proof. Checking if $D = (V, E)$ is a bidirectional cycle can be done by picking a vertex and by trying to traverse all

[1] When $l > l'$, sequence $q_l \ldots q_{l'}$ is defined as the empty set.

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_+

diBOX $(D, A, \alpha_0, \alpha_+)$
if D is a partially bidirectional cycle **then**
  if α_0 and α_+ represent the same ordering in D **then**
    ShiftAgentsInCycle (D, α_+)
  else
    return *UNSOLVABLE*
else
  let $[L_0, L_1, \ldots, L_r]$ be a regular open ear decomp. with non-trivial ear L_1 attached to L_0
  if $|\{x \in L_0 | \alpha_+^{-1}(x) = \text{blank}\}| < 2$ **then**
    $\alpha'_+ \leftarrow$ *BorrowBlanks* $\left(D, L_0, \alpha_+\right)$
    SolveEars $\left(D, [L_0, L_1, \ldots, L_r], \alpha_0, \alpha'_+\right)$
    ReturnBlanks $\left(D, \alpha_+, \alpha'_+\right)$
  else
    SolveEars $\left(D, [L_0, L_1, \ldots, L_r], \alpha_0, \alpha_+\right)$

SolveEars $\left(D, [L_0, L_1, \ldots, L_r], \alpha_0, \alpha_+\right)$
$\alpha \leftarrow \alpha_0$
for $i = r$ **downto** 1 **do**
  SolveDerivedEar $\left(D, L_i, \alpha_+\right)$
SolveBasicCycle $\left(L_0, L_1, \alpha_+\right)$

SolveDerivedEar $\left(D, L_i, \alpha_+\right)$
let $L_i = [x_0, x_1, x_2, \ldots, x_{z+1}]$
let C' be a cycle containing L_i in $D \setminus \bigcup_{j>i} \text{int}(L_j)$
for $l = z$ **downto** 1 **do**
  $q_l \leftarrow \alpha_+^{-1}(x_l)$
  if $\alpha^{-1}(x_0) \neq q_l$ **then**
    if $q_l \in \{\alpha^{-1}(x_z), \ldots, \alpha^{-1}(x_{z-l+1})\}$ **then**
      TakeAgentOutsideEar $\left(q_l, L_i, C'\right)$
    MoveAgentInSubgraph $\left(q_l, x_0, \bigcup_{j=1}^{i-1} L_j\right)$
  Rotate $\left(C'\right)$

SolveBasicCycle $\left(L_0, L_1, \alpha_+\right)$
let $L_0 = [x_1, x_2, \ldots, x_z]$
let L_1 be connected to L_0 in x_a and x_b
for $l = 1$ **to** $z - 1$ **do**
  $u \leftarrow \alpha^{-1}(x_l)$
  if $u \neq$ blank **then**
    $v \leftarrow$ *NextAgent* $\left(x_l, L_0, \alpha_+\right)$
    BringAgentsNextToEachOther $\left(L_0, L_1, u, v\right)$
ShiftAgentsInCycle $\left(L_0, \alpha_+\right)$

the vertices by following edges that lead to not yet visited vertices. If all the vertices of V are traversed and there is an edge from the last visited vertex to the initially picked one that completes a cycle then it remains to check unused edges. If they connect only vertices that follow each other in the cycle then D is a bidirectional cycle. In all other cases, that is if we do not manage to find a cycle of if there is an unused edge connecting other than consecutive vertices in the cycle, D is not a bidirectional cycle. All this checking can be done in $\mathcal{O}\left(|V| + |E|\right)$ steps.

A regular open ear decomposition $[L_0, L_1, \ldots, L_r]$ of required properties can be found in the worst-case time of $\mathcal{O}\left(|V| + |E|\right)$ (Schmidt 2013). Note that trivial derived ears are skipped by the algorithm. Ignoring these does not affect the completeness of diBOX. The number of remaining edges is within $\mathcal{O}\left(|V|\right)$, since, in every non-trivial ear, the numbers of edges and interior vertices differ by 1. This observation enables finding a path or a cycle connecting selected two vertices in $\mathcal{O}\left(|V|\right)$ steps by the breadth first search.

Processing a single agent when solving a derived ear L_i involves at most $|C'|$ rotations of the cycle C' associated with L_i, as well as at most two relocations of q_l within a sub-graph. As a single rotation requires $\mathcal{O}\left(|C'|\right)$ steps and produces the same number of moves, a single agent consumes $\mathcal{O}(|C'|^2)$ moves in rotations. The agent relocations add $\mathcal{O}(|V|^2)$ moves and time (Wu and Grumbach 2010). Altogether, processing all the agents in derived ears requires a time of $\mathcal{O}(|V|^3)$ and produces $\mathcal{O}(|V|^3)$ moves.

It remains to analyze the solving the basic cycle L_0. Bringing agents next to each other requires $\mathcal{O}\left(|L_0|\right)$ rotations of L_0, to bring the first agent to the entrance of the non-trivial ear L_1; then the second agent needs to be brought to the exit of the ear; and finally an agent that got out of the ear into L_0 needs to be put back. Rotations within L_0 consume a time of $\mathcal{O}(|L_0|^2)$ and produce the same number of moves. For all the agents in L_0, both the time and the number of moves are within $\mathcal{O}(|L_0|^3)$, which boils down to $\mathcal{O}(|V|^3)$.

A single bubble ride macro-step requires $|L_1|$ rotations of a cycle of a size bounded by $|L_0| + |L_1|$. This includes rotations necessary to make a movement of an agent inside and outside L_1. Counting all the agents for which the bubble ride is done, we obtain both a time and a number of moves of $|L_0| \cdot |L_1| \cdot (|L_0| + |L_1|)$, which boils down to $\mathcal{O}(|V|^3)$. The time and the number of moves for remaining operations such as borrowing and returning blanks is dominated by $\mathcal{O}(|V|^3)$. Altogether, the worst-case time complexity of the diBOX algorithm is $\mathcal{O}(|V|^3)$ and it produces $\mathcal{O}(|V|^3)$ moves. □

Conclusion

We have performed a formal study of multi-agent path finding on strongly biconnected digraphs. We found that instances with at least two blanks have a solution, except for the trivial case of badly ordered instances on partially biconnected cycles. We have presented a sub-optimal algorithm, called diBOX, whose worst-case time complexity and solution length are both within $\mathcal{O}(|V|^3)$, where V is the set of vertices. Similarly to BIBOX (Surynek 2009), a method for undirected biconnected graphs, our algorithm solves ears in reverse order. However, the restriction to uni-directional movements makes our algorithm more intricate than the case when bi-directional movement is possible.

In future work, we plan to extend our analysis to other classes of directed graphs. We are also interested in implementing and evaluating the performance of our method.

Acknowledgement

This research is partially supported by the Charles University in Prague within the PRVOUK project.

References

Bang-Jensen, J., and Gutin, G. Z. 2008. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition.

Bauer, R., and Delling, D. 2008. SHARC: fast and robust unidirectional routing. In *Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments, ALENEX 2008, San Francisco, California, USA, January 19, 2008*, 13–26.

de Wilde, B.; ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: Cooperative multi-agent path planning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, 87–94. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Jetcheva, J. G., and Johnson, D. B. 2006. Routing characteristics of ad hoc networks with unidirectional links. *Ad Hoc Netw.* 4(3):303–325.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. 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)*, 241–250.

Luna, R., and Bekris, K. E. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *IJCAI-11*, 294–300.

Marina, M. K., and Das, S. R. 2002. Routing performance in the presence of unidirectional links in multihop wireless networks. In *Proceedings of ACM MobiHoc*, 12–23.

Papadimitriou, C. H.; Raghavan, P.; Sudan, M.; and Tamaki, H. 1994. Motion planning on a graph. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, 511–520. Washington, DC, USA: IEEE Computer Society.

Ratner, D., and Warmuth, M. K. 1986. Finding a shortest solution for the n $\times$ n extension of the 15-puzzle is intractable. In *Proceedings of AAAI-86*, 168–172.

Schmidt, J. M. 2013. A simple test on 2-vertex and 2-edge-connectivity. *Information Processing Letters* 113(7):241–244.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artif. Intell.* 195:470–495.

Silver, D. 2006. Cooperative pathfinding. *AI Programming Wisdom*.

Standley, T. S. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of AAAI-10*, 28–29.

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

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In Fox, M., and Poole, D., eds., *Proceedings of AAAI-10*. AAAI Press.

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

Wang, K.-H. C., and Botea, A. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding. In *ICAPS*, 380–387.

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

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

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