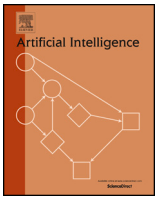
Artificial Intelligence 347 (2025) 104372

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Multi-agent pathfinding on strongly connected digraphs: Feasibility and solution algorithms

S. Ardizzoni [a,*], L. Consolini [a], M. Locatelli [a], B. Nebel [b], I. Saccani [a]

[a] Dipartimento di Ingegneria e Architettura, Università di Parma, Parco Area delle Scienze, 181/A, Parma, Italy
[b] Institut fur Informatik, Albert-Ludwigs-Universitat, Georges-Kohler-Allee 52, Freiburg, Germany

ABSTRACT

On an assigned graph, the problem of Multi-Agent Pathfinding (MAPF) consists in finding paths for multiple agents, avoiding collisions. Finding the minimum-length solution is known to be NP-hard, and computation times grows exponentially with the number of agents. However, in industrial applications, it is important to find feasible, suboptimal solutions, in a time that grows polynomially with the number of agents. Such algorithms exist for undirected and biconnected directed graphs. Our main contribution is to generalize these algorithms to the more general case of strongly connected directed graphs. In particular, we describe a procedure that checks the problem feasibility in linear time with respect to the number of vertices n, and we find a necessary and sufficient condition for feasibility of any MAPF instance. Moreover, we present an algorithm (diSC) that provides a feasible solution of length $O(kn^2c)$, where k is the number of agents and c the maximum length of the corridors of the graph.

1. Introduction

We consider a set of agents (or *pebbles*) on a graph $G = (V, E)$, with vertex (or node) set V and edge set E. Each agent occupies a different node and may move to unoccupied positions. The Multi-Agent Path Finding (MAPF) problem [25] consists in computing a sequence of movements (called *moves*) that repositions all agents to assigned target nodes, avoiding collisions. Depending on the environment, different definitions of a conflict are possible (see, e.g., [25]). According to the type of conflicts that are prohibited, different variants of the Multi-Agent Path Finding (MAPF) problem can be defined. Two types of conflicts that are common to all MAPF variants are *vertex conflicts*, which occur when two or more agents are scheduled to occupy the same vertex at the same time step, and *edge conflicts*, which arise when two agents attempt to traverse the same edge in the same direction at the same time. Another commonly prohibited conflict in the general MAPF problem is the *swapping conflict*, which occurs when two agents are scheduled to swap locations in a single time step. Recently, a MAPF variant has been introduced that does not consider this type of conflict and, instead, allows swaps between adjacent agents as the only permitted movement. This variant, known as Token Swapping (TSWAP) [9,26] considers a set of colored tokens, each placed on a vertex of the graph, with the objective of performing a sequence of swaps so that the token colors match the designated vertex colors. Another type of collision for MAPF is the *following conflict* that arises when an agent occupies a vertex that was occupied by another agent in the previous time step. There exist other variants of MAPF that impose different types of movement constraints on the agents. For instance, Pebble Motion on Graphs (PMG) [12,15] is a variant of MAPF in which it is assumed that agents (or pebbles) move one at a time, rather than simultaneously. Note that as a direct consequence of this constraint, avoiding vertex conflicts also avoids edge, swapping, and following conflicts.

In this paper, we deal with strongly connected digraphs, directed graphs in which it is possible to reach any node starting from any other node. The main motivation comes from the management of fleets of automated guided vehicles (AGVs). AGVs move items

* Corresponding author.
E-mail address: stefano.ardizzoni@unipr.it (S. Ardizzoni).

https://doi.org/10.1016/j.artint.2025.104372
Received 16 January 2024; Received in revised form 28 February 2025; Accepted 28 May 2025

Available online 4 June 2025

between different locations in a warehouse. Each AGV follows predefined paths, that connect the locations in which items are stored or processed. We associate the paths' layout to a directed graph. The nodes represent positions in which items are picked up and delivered, together with additional locations used for routing. The directed arcs represent the segments that make up the precomputed paths that connect these locations. If various AGVs move in a small scenario, each AGV represents an obstacle for the other ones. In some cases, the fleet can reach a deadlock situation, in which every vehicle is unable to reach its target. Hence, it is important to find a feasible solution to MAPF, even in crowded configurations.

In this paper, we address the problem of finding a feasible solution for PMG. Testing feasibility for pebble motion is equivalent to testing feasibility of MAPF with forbidden following conflicts. In particular, any feasible solution for the pebble motion problem is also a feasible solution for MAPF. Usually, such feasible solution is far from being optimal, but it can be improved, for instance, by employing local search procedures designed to accommodate simultaneous movements of agents (see, e.g., [4]).

Literature review. Various works address the problem of finding the optimal solution of MAPF (i.e., the solution with the minimum number of moves). For instance, Conflict Based Search (CBS) is a two-level algorithm which uses a search tree, based on conflicts between individual agents (see [23]). However, finding the optimal solution of MAPF is NP-hard (see [30]), and computational time grows exponentially with the number of agents.

Search-based suboptimal solvers aim to provide a high quality solution, but are not complete (i.e., they are not always able to return a solution). A prominent example is Hierarchical Cooperative A* (HCA*) [24], in which agents are planned one at a time according to some predefined order.

Instead, rule-based approaches include specific movement rules for different scenarios. They favor completeness at low computational cost over solution quality. Since their aim is to detect feasible solutions, agents are supposed to move one at a time [6,10,11,15]. Of course, once we have found a feasible solution with one of these algorithms, we can find a shorter solution by allowing simultaneous moves. For instance, we can search in a neighborhood of the feasible solution, as in [4,18]. Many of the rule-based algorithms deal with the special case of trees. In this case the problem is called *pebble motion on a tree* (PMT). Auletta et al. [6] present an algorithm for the feasibility of PMT, from which a solution can be derived requiring $O(k^2(n-k))$ moves, where n is the number of nodes and k the number of pebbles. Korshid et al. [14] present an algorithm for PMT (called TASS), easy to understand and implement, which provides solutions of $O(n^4)$ moves. Recently Ardizzoni et al. [5] propose the *Leaves Procedure for* PMT, simple and very easy to implement, which finds solutions with a number of moves $O(knc + n^2)$, where c represents the maximum length of the corridors (paths whose internal nodes all have degree equal to two and whose end nodes have degree different from two) in the tree. Moreover, they discuss a variant of the PMT problem, called PMT *with trans-shipment vertices* (*ts*-PMT). Trans-shipment [5,12,16] is a particular type of vertex that cannot host pebbles. This variant can be solved with the *Leaves Procedure for* PMT with some minor modifications and it is useful since any MAPF instance on an undirected graphs can be reduced to an instance of this problem on a tree. For example, reference [16] presents a method that converts the graph into a tree with trans-shipment vertices (as in [12]), and solves the resulting problem with TASS. Another important rule-based algorithm for MAPF on undirected graphs is *Push and Rotate* [1,11], which solves every MAPF instance on graphs that contains at least two holes (i.e., unoccupied vertices). One of the best rule-based algorithms for MAPF in terms of length complexity is presented by Kornhauser. Indeed, in his Ph.D. thesis [15] he describes a procedure that finds solutions with $O(n^3)$ moves. However, this approach is very difficult to implement [22]. In all the problems considered thus far, we assume that each agent is assigned a specific target. However, there exists a variant, known as the unlabeled problem, where each agent is allowed to reach any of the available targets, and it is only required that all targets are reached by exactly one agent. For instance, Le Bodic [17] addresses the unlabeled pebble motion problem on trees (UPMT) and introduces the first optimal algorithm, which is asymptotically as efficient as possible. The algorithm operates in time linear with respect to both the size of the input (the tree) and the size of the output (the optimal plan). Furthermore, it demonstrates that the length of the optimal solution is less than or equal to $k(n-1)$.

The literature cited so far concerns exclusively undirected graphs, where motion is permitted in both directions along graph edges. Fewer results are related to directed graphs. Nebel [19] proves that finding a feasible solution of MAPF on a general directed graph (digraph) is NP-hard. However, in some special cases this problem can be solved in polynomial time. One relevant reference is [10], which solves MAPF on the specific class of biconnected digraphs, i.e., strongly connected digraphs where the undirected graphs obtained by ignoring the edge orientations have no cutting vertices. The latter is a node of a connected graph which, if removed, makes the graph not connected. The proposed algorithm has polynomial complexity with respect to the number of nodes.

Statement of contribution. We consider MAPF on strongly connected digraphs, a class that is more general than biconnected digraphs, already addressed in [10]. This paper is an extension of our previous work [2]. We present three main contributions:

1. In Section 5 we present a procedure, based on [12], that checks the problem feasibility in linear time with respect to the number of nodes (see Theorem 5.2). This approach can also be used to solve MAPF, but it leads to very redundant solutions.
2. In Section 5.1 we provide a necessary and sufficient condition for feasibility of any MAPF on a strongly connected digraph, adapted from [15] and [5].
3. In Section 6 we present diSC (digraph Strongly Connected) algorithm that finds a solution for all admissible problems with at least two holes, extending the method in [16] (see Section 3). The details of diSC are described in the Appendix. Moreover, in Section 7 we prove that the length complexity of this algorithm is $O(kn^2c)$.

2. Problem definition

Let $G = (V, E)$ be a digraph, with vertex set V and edge set E. We assign a unique label to each pebble and hole. Sets P and H contain the labels of the pebbles and, respectively, the holes. Each vertex of G is occupied by either a pebble or a hole, so that

$|V| = |P| + |H|$. A *configuration* is a function $\mathcal{A} : P \cup H \to V$ that assigns the occupied vertex to each pebble or hole. A configuration is *valid* if it is one-to-one (i.e., each vertex is occupied by only one pebble or hole). Set $\chi \subset \{P \cup H \to V\}$ represents all valid configurations.

Given a configuration $\mathcal{A}$ and $u, v \in V$, we denote by $\mathcal{A}[u, v]$ the configuration obtained from $\mathcal{A}$ by exchanging the pebbles (or holes) placed at u and v:

$$\mathcal{A}[u,v](q) := \begin{cases} v, & \text{if } \mathcal{A}(q) = u; \\ u, & \text{if } \mathcal{A}(q) = v; \\ \mathcal{A}(q), & \text{otherwise.} \end{cases} \tag{1}$$

Function $\rho : \chi \times E \to \chi$ is a partially defined transition function such that $\rho(\mathcal{A}, u \to v)$ is defined if and only if v is empty (i.e., occupied by a hole). In this case $\rho(\mathcal{A}, u \to v)$ is the configuration obtained by exchanging the pebble or the hole in u with the hole in v. Notation $\rho(\mathcal{A}, u \to v)!$ means that the function is defined. In other words $\rho(\mathcal{A}, u \to v)!$ if and only if $(u, v) \in E$ and $\mathcal{A}^{-1}(v) \in H$, and, if $\rho(\mathcal{A}, u \to v)!$, $\rho(\mathcal{A}, u \to v) = \mathcal{A}[u, v]$. Note that the hole in v moves along edge $u \to v$ in reverse direction, while the pebble or hole on u moves to v.

We represent plans as ordered sequences of directed edges. It is convenient to view the elements of E as the symbols of a language. We denote by E^* the Kleene star of E, that is the set of ordered sequences of elements of E with arbitrary length, together with the empty string ϵ:

$$E^* = \bigcup_{i=1}^{\infty} E^i \cup \{\epsilon\}.$$

We extend function $\rho : \chi \times E \to \chi$ to $\rho : \chi \times E^* \to \chi$, by setting $(\forall \mathcal{A} \in \chi)\rho(\mathcal{A}, \epsilon)!$ and $\rho(\mathcal{A}, \epsilon) = \mathcal{A}$. Moreover, $(\forall s \in E^*, e \in E, \mathcal{A} \in \chi)$ $\rho(\mathcal{A}, se)!$ if and only if $\rho(\mathcal{A}, s)!$ and $\rho(\rho(\mathcal{A}, s), e)!$ and, if $\rho(\mathcal{A}, se)!$, $\rho(\mathcal{A}, se) = \rho(\rho(\mathcal{A}, s), e)$. A *move* is an element of E, and a *plan* is an element of E^*. Note that ϵ is the trivial plan that keeps all pebbles and holes on their positions. Note that a *move* corresponds to a movement action for a single agent, while all other agents execute a *waiting* action, i.e., they remain at their current node. This definition is consistent with the assumption that agents cannot move simultaneously. We define an equivalence relation $\sim$ on E^*, by setting, for $s, t \in E^*$, $s \sim t \leftrightarrow (\forall \mathcal{A} \in \chi \text{ s.t. } \rho(\mathcal{A}, s)!, \rho(\mathcal{A}, t)!)\, \rho(\mathcal{A}, s) = \rho(\mathcal{A}, t)$. In other words, two plans are equivalent if they reconfigure pebbles and holes in the same way. Given a configuration $\mathcal{A}$ and a plan f such that $\rho(\mathcal{A}, f)!$, a plan f^{-1} is a *reverse* of f if $\mathcal{A} = \rho(\mathcal{A}, f), f^{-1})$ (i.e., f^{-1} moves each pebble and hole back to their initial positions). We can also write $f f^{-1} \sim \epsilon$, so that f^{-1} behaves like a right-inverse. In Proposition 5.1 of Section 5 we will state that in the particular case of a strongly connected digraph, each plan has a reverse plan.

Our main problem is the Multi-Agent Path Finding (MAPF), which consists in finding a plan that re-positions all pebbles to assigned target vertices, avoiding collisions. For this problem, and the ones we will present later, the position of the holes is not relevant. Therefore, we introduce an equivalence relation $\dot{\sim}$ between configurations

$$\mathcal{A}^1 \dot{\sim} \mathcal{A}^2 \iff \forall\, p \in P \quad \mathcal{A}^1(p) = \mathcal{A}^2(p),$$

and we indicate with $\tilde{\mathcal{A}}^1$ the equivalence class of $\mathcal{A}^1$.

Definition 2.1 *(MAPF problem).* Let $G = (V, E)$ be a graph, P be a pebble set, $\tilde{\mathcal{A}}^s$ an initial valid configuration, $\tilde{\mathcal{A}}^t$ a final valid configuration. The MAPF instance $\langle G, \tilde{\mathcal{A}}^s, \tilde{\mathcal{A}}^t \rangle$ consists in finding a plan f such that $\tilde{\mathcal{A}}^t = \rho(\tilde{\mathcal{A}}^s, f)$.

We denote by $|f|$ the length of a plan f (i.e., the number of moves of f). If f is the solution to a MAPF instance, then $|f|$ represents the total number of actions, including waiting actions, made by each agent in such a way that each agent reaches its target, or, equivalently, the number of time steps needed to achieve the final configuration, which is the Makespan.

If G is an undirected tree, this problem is called *pebble motion on trees* (**PMT**). A variant of PMT problem is *pebble motion on trees with trans-shipment vertices* (**ts-PMT**). This problem is a PMT on a tree in which the vertex set V is partitioned in trans-shipment and regular vertices.

Definition 2.2. A *trans-shipment vertex* is a vertex with degree greater than one that cannot host a pebble: pebbles can cross this node, but cannot stop there. More formally, given a trans-shipment vertex s,

1. $\deg(s) \geq 2$;
2. $\rho(\mathcal{A}, (u \to s)(w \to v))!$ if and only if $w = s$, $(u, s), (s, v) \in E$, and $\mathcal{A}^{-1}(v) \in H$. If $\rho(\mathcal{A}, (u \to s)(w \to v))!$, then $\rho(\mathcal{A}, (u \to s)(w \to v)) = \mathcal{A}[u, v]$.

The second property means that, if a pebble is moved to a trans-shipment vertex, then it must be immediately moved to another node. We denote with V^t the set of all the trans-shipment vertices and $V^r = V \setminus V^t$ the set of regular vertices. Ardizzoni et al. [5] presented an algorithm to solve ts-PMT in polynomial time.

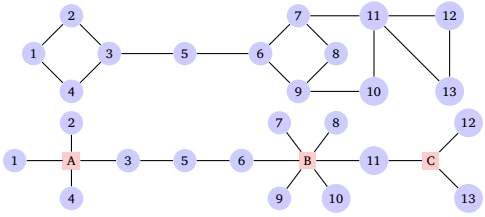


Fig. 1. Undirected graph and corresponding biconnected component tree. A, B, and C are the trans-shipment vertices.

Another variant of MAPF is the *motion planning problem*, which consists in finding a plan such that a single marked pebble reaches a desired target vertex, avoiding collisions with other pebbles, that are moving *obstacles* [5,7,21,28,29]. We recall its definition.

Definition 2.3 *(Motion Planning problem).* Let $G = (V, E)$ be a graph, P a set of pebbles, and $\tilde{\mathcal{A}}^s$ an initial valid configuration. Given a pebble $\bar{p}$ and a target node $t \in V$, find a plan f such that $t = \rho(\tilde{\mathcal{A}}^s, f)(\bar{p})$. We indicate such a plan with notation $\tilde{\mathcal{A}}(\bar{p}) \Rightarrow t$.

Wu and Grumbach [28] discuss the feasibility of the motion planning problem on a strongly biconnected digraph and prove the following:

Theorem 2.4. *(Theorem 14 of [28]) Let D be a strongly biconnected digraph, P a set of pebbles and H a set of holes. Then any motion planning problem on D is feasible if and only if $|H| \geq 1$.*

Ardizzoni et al. [5] provide an algorithm (CATERPILLAR) which finds solutions of this problem on a tree with $O(nc)$ moves.

3. Solving MAPF on undirected graphs

In this section, we recall the planning method for a connected undirected graph as introduced by Krontiris et al. [16]. The main idea is to transform the graph $G = (V, E)$ into a tree $T := \mathcal{T}(G)$, and the MAPF problem into a ts-PMT problem. It is possible to prove that the MAPF problem is solvable on G if and only if the corresponding ts-PMT problem is solvable on T [12]. Moreover, the solution of MAPF can be obtained from the solution of the corresponding ts-PMT [16].

3.1. Convert MAPF into ts-PMT

Given a connected graph $G = (V, E)$, we construct the *biconnected component tree* $\mathcal{T}(G) = (V_T, E_T)$ as follows. We initialize $V_T = V$, $E_T = E$, and we convert each maximal non-trivial (i.e., with at least three vertices) biconnected component $S = (V_S, E_S) \subset G$ into a star subgraph. The nodes in V_S are the leaves of the star. The internal node of the star is a newly added trans-shipment vertex.

The conversion of S into a star involves the following steps:

1. add a trans-shipment vertex s,
2. remove every edge $e \in E_S$,
3. add the edges $\{(u, s) | u \in V_T\}$.

Note that $V_T = V \cup V^t$, where V^t is the set of all trans-shipment vertices and V represents the set of regular vertices of tree T. Note that trans-shipment vertices of the biconnected component tree have the following properties:

- $|V^t| = K$, where K is the number of non-trivial biconnected component of G;
- $\forall\, v \in V^t$ $\deg(v) \geq 3$;
- $\forall\, v, w \in V^t$ $d(v, w) > 1$, where $d(v, w)$ is the edge-distance between v and w.

The latter property, which means that two trans-shipment vertices cannot be adjacent, is introduced in the definition of ts-PMT in [5].

Note that G and $\mathcal{T}(G)$ have a similar structure. Biconnected components of G correspond to star subgraphs in $\mathcal{T}(G)$, with trans-shipment vertices as internal nodes. Fig. 1 shows an undirected graph and its corresponding biconnected component tree. Building $\mathcal{T}(G)$ from G takes a linear time with respect to $|V|$ [13].

G and $\mathcal{T}(G)$ have the same number of pebbles but a different number of holes. Denoting with H_T the set of holes of the tree, it holds that $H_T = H \cup H^t$, where H^t is a new set of holes added for trans-shipment vertices ($|H_T| = |V_T|$).

Let $\mathcal{A} : P \cup H \to V$ be a configuration on G. We associate it to a configuration on $\mathcal{T}(G)$, $\mathcal{A}_T : P \cup H_T \to V_T$ such that ($\forall q \in P \cup H$) $\mathcal{A}_T(q) = \mathcal{A}(q) \subset V$ and $\mathcal{A}_T(H_t) = V_T$. Note that $\tilde{\mathcal{A}}_T|_{P \cup H} = \tilde{\mathcal{A}}$, since pebbles are on the same vertices. In this way, we associate every MAPF instance $\langle G, \tilde{\mathcal{A}}^s, \tilde{\mathcal{A}}^t \rangle$ to a ts-PMT instance $\langle \mathcal{T}(G), \tilde{\mathcal{A}}^s_T, \tilde{\mathcal{A}}^t_T \rangle$. Reference [12] proves the following important result.

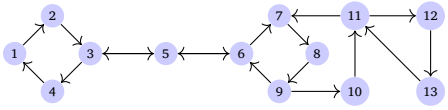


Fig. 2. Example of strongly connected digraph: the corresponding underlying graph and biconnected component tree are shown in Fig. 1.

Lemma 3.1. *[12] Let $G = (V, E)$ be a connected undirected graph, which is not a cycle, and let $\mathcal{T}(G)$ be the corresponding biconnected component tree. Let $\mathcal{A}$ be an initial configuration on G and $\mathcal{A}_T$ the corresponding configuration on $\mathcal{T}(G)$. Let $a, b \in V$. Then, if $|H| \geq 2$, there is a plan f_{ab} such that $\mathcal{A}[a, b] = \rho(\mathcal{A}, f_{ab})$ if and only if there is a plan $f'_{a'b'}$ such that $\mathcal{A}_T[a', b'] = \rho(\mathcal{A}_T, f'_{a'b'})$.*

As a consequence of this Lemma, it follows that:

Theorem 3.2. *[12] Let G be a connected undirected graph which is not a cycle, with at least two holes. Then, MAPF on graph G is feasible if and only if ts-PMT on tree $\mathcal{T}(G)$ is feasible.*

Since feasibility of PMT on a tree $T = (V_T, E_T)$ is decidable in $O(|V_T|)$ time (see [6]), it follows that also MAPF on an undirected graph, which is not a cycle and with at least two holes, is decidable in linear time with respect to the number of nodes.

Moreover, any MAPF instance on a cycle is feasible if and only if in the final configuration the pebbles are in the same order as in the initial one [10].

In the case of graphs with only one hole, pebbles cannot move between different biconnected components (see [12]). Therefore, a MAPF problem on a graph with one hole is feasible only if the final position of each pebble is in the same biconnected component as the initial position. Moreover, the feasibility on a biconnected graph with one hole is decidable in linear time (see Remark 8.8 of [12]).

In conclusion, for *any* undirected graph $G = (V_G, E_G)$ it holds that:

Theorem 3.3. *[12] The feasibility of a MAPF instance on G is decidable in $O(|V_G|)$ time.*

3.2. Summary of algorithm presented in [16] to solve MAPF on graphs

The algorithm presented in [16] to solve MAPF problem on a graph has the following structure:

1. Convert G into the biconnected component tree $\mathcal{T}(G)$ and convert MAPF into ts-PMT;
2. Solve ts-PMT problem on $\mathcal{T}(G)$;
3. Convert the solution of ts-PMT on $\mathcal{T}(G)$ into solutions of MAPF on G.

In particular, [16] presents a function *CONVERT-PATH* that converts the solutions of ts-PMT on $\mathcal{T}(G)$ into solutions of MAPF on G. When a pebble moves from v to u via a trans-shipment vertex s, this function first checks if there is a pebble-free path between v and u on G. If there is, this movement can be achieved. Otherwise, a more complex process is implied by the feasibility algorithm for graphs, presented more in detail in [14].

The algorithm that we will present in Section 6 to solve MAPF problem on strongly connected digraphs has the same structure as the one just described for undirected graphs. The only difference consists in the *CONVERT-PATH* function: while the one presented in [16] converts a plan on a tree into a plan on the undirected graph, we will study a function that provides a plan on the strongly connected digraph.

4. Strongly connected digraphs

As said, we consider MAPF for *strongly connected digraphs*. In this section we define this type of directed graph and we go into the detail of their structure. This will be useful to present an algorithm to solve MAPF.

Definition 4.1. A digraph $D = (V, E)$ is *strongly connected* if for each $v, w \in V$, $v \neq w$, there exist a directed path from v to w, and a directed path from w to v in D.

Given a digraph D, we indicate with $\mathcal{G}(D)$ its *underlying graph*, that is the undirected graph obtained by ignoring the orientations of the edges, and $\mathcal{T}(\mathcal{G}(D))$ the corresponding biconnected component tree. Note that D is strongly connected only if $\mathcal{G}(D)$ is connected (see Figs. 1 and 2).

An important property of strongly connected digraphs is that they can be decomposed in strongly biconnected components.

Definition 4.2. A digraph D is said to be strongly biconnected if D is strongly connected and $\mathcal{G}(D)$ is biconnected.

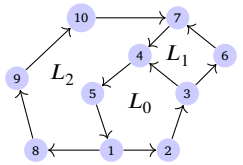


Fig. 3. Digraph with an open ear decomposition.

We recall that an undirected graph G is biconnected if it is connected and there are no cut vertices, i.e., the graph remains connected after removing any single vertex. The *partially-bidirectional cycle* is a simple example of a strongly biconnected digraph:

Definition 4.3. 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).

Reference [28] shows that strongly biconnected (respectively, strongly connected) digraphs have an open (respectively, closed) ear decompositions. We recall the definitions of open and closed ear decompositions. Given a graph $D = (V_D, E_D)$ and a sub-digraph $M = (V_M, E_M)$, a path π in D is a M-path if it is such that its startpoint and its endpoint are in V_M, no internal vertex is in V_M, and no edge of the path is in E_M. Moreover, a cycle C in D is a M-cycle if there is exactly one vertex of C in V_M.

Definition 4.4. Let $D = (V_D, E_D)$ be a digraph and $L = [L_0, L_1, \dots, L_r]$ an ordered sequence of sub-digraphs of D, where $L_i = (V_{L_i}, E_{L_i})$. We say that L is:

1. a *closed ear decomposition*, if:
   - L_0 is a cycle,
   - for all $0 < i \leq r$, L_i is a D_i-path or a D_i-cycle, where $D_i = (V_{D_i}, E_{D_i})$ with $V_{D_i} = \bigcup_{0 \leq j < i} V_{L_j}$ and $E_{D_i} = \bigcup_{0 \leq j < i} E_{L_j}$,
   - $V_D = \bigcup_{0 \leq j \leq r} V_{L_j}$, $E_D = \bigcup_{0 \leq j \leq r} E_{L_j}$
2. an *open ear decomposition* (*oed*), if it is a closed ear decomposition such that for all $0 < i \leq r$, L_i is a D_i-path, (i.e., it is not a D_i-cycle).

In Definition 4.4, each L_i is called an *ear*. In particular, L_0 is the basic cycle and the other ears are derived ears. An ear is *trivial* if it has only one edge.

Definition 4.5. We say that an open ear decomposition of a strongly biconnected digraph is *regular* (*r-oed*) 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.

Observation 4.6. Let $D = (V, E)$ be a digraph with an *oed* $L = [L_0, L_1, \dots, L_n]$. For each pair $v, w \in V$, there exists a sequence of cycles $C = [C_1, \dots, C_n]$ such that:

- $v \in V_{C_1}$ and $w \in V_{C_n}$;
- for all $j = 1, \dots, n-1$, $\exists a_j, b_j \in V_{C_j} \cap V_{C_{j+1}}$ such that $(a_j, b_j) \in E$.

Proof. Let $\pi = u_1 = v, u_2, \dots, u_{n-1}, u_n = w$ be a shortest path from v to w. Let L_i be an ear such that $v, u_2 \in V_{L_i}$. Let n_1 and m_1 be the startpoint and endpoint of L_i. Then, there exists a path π_1 from m_1 to n_1 and $C_1 = \pi_1 \cup L_i$ is the first cycle of the sequence. We initialize $C = [C_1]$ and we set $k = 1$. Now, if $n > 2$, for $j = 3, \dots, n$:

- if $u_j \in V_{C_k}$ we go to next iteration;
- otherwise, let π_{j-1} be a path from u_j to u_{j-2}, $C_{k+1} = (V_{C_{k+1}}, E_{C_{k+1}}) = \pi_{j-1} \cup (u_{j-2}, u_{j-1}) \cup (u_{j-1}, u_j)$ (note that $u_{j-2}, u_{j-1} \in V_{C_k} \cap V_{C_{k+1}}$); we add C_{k+1} to C and set $k = k + 1$, then we go to the next iteration. □

Fig. 3 shows a digraph with an *oed* $[L_0, L_1, L_2]$. The sequence of cycles associated to pair $v = 2$, $w = 10$ is $C = [C_0, C_2]$, where $C_0 = L_0$ and C_2 is the subgraph induced by $\{1, 8, 9, 10, 7, 4, 5\}$. Note that $(4, 5) \in C_0 \cap C_2$. The sequence associated to pair $v = 1$, $w = 6$ is simply $C = [C_1]$, where C_1 is the subgraph induced by $\{1, 2, 3, 6, 7, 4, 5\}$. In fact, nodes 1 and 6 belong to the same cycle.

We recall the following results, that characterize strongly biconnected and strongly connected digraphs:

Theorem 4.7. *Let D be a non-trivial digraph.*

- *D is strongly biconnected if and only if D has an oed. Any cycle can be the starting point of an oed [27].*
- *D is strongly biconnected if and only if exactly one of the following holds [10]:*

1. *D is a partially-bidirectional cycle;*
2. *D has a r-oed.*

Theorem 4.8. *[8] Let D be a non-trivial digraph. D is strongly connected if and only if D has a closed ear decomposition.*

Observation 4.9. Roughly speaking, this last result means that a strongly connected digraph is composed of non-trivial strongly biconnected components connected by corridors, or articulation points. A *corridor* on a digraph $D = (V, E)$ is a sequence of adjacent vertices $u_1, \dots, u_n$ such that

- $(u_i, u_{i+1}), (u_{i+1}, u_i) \in E$ for each $i = 1, \dots, n-1$;
- all the vertices of the sequence except u_1 and u_n (called *endpoints*) have exactly two neighbors.

For example, in Fig. 2 the subgraph induced by nodes 3, 5 and 6 is a corridor. We indicate with $\mathscr{C}(D)$ the set of all corridors of D. Vertex $v \in V$ is an *articulation point* if its removal increases the number of connected components of the underlying graph $\mathcal{G}(D)$. In Fig. 2 nodes 3, 6 and 11 are articulation points. We define the subclass of corridors $\bar{\mathscr{C}}(D) \subset \mathscr{C}(D)$ that have only articulation points as endpoints.

Note that each star subgraph of $\mathcal{T}(\mathcal{G}(D))$ represents a biconnected component of $\mathcal{G}(D)$, which corresponds to a strongly biconnected component of D. Indeed, Theorem 9 of [28] defines a one-to-one correspondence between strongly biconnected components of D and biconnected components of $\mathcal{G}(D)$.

Finally, we recall the following definition about strongly biconnected digraphs adapted from [12], that will be useful in Section 6:

Definition 4.10. Let $B = (V, E)$ be a strongly biconnected digraph and $v \notin V$ be an external node. We consider a digraph $G = (V \cup \{v\}, \bar{E})$ with $E \subset \bar{E}$. We say that G is:

- *a strongly biconnected digraph with an entry-attached edge*, if there exists $z \in V$ such that $\bar{E} = \{(v, z)\} \cup E$;
- *a strongly biconnected digraph with an attached edge*, if there exists $z \in V$ such that $\bar{E} = \{(v, z), (z, v)\} \cup E$.

5. Feasibility of MAPF on strongly connected digraphs

In this section, we focus on feasibility of MAPF problem on strongly connected digraphs.

As shown in Proposition 2 and 3 of [20], in strongly connected digraphs each move is reversible. From this, a more general result follows:

Proposition 5.1. *In a strongly connected digraph each plan has a reverse plan.*

Proposition 5.1 leads to the following result about the feasibility of MAPF on digraphs:

Theorem 5.2. *Let $D = (V_D, E_D)$ be a strongly connected digraph. Then,*

1. *any MAPF instance on D is feasible if and only if it is feasible on the underlying graph $G = \mathcal{G}(D)$;*
2. *feasibility of any MAPF instance on D is decidable in linear time with respect to $|V_D|$.*

Proof.
1. The necessity is obvious. To prove sufficiency, let f' be a plan which solves a MAPF instance on $\mathcal{G}(D)$. Then we can define a plan f on D in the following way. For each pebble move $u \to v$ in f', if $(u, v) \in E_D$, we perform move $u \to v$ on D. Otherwise, since $(v, u) \in E_D$, we execute a reverse plan for $v \to u$, $(v \to u)^{-1}$, that exists by Proposition 5.1.
2. It follows from Theorem 3.3. □

A direct consequence of Theorem 5.2 and Theorem 3.2 is the following important result:

Corollary 5.3. *Let D be a strongly connected digraph with at least two holes and such that the corresponding underlying graph $\mathcal{G}(D)$ is not a cycle. Then, MAPF on D is feasible if and only if ts-PMT on tree $\mathcal{T}(\mathcal{G}(D))$ (i.e., the biconnected component tree of $\mathcal{G}(D)$) is feasible.*

The proof of Theorem 5.2 leverages the reversibility of each pebble motion in strongly connected digraphs. It presents a simple algorithm that reduces MAPF for strongly connected digraphs to the case of undirected graphs. However, this approach leads to redundant solutions, since it does not exploit the directed graph structure. This fact is illustrated in Fig. 4, that shows a digraph D and its associated underlying graph $\mathcal{G}(D)$.

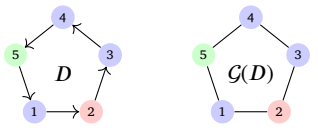


Fig. 4. A digraph D and its underlying graph $\mathcal{G}(D)$.

Example 5.4. A pebble p is placed at node 2, while all other nodes are free. We want to move p to 5. Plan $f' = (2 \to 1)(1 \to 5)$ is a solution of the corresponding problem on $\mathcal{G}(D)$. We convert this to a plan on D by applying the method in Theorem 5.2. Since $(2,1) \notin E_D$, move $(2 \to 1)$ is converted into plan $(2,3)(3,4)(4,5)(5,1)$. Similarly, move $(1 \to 5)$ is converted into $(1,2)(2,3)(3,4)(4,5)$. This solution is longer than necessary, since the shorter plan $f = (2 \to 3)(3 \to 4)(4 \to 5)$ solves the overall problem.

Definition 5.5. We say that a MAPF solution algorithm has *length complexity* $O(n^k)$ if there exists a positive real constant $\gamma \in \mathbb{R}$ such that, for any MAPF instance on a graph with n nodes, the algorithm is able to provide a solution plan f of length $|f| \leq \gamma \cdot n^k$.

Given a strongly directed graph $D = (V, E)$, and denoting by $\mathcal{C}(D)$ the set of all the cycles contained in D, we define

$$N = \max_{e \in E} \quad \min\{|C| : C \in \mathcal{C}(D),\, e \in E_C\}. \tag{2}$$

Proposition 5.6. *The length complexity of the algorithm described in Theorem 5.2 is* $O(n^3 N^2)$.

Proof. On undirected graphs, Kornhauser ([15]) proposes a MAPF algorithm that computes solutions of length γn^3, where $\gamma \in \mathbb{R}$ is a positive real constant. In the worst case, each pebble move $u \to v$ on the undirected graph must be converted into a reverse plan (as in Example 5.4). A reverse plan consists of a complete rotation of a cycle, which takes $O(n_c^2)$ moves, where n_c is the number of nodes of the cycle that contains arc $u \to v$. Each edge $e_i \in E$, crossed by the pebbles in the solution, belongs to a collection of cycles $\{C_i^j\}_j$ of the graph, among which we can choose the one with minimum length $n_i \leq N$, denoted with $C_i = \arg\min\{|C| : C \in \mathcal{C}(D),\, e_i \in E_C\}$. Let $[C_1, C_2, ..., C_m]$ be the sequence of cycles of minimum length, chosen for each edge, and $(n_1, n_2, ..., n_m)$ the corresponding number of nodes. Then, the overall solution length is bounded by

$$\gamma\, n^3 \cdot \sum_{i=1}^{m} O(n_i^2) \leq \gamma\, n^3 N^2. \quad \square$$

5.1. Necessary and sufficient condition for feasibility

Let us consider a strongly connected digraph $D = (V, E)$ and the corresponding biconnected component tree $T = (V_T, E_T)$ with $V_T = V \cup V^t$. In this section we want to derive a necessary and sufficient condition (n.s.c.) for feasibility of MAPF on D from the n.s.c. of the ts-PMT on the corresponding tree T.

First of all, we remind the n.s.c. for ts-PMT. To do that, on T we define a vertex-distance $\tilde{d}$ which does not take into account trans-shipment vertices. Given $u, v \in V_T$, $\tilde{d}(u, v)$ counts how many regular vertices belong to the path π_{uv}:

$$\tilde{d}_T(u, v) := |\pi_{uv} \cap V|.$$

Consequently, we also define $\tilde{c}_1$ and $\tilde{c}_2$ which count corridor lengths according to the new definition of distance:

$$\tilde{c}_T^1 := \max\{\tilde{d}(a, b) :\ \pi_{ab} \in \mathscr{C}(T)\},$$

$$\tilde{c}_T^2 := \max\{\tilde{d}(a, b) :\ \pi_{ab} \in \bar{\mathscr{C}}(T)\}.$$

Moreover, we define $\tilde{c}_T := \max\{\tilde{c}_T^1, \tilde{c}_T^2 + 1\}$. In the same way we can define $\tilde{d}_D$, $\tilde{c}_D^1$, $\tilde{c}_D^2$ and $\tilde{c}_D$ on the digraph D.

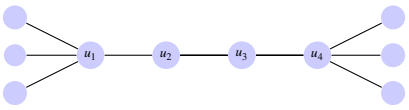
Kornhauser et al. [15] provided a n.s.c. for the feasibility of PMT on trees, from which Ardizzoni et al. [5] derived the following n.s.c. for the feasibility of ts-PMT:

$$|H| \geq |V^t| + \tilde{c}^T, \tag{3}$$

where we recall that H denotes the set of holes. Since establishing feasibility of MAPF on a digraph is equivalent to establishing the feasibility on the corresponding biconnected component tree (see Corollary 5.3), we can easily deduce that (3) is the n.s.c. for solvability of MAPF on a strongly connected digraph which is not a partially-bidirectional cycle and which has at least two holes. However, we want to translate (3) into a condition directly verifiable on the digraph.

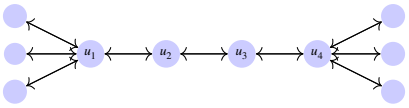
To do that, we have to find the relation between $\tilde{c}_T$ and $\tilde{c}_D$.

We consider a corridor $u_1, u_2 \ldots, u_{n-1}, u_n$ on the tree.

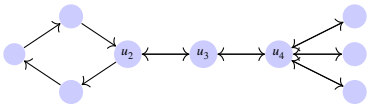


There are three possibilities:

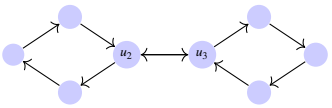
1. $u_1, u_n \notin V^t$: the corresponding corridor on D is exactly the same. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_1, u_n)$.



2. $u_1 \in V^t, u_n \notin V^t$ (or vice versa): the corresponding corridor on D is $u_2 \dots, u_{n-1}, u_n$. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_2, u_n)$.



3. $u_1, u_n \in V^t$: the corresponding corridor on D is $u_2, \dots, u_{n-1}$. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_2, u_{n-1})$.



Since the vertex-length of the corridors on the digraph is equal to the vertex-length of the corresponding corridors on the tree, it follows that $\tilde{c}_D^1 = \tilde{c}_T^1$, $\tilde{c}_D^2 = \tilde{c}_T^2$ and $\tilde{c}_D = \tilde{c}_T$. Since the number of trans-shipment vertices is equal to the number of non trivial biconnected components (i.e., $|V^t| = K$) condition (3) is equivalent to

$$|H| \geq K + \tilde{c}_D. \tag{4}$$

6. Path planner for strongly connected digraph

The first possible approach to solve MAPF on strongly connected digraphs is described in Theorem 5.2. This algorithm solves MAPF on the corresponding underlying graph and then converts the solution into a solution on the original digraph. However, as discussed in Section 5, in some cases solutions found with this procedure may be too long.

To find shorter solutions in polynomial time, we present diSC algorithm, that better exploits the structure of the directed graph. In particular, we will exploit the fact that strongly connected digraphs can be decomposed into strongly biconnected components (see Observation 4.9).

In Section 5 we proved that MAPF on D is feasible if and only if ts-PMT on $\mathcal{T}(\mathcal{G}(D))$ is feasible. Therefore, as in the case of undirected graphs (see Section 3), we can reduce MAPF to ts-PMT, solve it, and then convert the solution on the tree into a solution on the original directed graph.

Thus, diSC strategy is the same presented in Section 3.2 to solve MAPF on undirected graphs. The main steps are the following ones:

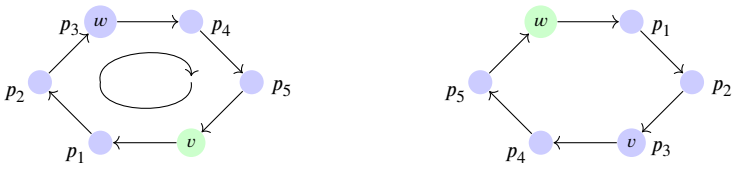
1. Convert the digraph D into a tree $T = \mathcal{T}(\mathcal{G}(D))$ and consider the corresponding ts-PMT problem.
2. Solve the ts-PMT problem on tree T.
3. Convert the solution plan on T into a plan on D by a suitable algorithm *CONVERT-PATH*.

The novelty in diSC algorithm lies in the definition of algorithm *CONVERT-PATH*, which allows converting a plan f', that solves the ts-PMT, into a plan f, that solves the MAPF on the digraph. In Theorem 6.8, we will describe in detail algorithm *CONVERT-PATH*.

6.1. Basic plans for CONVERT-PATH algorithm

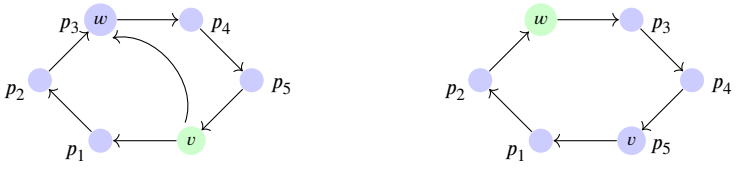
Algorithm *CONVERT-PATH* uses a number of basic plans, that perform simple tasks:

1. k-**Cycle Rotation**: this plan rotates all pebbles on a cycle, moving each one by k positions (Fig. 5). In particular, if $k = 1$ each pebble moves forward by one position. If $k = n$ (where n is the cycle length) it performs a *complete* rotation, so that each agent returns to its initial position. We denote this plan with r_k^C.



(a) Initial configuration. (b) Final configuration.

Fig. 5. Example of 3-CYCLE ROTATION.



(a) Initial configuration. (b) Final configuration.

Fig. 6. Example of BRING HOLE FROM v TO w.

2. **BRING HOLE FROM v TO w**: holes are needed to execute any move or plan (Fig. 6). For example, a k-**CYCLE ROTATION** is not possible unless there is at least one hole in the cycle. For this reason, it is useful to define this plan, which allows moving a hole from one position of the graph to another. Plan **BRING BACK HOLE FROM w TO v** is a reverse of BRING HOLE FROM v TO w. Hence, it returns the hole to its initial position, and moves back all agents to their initial positions. We denote these plans with $h_{v,w}$ and $h_{v,w}^{-1}$.

We formally define the plans just described in the Appendix.

Observation 6.1. Let $\mathcal{A}$ be an initial configuration, $h_{v,w}$ a plan BRING HOLE FROM v TO w, and $\bar{\mathcal{A}} = \rho(\mathcal{A}, h_{v,w})$ the corresponding final configuration. Given $a, b \in V$ with $b \neq v$, $p = \mathcal{A}^{-1}(a)$ and $q = \mathcal{A}^{-1}(b)$ are the pebbles or holes that occupy a, b. Then, $\bar{\mathcal{A}}[\bar{\mathcal{A}}(p), \bar{\mathcal{A}}(q)] = \overline{\mathcal{A}[a, b]}$. That is, the configuration obtained performing $h_{v,w}$ on $\mathcal{A}[a, b]$ is equal to the one obtained by exchanging $\bar{\mathcal{A}}(p)$ and $\bar{\mathcal{A}}(q)$ on $\bar{\mathcal{A}}$.

6.2. Plans for movements through biconnected components

As said, *CONVERT-PATH* algorithm converts a PMT solution into a MAPF solution. The most complex subtask is the conversion of a movement in a 'star' into a sequence of movements within the associated biconnected component. In particular, each pebble motion to a different spike of the same star requires crossing the star transshipment vertex. We need to convert this motion into a plan on the digraph that allows moving the pebble to the same final position, without altering the positions of all other pebbles.

Observation 6.2. In Observation 4.9 we noted that a strongly connected digraph $D = (V, E)$ is composed of non-trivial biconnected components and corridors. Following the classification presented in [12] for undirected graphs, in strongly connected directed graphs we defined three different motion tasks. Following [12], for each task we present one or more Lemmas, that describe a possible plan:

1. a pebble moves within the same strongly biconnected component: *Stay in Lemma* 6.6;
2. a pebble moves from a corridor to a strongly biconnected component (or viceversa): *Entry Lemma* 6.3, *Go out Lemma* 6.4, *Attached-Edge Lemma* 6.5;
3. a pebble moves from a strongly biconnected component to another one, connected by an articulation point: *Two Biconnected Components Lemma* 6.7.

All these Lemmas define plans that require two holes, with the following roles:

- the *destination* hole h_1 is in the final position of pebble p. In the final configuration, h_1 and p will be exchanged;
- the *transport* hole h_2 is used to move p to the position of h_1. Indeed, each pebble motion requires the presence of an hole. Each plan, at the end, brings back h_2 to its initial position, together with all pebbles with the exception of p.

Lemma 6.3 *(Entry). Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $G = (V \cup \{v\}, \bar{E})$, where G is a strongly biconnected digraph with an entry-attached edge (v, y) (see Definition 4.10). Let $\mathcal{A}$ be a configuration, $p \in P$ such that $\mathcal{A}(p) = v$, and $w \in \mathcal{A}(H)$. Let $\mathcal{A}[v, w]$ be the configuration defined in (1). Then, there exists a plan f_{vw} such that $\mathcal{A}[v, w] = \rho(\mathcal{A}, f_{vw})$, i.e., that moves p from v to w, without altering the locations of the other pebbles.*

Fig. 7 shows an example of this situation.

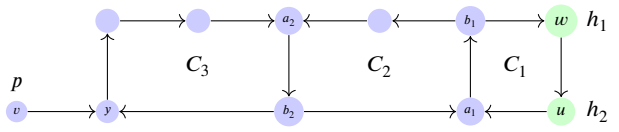
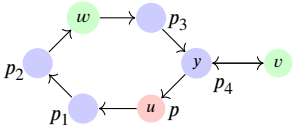
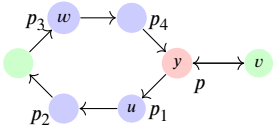


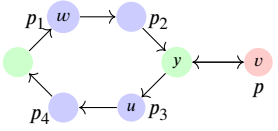
Fig. 7. An example of situation of Entry Lemma 6.3. Pebble p has to move on w without altering the final location of the other pebbles. h_1 and h_2 are, respectively, the destination and transport holes. The solution plan is $f_{vw} = r_2^{C_1} r_3^{C_2} r_2^{C_3} (v \to y) r_3^{C_3} r_2^{C_2} r_2^{C_1}$.



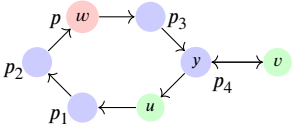
(a) Initial positions.



(b) $r_{d(u,y)}$.



(c) $(y \to v)\ r_k$.



(d) $(v \to y)\ r_{d(y,w)}$.

Fig. 8. An example of situation of Attached-edge Lemma 6.5. Pebble p has to move on w without altering the final location of the other pebbles. The solution plan is $f_{vw} = r_{d(u,y)}\,(y \to v)\, r_k\, (v \to y)\, r_{d(y,w)}$.

Lemma 6.4 *(Go out). Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $G = (V \cup \{w\}, \bar{E})$, where G is a strongly biconnected digraph with an attached edge (y, w) (see Definition 4.10). Let $\mathcal{A}$ be a configuration, $p \in P$ such that $\mathcal{A}(p) = v \in V$, and $w \in \mathcal{A}(H)$. Let $\mathcal{A}[v, w]$ be the configuration defined in (1). Then, there exists a plan f_{vw} such that $\mathcal{A}[v, w] = \rho(\mathcal{A}, f_{vw})$, i.e., that moves p from v to w, without altering the locations of the other pebbles.*

Lemma 6.5 *(Attached-Edge). Let P be a set of pebbles and H a set of holes on $D = (V \cup \{v\}, \bar{E})$, a strongly biconnected digraph with an attached edge such that $|H| \geq 2$. Let $\mathcal{A}$ be a configuration, $p \in P$ such that $\mathcal{A}(p) = u$, and $w \in \mathcal{A}(H)$. Let $\mathcal{A}[u, w]$ be a final configuration defined as in (1), then there exists a plan f_{uw} such that $\mathcal{A}[u, w] = \rho(\mathcal{A}, f_{uw})$.*

Fig. 8 shows an example of this situation.

Lemma 6.6 *(Stay in). Let P be a set of pebbles and H, with $|H| \geq 2$, be a set of holes on $D = (V, E)$, a strongly biconnected digraph with a r-oed. Let $\mathcal{A}$ be a configuration, $p \in P$ such that $\mathcal{A}(p) = v$, and $w \in \mathcal{A}(H)$. Let $\mathcal{A}[v, w]$ be a configuration defined as in (1), then there exists a plan f_{vw} such that $\mathcal{A}[v, w] = \rho(\mathcal{A}, f_{vw})$.*

Fig. 9 shows an example of this situation.

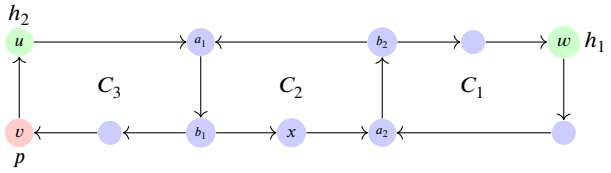
The next lemma deals with the case of two biconnected components joined by an articulation point (e.g., like $\{6, 7, 8, 9, 10, 11\}$ and $\{11, 12, 13\}$ in Fig. 2, where the articulation point is node 11).

Lemma 6.7 *(Two Biconnected Components). Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $D = (V, E)$, a strongly connected digraph, composed of two biconnected components joined by an articulation point. Let $\mathcal{A}$ be a configuration, $p \in P$ be such that $\mathcal{A}(p) = a$, and $b \in \mathcal{A}(H)$. Let $\mathcal{A}[a, b]$ be a final configuration defined as in (1). Then, there exists a plan f_{ab} such that $\mathcal{A}[a, b] = \rho(\mathcal{A}, f_{ab})$.*

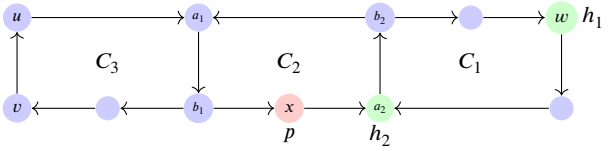
6.3. CONVERT-PATH algorithm

The following theorem shows that each plan on the biconnected components tree can be converted into a plan on the original digraph. The proof (which can be found in the Appendix) uses the Lemmas defined above to explicitly show how to convert a plan f' on the tree into a plan f on the digraph. This allows explicitly defining *CONVERT-PATH* algorithm.

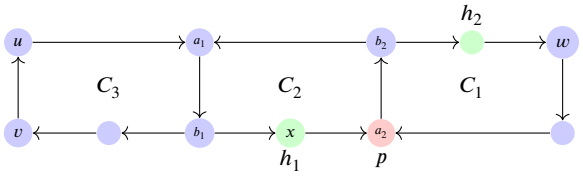
Theorem 6.8. *Let P be a set of pebbles and H a set of holes on a strongly connected digraph $D = (V, E)$, which is not a partially-bidirectional cycle, and let $T = (V_T, E_T)$ be the corresponding biconnected component tree. Let $\mathcal{A}$ be an initial configuration on D and $\tilde{\mathcal{A}}$ the corresponding configuration on T. Let $a, b \in V$ and $p \in P$ be a pebble on a. Then, if $|H| \geq 2$, there is a plan f_{ab} on D such that $\mathcal{A}[a, b] = \rho(\mathcal{A}, f_{ab})$ if and only if there is a plan f'_{ab} on T such that $\mathcal{A}_T[a, b] = \rho(\mathcal{A}_T, f'_{a'b'})$.*



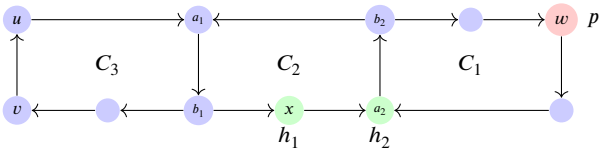
(a) Initial configuration.



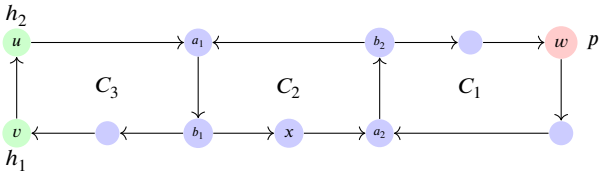
(b) New configuration after performing $r_2^{C_3} r_2^{C_2}$



(c) New configuration after performing $r_2^{C_1}(x \to a_2)$



(d) New configuration after performing $r_3^{C_1}$



(e) Final configuration after performing $r_3^{C_2} r_3^{C_3}$. Pebble p reached the target w and the other pebbles are in the same initial positions.

Fig. 9. An example of situation of Stay in Lemma 6.6. Pebble p has to move on w without altering the final location of the other pebbles. h_1 and h_2 are, respectively, the destination and transport holes. The solution plan is $f_{vw} = r_2^{C_3} r_2^{C_2} r_2^{C_1}(x \to a_2) r_3^{C_1} r_3^{C_2} r_3^{C_3}$.

7. Upper bound for solutions length and time complexity

In this section, we provide an upper bound for the length of MAPF solutions found by diSC Algorithm. For a MAPF instance on a graph G composed by K non-trivial biconnected components, this length depends on:

A. the solution length of the ts-PMT problem on the associated biconnected components tree,
B. the number of trans-shipment vertex crossings in the ts-PMT solution,
C. the length complexity of the conversion of a movement through a trans-shipment vertex on the tree into a plan on the digraph,
D. the length complexity of plan BRING HOLE FROM v TO w and the number of times this plan is used.

We can bound these quantities as follows.

A. In the literature, the best upper bound for PMT solutions is $O(|P|nc + n^2)$, where c the maximum length of the corridors of the tree (it is proved by Ardizzoni et al. [5]).

B. In the procedure described by Ardizzoni et al. [5], any vertex is crossed at most $O(|P|c)$ times by the pebbles in the overall solution (see Proposition 5.3 of reference [5]).

C. To calculate the cost of the conversion, first we have to compute the length of a k-CYCLE ROTATION. Let C_i a cycle with n_i nodes and $k \leq n_i$. Then a 1-CYCLE ROTATION takes $O(n_i)$, while a k-CYCLE ROTATION takes $O(k \cdot n_i)$ which, in the worst case, is $O(n_i^2)$. If $C = [C_1, ..., C_m]$ is an ordered sequence of cycles with n nodes, and $k = (k_1, \dots, k_m) \in \mathbb{N}^m$, the length complexity of the plan R_k^C, obtained by concatenating a k_1-CYCLE ROTATION over C_1, a k_2-CYCLE ROTATION over C_2, and analogous rotations over the remaining cycles of C, is given by

$$O\left(\sum_{i=1}^{m} k_i \cdot n_i\right)$$

Therefore, the complexity of a *complete* k-CYCLE ROTATION of C is $O(n^2)$. Note that also the plan BRING HOLE FROM v TO w has quadratic length, since it is a k-CYCLE ROTATION.

The plans described in *Stay in Lemma* 6.6, *Entry Lemma* 6.3, *Go out Lemma* 6.4, *Attached-Edge Lemma* 6.5 and *Two Biconnected Components Lemma* 6.7 are all combination of plans of the type k-CYCLE ROTATION and BRING HOLE FROM v TO w. Therefore, on a biconnected component with n_j nodes, they have $O(n_j^2)$ length.

D. To calculate the total cost of the function *CONVERT-PATH*, in addition to the cost of the plans presented in the lemmas, we must also consider how many times we use function BRING HOLE FROM v TO w. Indeed, in the Proof of Theorem 6.8, when a pebble has to move within a biconnected component but there are not enough holes to perform this movement, we use BRING HOLE FROM v TO w to add a hole. In the worst case, BRING HOLE FROM v TO w is used once for each traversal of a trans-shipment vertex. Since the complexity of this plan is $O(n^2)$ and from point B. we know that each vertex is crossed $O(|P|c)$ times, we conclude that the overall complexity due to BRING HOLE FROM v TO w is $O(|P|n^2c)$.

If graph G is composed by K non-trivial biconnected components, the cost of the conversion performed by function *CONVERT-PATH* (see the proof of Theorem 6.8 in the Appendix) is

$$\underbrace{O(|P|n^2c)}_{D} + \sum_{i=1}^{K}\left(\underbrace{O(|P|c)}_{B} \cdot \underbrace{O(n_i^2)}_{C}\right) =$$

$$O(|P|n^2c) + O(|P|c) \cdot O(n^2 + K) = O(|P|n^2c),$$

where n_i is the number of nodes of the i-th biconnected component, and $\sum_{i=1}^{K} n_i \leq n + K$, since some nodes can belong to more than one biconnected component.

Since the length complexity of PMT solution is $O(|P|nc + n^2)$ (see point A.), the overall complexity is dominated by the conversion cost and the following results hold:

Theorem 7.1. *The length complexity of diSC algorithm is* $O(|P|n^2c)$.

Note that with diSC Algorithm we find a better complexity result than the one obtained through the method of Theorem 5.2 (see Proposition 5.6).

Remark 7.2. Note that the choice between the diSC algorithm and the procedure described in Theorem 5.2 depends on the structure of the digraph and on the number of pebbles. If cycles of the graph have small length, the latter procedure might be more convenient. For example, if $N = 3$ the complexity is $O(9n^3) = O(n^3)$. Instead, if the corridors have a delimited length with respect to the dimensions of the graph (for example $c = 6$), the diSC algorithm is usually preferable. In any case, we note that in the worst case diSC has a complexity of $O(n^4)$, while the other procedure has complexity $O(n^5)$ in the worst case.

8. Experimental results

We implemented diSC algorithm in Matlab. We tested it on random graphs as well as on real-life scenarios. We used a *Intel(R) Core(TM) i7-4510U CPU @ 2.60 GHz* processor with 16 GB of RAM. Note that we did not employ standard benchmark graphs used for MAPF (see, e.g., [25]), such as graphs from the MovingAI library. Indeed, while these graphs may vary in shape, they are usually grid-based, undirected and biconnected graphs (see, e.g., DAO maps and Open $N \times N$ grids). As a result, they are not appropriate for testing diSC, an algorithm specifically designed for directed, strongly connected, but not biconnected graphs (for biconnected graphs, the algorithm presented in [10] is more suitable).

8.1. Random graphs

We generated random graphs in two different ways.

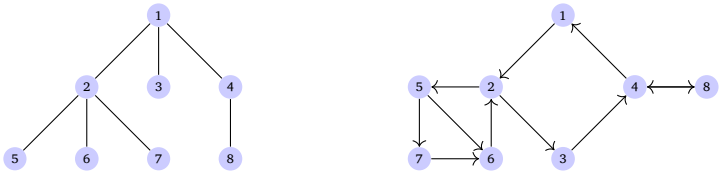


Fig. 10. From tree to strongly connected digraph.

Random graphs based on trees.

In order to generate graphs with multiple biconnected components, we used the following procedure, described also in [2]. First, we create a random connected undirected graph with function *networkx.connected_watts_strogatz_graph()*, of NetworkX library.[1] Then, we construct a maximum spanning rooted tree (Fig. 10). We process the tree nodes with a breadth-first order. Every node that has a number n of children higher than 1 is converted into a biconnected component, together with its children, with the following method. We substitute the parent node and its children with a directed cycle with a random number of nodes lower or equal than $n+1$. Then, we add directed ears of random length (but sufficiently small, not to exceed the total number of $n+1$ nodes assigned to the biconnected component) and random initial and final nodes, until the number of nodes in the resulting biconnected component equals $n+1$. After processing the tree, every remaining undirected edge $\{u, v\}$ is converted into two directed edges (u, v), (v, u).

For every number of nodes, we generated a set of 100 graphs. We ran the algorithm varying both the number of nodes and the number of agents. For every generated graph (100 for every different number of nodes), we created MAPF problems instances with agents that vary from 1 to half of the number of nodes, and random initial and final positions. For example, we generated 100 graphs with 60 nodes, and for every graph we created one instance for every number of agents varying from 1 to 30, resulting in 3000 instances. For each obtained solution, we recorded the overall number of moves and the computation time.

Fig. 11 shows the solutions lengths of all the instances as a function of the number of agents on graphs with 20, 60 and 100 nodes. In red is shown the medians of the number of steps for every different number of agents. Roughly, the overall number of steps grows polynomially with respect to the number of agents, in line with Theorem 7.1. In Fig. 12, we can see the running times of the different instances with the corresponding medians shown in red. Similarly to the number of steps, the running time grows polynomially with respect to the number of agents.

Graphs generated by randomly added edges.

We also considered a second method for generating random graphs. We start from an empty graph, and randomly add edges until the resulting graph is strongly connected. This procedure gives us graphs that are more connected than the one generated before. With this procedure, we generated random graphs with 20, 60 and 100 nodes. As done before, for every number of agents and number of nodes, we created 100 MAPF problem instances. For each number of nodes the number of agents vary from 1 to half of the number of nodes in the first test, and from 1 to the highest number of agents possible for the second one. For each solution, we recorded the number of steps and the computation time.

Fig. 13 shows the number of steps of the obtained solutions, with medians highlighted in red. Also in this case the number of steps appears to increase polynomially with respect to the number of agents. By comparing this picture and Fig. 11, we can see how in the second case the number of steps is much lower. This can be explained by looking at the structure of the graphs. In the first case we have more corridors and more biconnected components, whereas in the second case the graph is more connected and the number of biconnected components is much lower. The lower number of biconnected components results in a lower number of steps in both the initial solution on the tree and the final solution on the strongly connected graph. Fig. 14 shows the running times of the solutions. The red line represents the medians of the running times, for each number of agents. Apparently, running times grow polynomially with respect to the number of agents.

In Fig. 15 the number of steps of instances with a high number of agents is shown, with medians highlighted in red. In particular, the maximum considered number of agents is $|P| = n-2$, where n is the number of nodes. Note that some instances with many agents can be infeasible. We removed such instances. Note that the growth of the solution length with respect to the number of agents seems to be polynomial in all the considered range. In Fig. 16, we can see the running time of these instances, with the medians shown in red. Also in this case, the running time seems to grow polynomially with respect to the number of agents.

8.2. Graph based on movingai MAPF benchmarks

We tested the algorithm on a benchmark for Multi-Agent Path Finding problems presented in [25].

Fig. 17 shows a grid map composed of 64 rooms, each consisting of $3x3$ cells, and interconnected by doors. We defined an undirected graph, where each white cell corresponds to a node, and each edge links nodes of adjacent cells. Then, we generated a directed graph by transforming each edge of the undirected graph into two opposite edges of the directed graph. Then, we randomly removed some directed edges, still guaranteeing that the resulting graph is strongly connected. The result is a strongly connected directed graph with 682 nodes and 870 edges. We created 100 instances for each number of agents from 5 to 35, with a step of 5. In these instances, we randomly selected the initial and final configurations of the agents.

[1] https://networkx.org/.

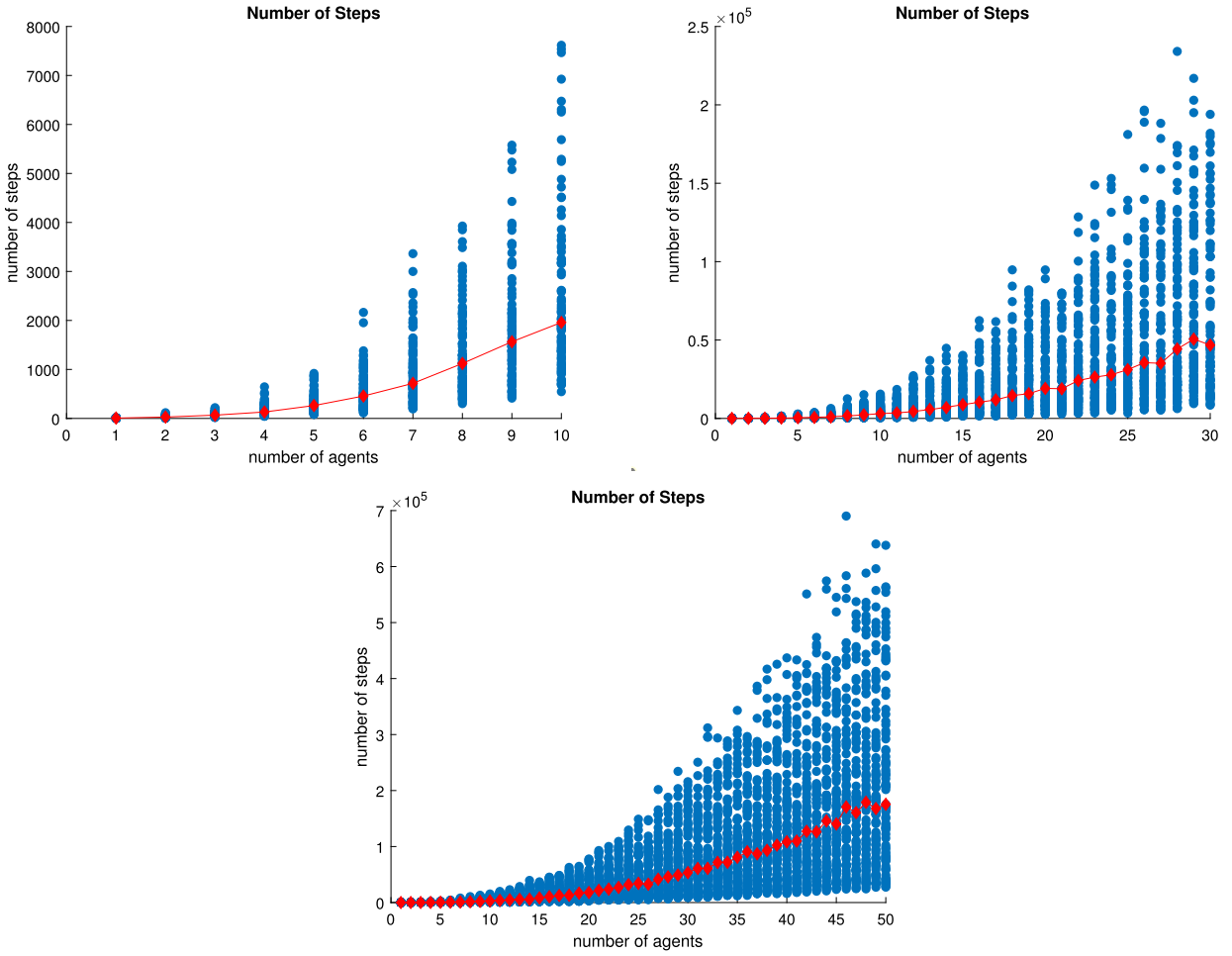


Fig. 11. Number of steps per number of agents of every instance in random graphs with 20, 60 and 100 nodes. The median of the number of steps of instances for each number of agents is shown in red. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Fig. 18 shows the number of steps and the computation times wrt the number of agents, with medians highlighted in red. Even in this case, both the number of steps and the computation time appear to grow polynomially wrt the number of agents.

8.3. Real-life applications examples

As a second example, we considered a real-life warehouse layout, provided by packaging company Ocme S.r.l., based in Parma, Italy. This layout is associated to a strongly connected digraph with 397 nodes (see Fig. 19). This graph consists of a biconnected component with 304 nodes and corridors connected to it. The corresponding biconnected component tree does not contain any corridors with endpoints that are only articulation points. Therefore, $\bar{C}$ is empty and $\bar{c}_T^2 = 0$. Additionally, since $\bar{c}_T^1 = 4$, it follows that $\bar{c}_T = 5$. We note that the necessary and sufficient condition (3) becomes $|H| \geq 6$, which is always satisfied in the simulations we performed.

We ran the algorithm varying the number of agents from 1 to 10. For each number of agents we generated 100 random MAPF instances, and we solved them with diSC algorithm. Fig. 20 shows the number of steps and the running time, with respect to the number of agents, with the corresponding medians highlighted in red. Both the number of steps and the running time seem to grow polynomially with respect to the number of agents.

Moreover, we generated graphs with the same structure of the warehouse of Fig. 19. In particular, we created graphs with the same number of nodes and the same nodes degrees as the warehouse. To do so we used the NetworkX[2] function *directed_configuration_model()* that creates a random graph with the specified in and out degrees.

[2] https://networkx.org/.

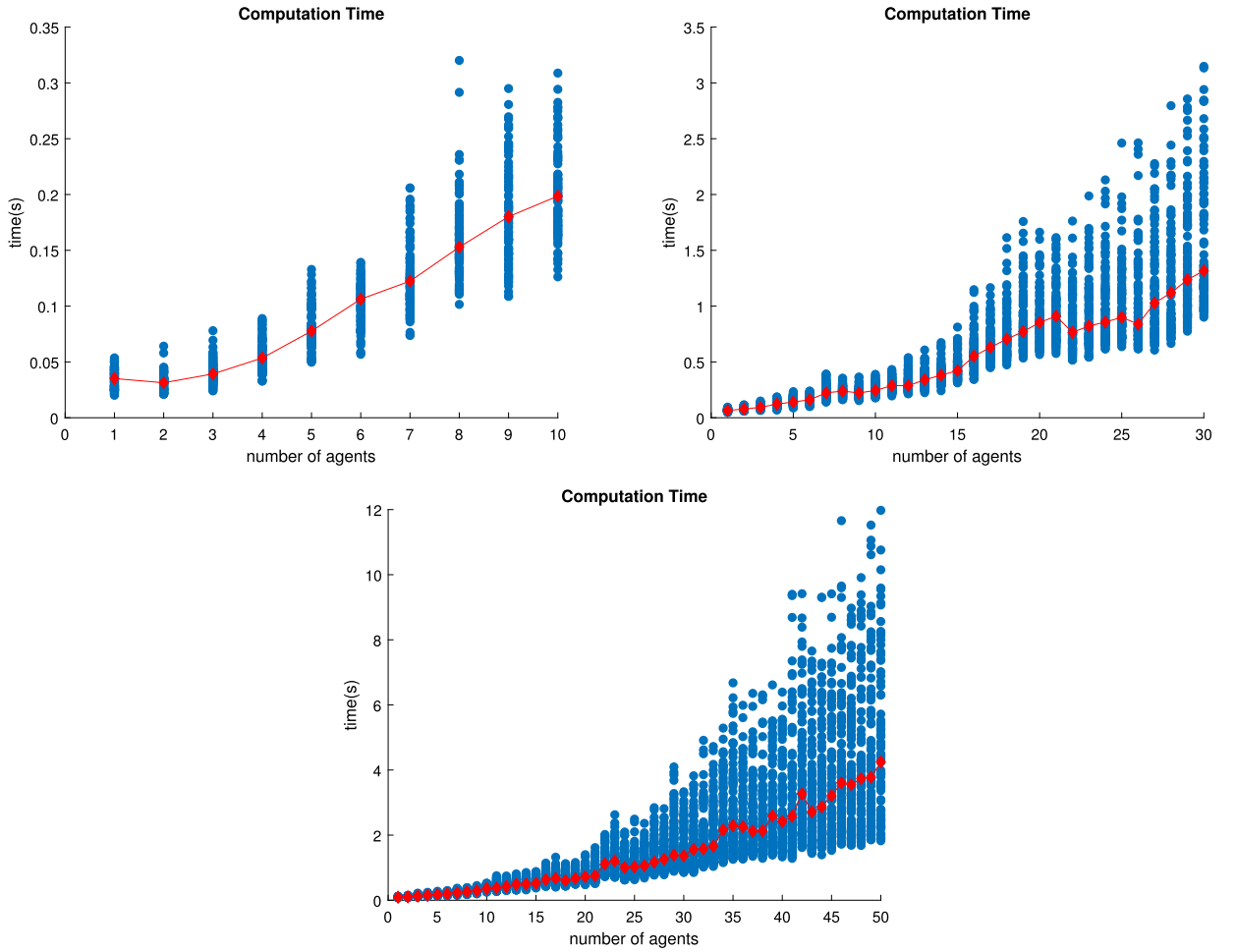


Fig. 12. Running time per number of agents on random graphs with 20, 60 and 100 nodes. The median of the running times of instances for each number of agents is shown in red.

We generated 100 random instances for each number of agents, varying from 1 to 10 agents. Fig. 21 shows the number of steps and the running time, with the corresponding medians highlighted in red. Both seem to grow polynomially with respect to the number of agents.

As demonstrated by these tests, the diSC algorithm proves effective in managing warehouse agents, particularly in scenarios involving congestion and deadlock risk. The results of the simulations show that agent paths can be computed in a time below one second, even in cases with a growing number of agents. This indicates that the algorithm is suitable for real-time applications, enabling frequent path replanning. However, the solution provided by diSC cannot be considered final, as it does not allow for simultaneous agent movements. Nevertheless, these solutions can serve as a starting point for local search algorithms that minimize agent movements and allow for simultaneous motion [4,18]. Furthermore, the algorithm does not take into account the physical dimensions of the agents, treating them as point-like entities. Therefore, it may be beneficial to complement it with other algorithms that consider the volume occupied by the agents, as well as additional constraints imposed by the warehouse structure (see, e.g., [3]).

9. Conclusions and future work

We proved that the feasibility of MAPF problems on strongly connected digraphs is decidable in linear time (Theorem 5.2). Moreover, we showed that a MAPF problem on a strongly connected digraph is feasible if and only if the corresponding PMT problem on the biconnected component tree is feasible (Corollary 5.3).

Then, we presented two approaches for finding the solution of a MAPF instance on a strongly connected digraph D. The first, described in the proof of Theorem 5.2, consists in reducing MAPF for strongly connected digraphs to the case of undirected graphs. This strategy is easy to implement, but leads to redundant solutions as shown in Example 5.4. Moreover, in the worst case the length complexity of the solutions is $O(n^5)$. The second approach consists in reducing MAPF on strongly connected digraphs to PMT (pebble

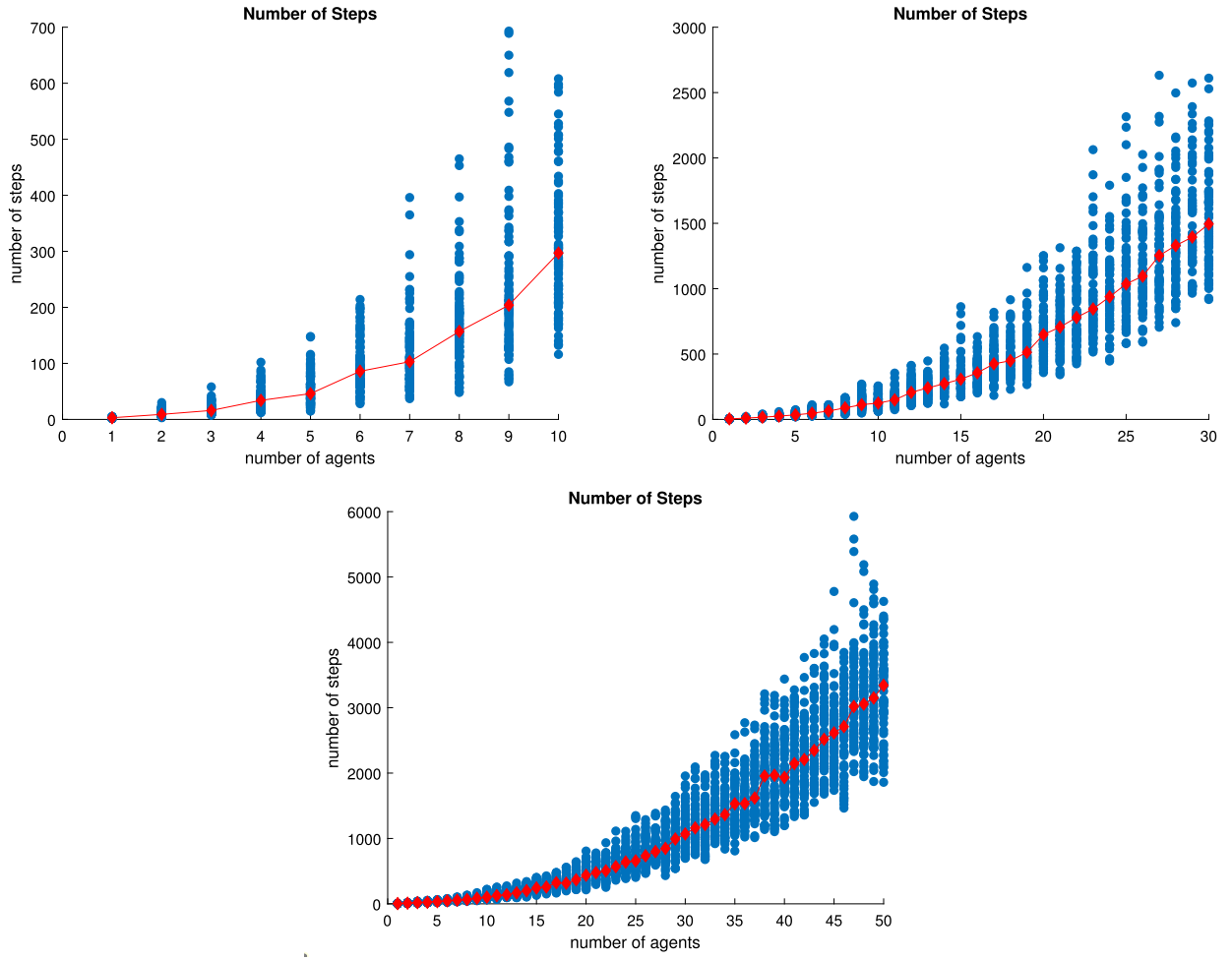


Fig. 13. Number of steps per number of agents of every instance in random graphs with 20, 60 and 100 nodes. The median of the number of steps of instances for each number of agents is shown in red.

motion on trees). In particular, we described an algorithm based on this strategy called diSC, which converts MAPF to the transshipment variant of PMT, and then converts back the obtained solution over the original graph. To convert the solution, diSC uses the *CONVERT-PATH* function, a procedure described in the proof Theorem 6.8. The overall length complexity of diSC algorithm is $O(|P|n^2c)$, where $|P|$ is the number of pebbles, n the number of nodes and c the maximum length of the corridors. In the worst case diSC has complexity of $O(n^4)$.

Finally, we implemented diSC in Matlab, and we presented some experimental results on random graphs and on graphs representing warehouse layouts. As already said, diSC algorithm finds solutions that have often a much larger number of steps than the shortest ones. This suggests that a possible topic for future research is to study strategies to shorten the solution.

CRediT authorship contribution statement

S. Ardizzoni: Writing – review & editing, Writing – original draft, Formal analysis, Conceptualization. **L. Consolini:** Writing – review & editing, Writing – original draft, Supervision. **M. Locatelli:** Writing – review & editing, Writing – original draft, Supervision. **B. Nebel:** Writing – review & editing, Writing – original draft, Supervision. **I. Saccani:** Writing – review & editing, Writing – original draft, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

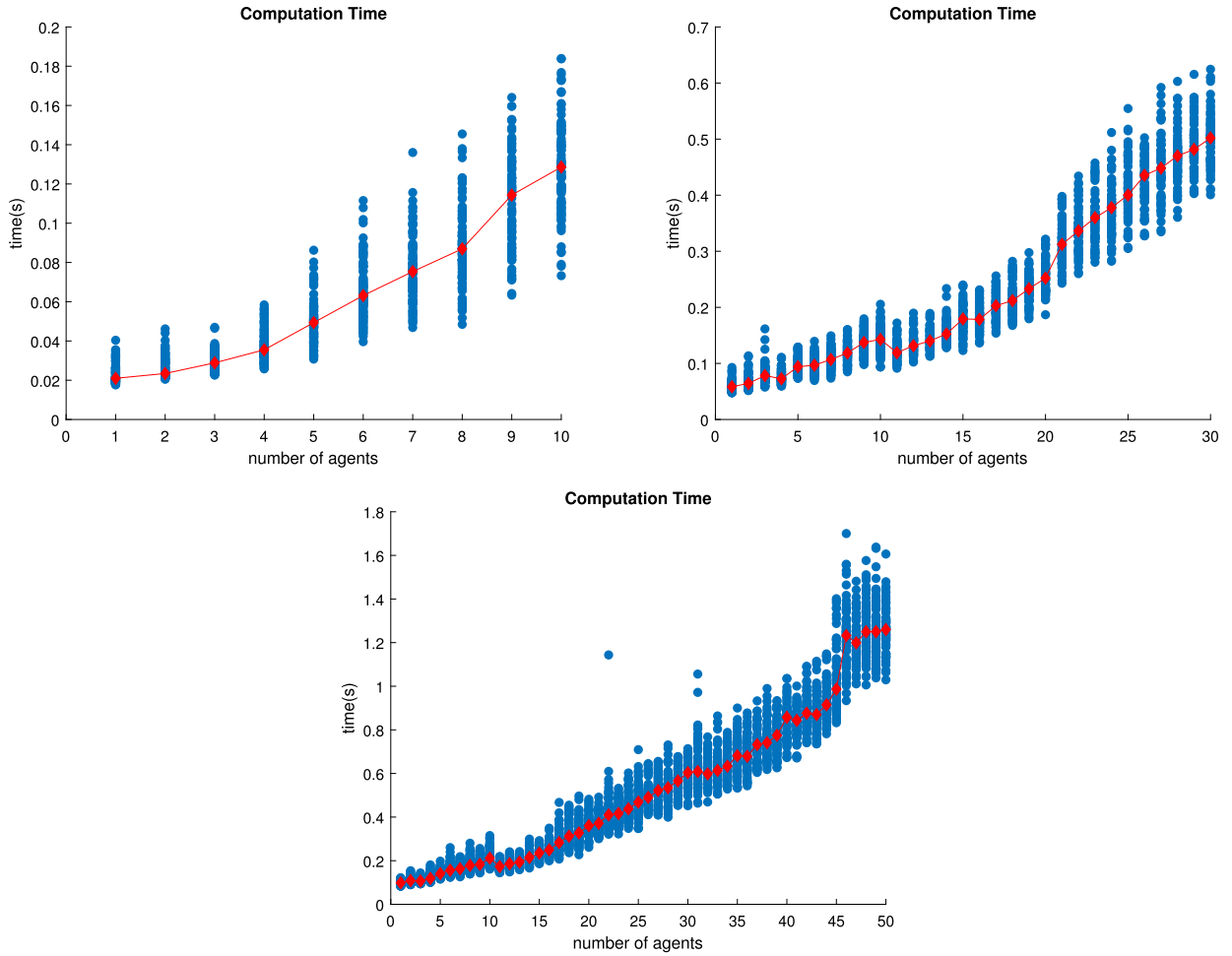


Fig. 14. Running time per number of agents of every instance in random graphs with 20, 60 and 100 nodes. The median of the running times of instances for each number of agents is shown in red.

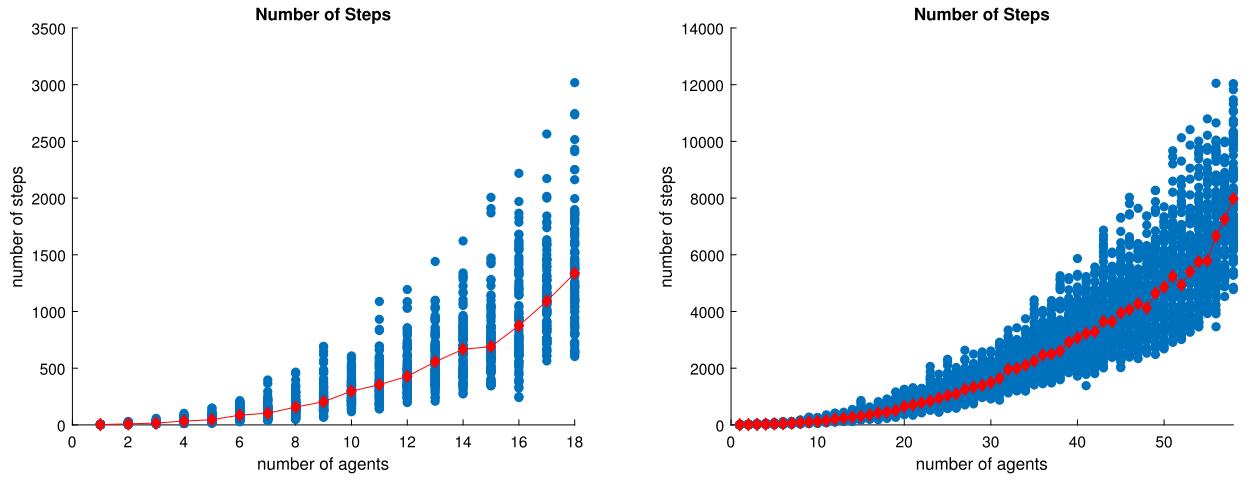


Fig. 15. Number of moves per number of agents on random graphs with 20 and 60 nodes for a high number of agents. The median of the number of steps of instances for each number of agents is shown in red.

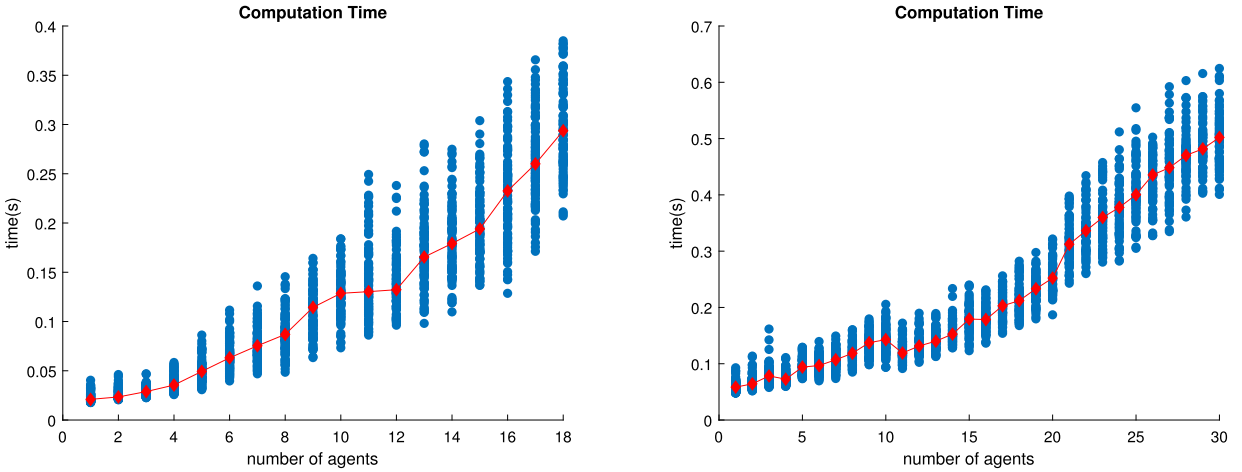


Fig. 16. Running time per number of agents on random graphs with 20 and 60 nodes for a high number of agents. The median of the running time of instances for each number of agents is shown in red.

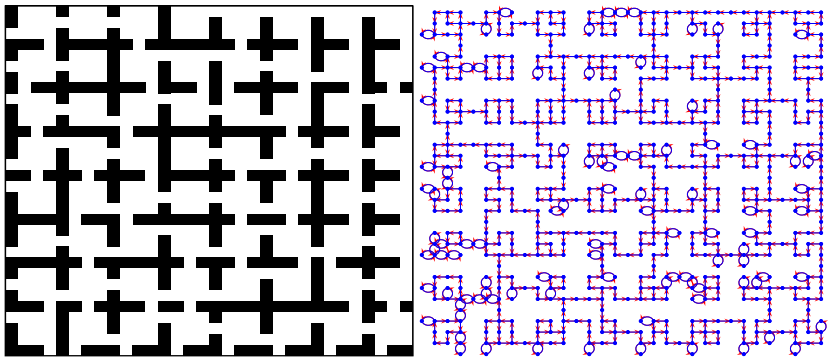


Fig. 17. Strongly connected digraph obtained from room graph of *movingai* MAPF benchmark.

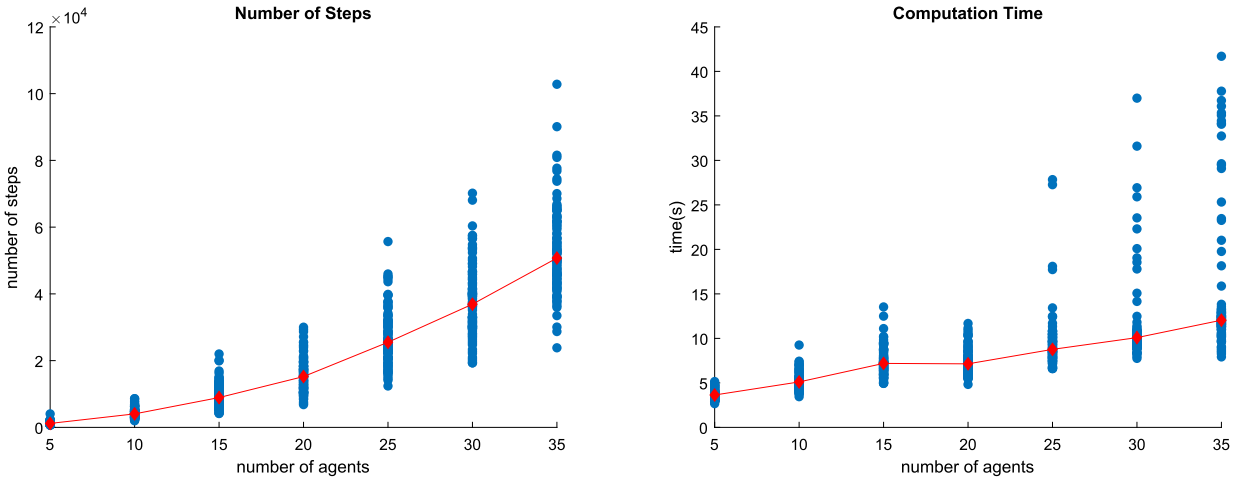


Fig. 18. Number of moves and running time per number of agents on graph in Fig. 17. The median of the number of steps and running time of instances for each number of agents is shown in red.

Fig. 19. Graph representing the warehouse.

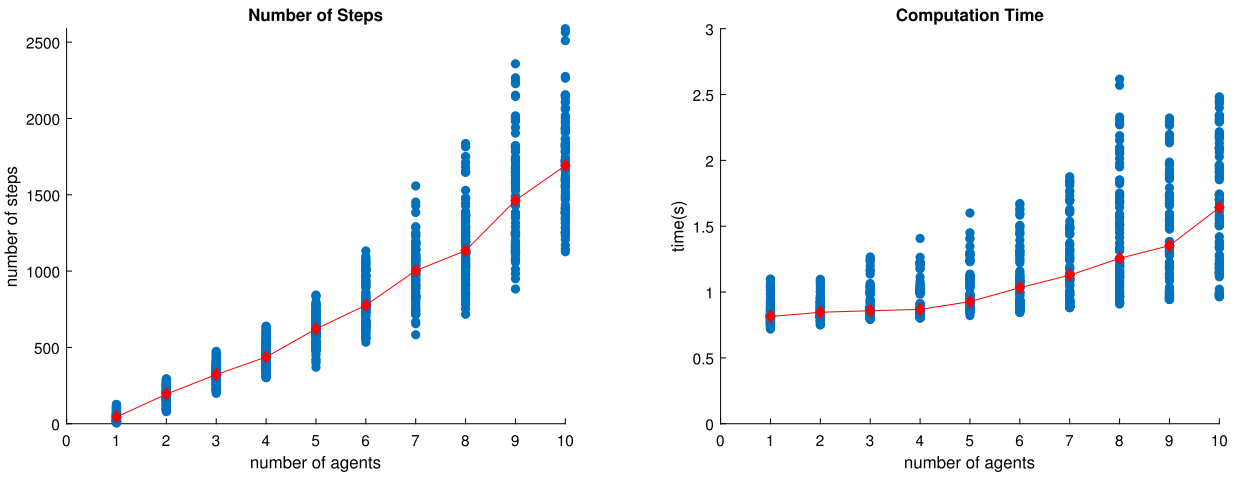


Fig. 20. Number of moves and computation times per number of agents in the warehouse of Fig. 19, with medians highlighted in red.

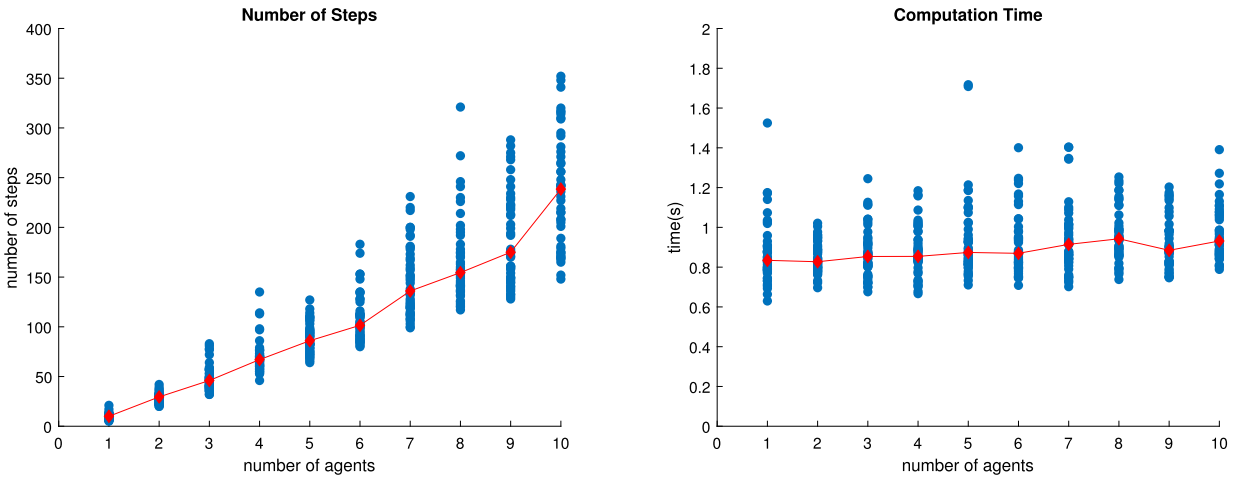


Fig. 21. Number of moves and computation times per number of agents in the graphs with same structure as Fig. 19.

Appendix A. Basic plans

In this Appendix we explain in more detail the basic plans.

k-Cycle Rotation. Let $C = (V_C, E_C)$ be a cycle, with a set of pebbles P, and a set of holes H, with $|H| \geq 1$. Let $\mathcal{A}$ be an initial configuration, $v \in \mathcal{A}(H)$, and $w \in V_C$ be such that $(v, w) \in E_C$ (i.e., w is the successor of v on C).

Let $\pi = u_1 = w, \dots, u_n = v$ be the path on C from w to v. We define plan 1-Cycle Rotation as

$$r_1^C = (u_{n-1} \to u_n, \dots, u_1 \to u_2). \tag{A.1}$$

In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. The new configuration $\bar{\mathcal{A}}$ is defined as follows:

$$\bar{\mathcal{A}}(q) := \begin{cases} u_{j+1}, & \text{if } \mathcal{A}(q) = u_j \quad j = 1, \dots, n-1; \\ w, & \text{if } \mathcal{A}(q) = v; \end{cases} \tag{A.2}$$

which means that all pebbles and holes on C change positions.

For $k \in \mathbb{N}$, a k-Cycle Rotation over C is obtained by performing k 1-Cycle Rotations over C. We denote the plan corresponding to a k-Cycle Rotation over C by r_k^C. Let $l_C = |V_C|$ be the length of cycle C. Plan $r_{l_c}^C$ is a *complete* rotation of C and brings all pebbles and holes back to their initial positions. In other words $r_{l_c}^C \sim \epsilon$, where ϵ is the empty plan. Since $\epsilon \sim r_{l_C}^C = r_k^C r_{l_C - k}^C$, it follows that complementary rotation $r_{l_C-k}^C$ is an inverse plan of r_k^C.

If $C = [C_1, \dots, C_m]$ is an ordered sequence of cycles, that are subgraphs of the same graph, and $k = (k_1, \dots, k_m) \in \mathbb{N}^n$, R_k^C denotes the plan obtained by concatenating a k_1-Cycle Rotation over C_1, a k_2-Cycle Rotation over C_2, and analogous rotations over the remaining cycles of C, namely:

$$R_k^C = r_{k_1}^{C_1} \dots r_{k_n}^{C_n}.$$

Set $s = (s_n, s_{n-1}, \dots, s_1) = (l_{C_n} - k_n, \dots, l_{C_1} - k_1)$ and $\hat{C} = [C_n, C_{n-1}, \dots, C_1]$. Then

$$R_k^C R_s^{\hat{C}} = r_{k_1}^{C_1} \dots r_{k_n}^{C_n} r_{s_n}^{C_n} \dots r_{s_1}^{C_1} \sim \epsilon,$$

since $r_{k_n}^{C_n} r_{s_n}^{C_n} \sim \epsilon$, and analogous reductions hold for the remaining terms. This implies that $R_s^{\hat{C}}$ is an inverse plan of R_k^C. We denote $R_s^{\hat{C}}$ by $(R_k^C)^+$. In other words, an inverse plan of R_k^C consists in a sequence of complementary rotations, in inverse order.

Bring hole from v to w. Let $\mathcal{A}$ be an initial configuration, such that $v \in \mathcal{A}(H)$ (i.e., v is an unoccupied vertex). There are two cases:

1. v and w belong to the same cycle C. In this case, let $\pi = u_1 = w, \dots, u_n = v$ be the path on C from w to v. We define the plan Bring hole from v to w as

   $$h_{v,w} = (u_{n-1} \to u_n, \dots, u_1 \to u_2). \tag{A.3}$$

   In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. The new configuration $\bar{\mathcal{A}}$ is defined as follows:

   $$\bar{\mathcal{A}}(q) := \begin{cases} u_{j+1}, & \text{if } \mathcal{A}(q) = u_j \quad j = 1, \dots, n-1; \\ w, & \text{if } \mathcal{A}(q) = v; \\ \mathcal{A}(q), & \text{otherwise}, \end{cases} \tag{A.4}$$

   which means that only pebbles and holes along path π change positions. Moreover, since the graph is strongly connected, by Proposition 5.1 there exists a reverse plan $h_{v,w}^{-1}$, which returns pebbles and holes to their initial positions. In particular in this case the reverse plan consists in moving the pebbles along the cycle until the initial configuration is obtained again. We call bring back hole from w to v the plan $h_{v,w}^{-1}$.
2. Otherwise, by Observation 4.6, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_n}]$ such that $v \in C_{i_1}$, $w \in C_{i_n}$, and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ contains at least one node a_{i_j}. Starting from C_{i_1}, we perform the following operations: we perform a $d(v, a_{i_1})$-Cycle Rotation over cycle C_{i_1}; for all $j = 2, \dots, n-1$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-Cycle Rotation over cycle C_{i_j}; we perform a $d(a_{i_{n-1}}, y)$-Cycle Rotation over cycle C_{i_n}. Setting $k = (d(w, a_{i_1}), d(a_{i_2}, a_{i_3}), \dots, d(a_{i_{n-2}}, a_{i_{n-1}}))$, this sequence of rotations corresponds to R_k^C. Now, the hole is in the same cycle as w, so we continue as in step 1. So, Bring hole from v to w is defined as

   $$h_{v,w} = R_k^C \, h_{a_{i_{n-1}}, w}.$$

   In this case, the reverse plan bring back hole from w to v is defined by

   $$h_{v,w}^{-1} = h_{a_{i_{n-1}}, w}^{-1} (R_k^C)^+,$$

where $s = (s_n, s_{n-1}, \dots, s_1) = (l_{C_n} - k_n, \dots, l_{C_1} - k_1)$ and $\hat{C} = [C_n, C_{n-1}, \dots, C_1]$.

Bring hole from v to a successor of w. Let $\mathcal{A}$ be an initial configuration, such that $v \in \mathcal{A}(H)$. Let $\pi = u_1 = w, \dots, u_n = v$ be a shortest path from w to v, where u_2 is the successor of w along π. Then, Bring hole from v to a successor of w ($h_{v,s(w)}$) is defined as Bring hole from v to u_2.

Bring back hole from a successor of w to v. Let $h_{v,s(w)}$ be a plan bring hole from v to w. We call bring back hole from w to v its reverse plan $h^{-1}_{v,s(w)}$.

Appendix B. Proofs of the lemmas

In this appendix we show proofs of the lemmas for movements through biconnected components. These proofs provide instructions for the algorithm.

Proof of Entry Lemma 6.3. Fig. 7 illustrates this proof. Let $y \in V$ be such that $(v, y) \in E$. Let h_1 be the *destination* hole in w ($\mathcal{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$ (note that h_2 exists since we are assuming that $|H| \geq 2$). If G is a partially-bidirectional cycle, we set $n = 1$ and $C_1 = C_G$, where C_G is the directed cycle contained in G. We perform a $d(w, y)$-Cycle Rotation over cycle C_G, where $d(w, y)$ is the distance between nodes w and y. In this way, hole h_1 moves to y. Next, pebble p moves on y with $v \to y$. Finally, we perform the complementary $L_{C_G} - d(y, w)$-Cycle Rotation over C_G, in order to move p to w. Namely, the plan is $f_{vw} = r^{C_G}_{d(w,y)}(v \to y)(r^{C_G}_{d(w,y)})^+$, and the final configuration is $\bar{\mathcal{A}}[v, w]$. Indeed, apart from p and h_1, which are exchanged, all pebbles and holes are moved l_{C_G} times, which means that they complete a full revolution, returning to their initial positions. If G has a *r-oed* $L = [L_0, \dots, L_r]$, let u be a successor of w such that $\mathcal{A}(h_2) = u$. If it does not exist, we perform bring hole from $\mathcal{A}(h_2)$ to a successor of w and set u as the successor of w, which corresponds to the new position of h_2. At the end, we will bring back h_2 to its initial position with bring back hole from a successor of w to $\mathcal{A}(h_2)$. By Observation 4.6, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_n}]$ such that $v \in C_{i_1}$ and $u \in C_{i_n}$ and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. Starting from C_{i_1}, we perform the following operations: we perform a $d(w, a_{i_1})$-Cycle Rotation over cycle C_{i_1}; for all $j = 2, \dots, n-1$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-Cycle Rotation over cycle C_{i_j}; we perform a $d(a_{i_{n-1}}, y)$-Cycle Rotation over cycle C_{i_n}. Setting $k = (d(w, a_{i_1}), d(a_{i_2}, a_{i_3}), \dots, d(a_{i_{n-1}}, y))$, this sequence of rotations corresponds to R^C_k. Then, we move p to y and perform the inverse sequence $(R^C_k)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions. Hence, the overall plan that allows us to prove the thesis is $f_{vw} = h_{h_2 s(w)} R^C_k (v \to y)(R^C_k)^+ h^{-1}_{hs(w)}$. □

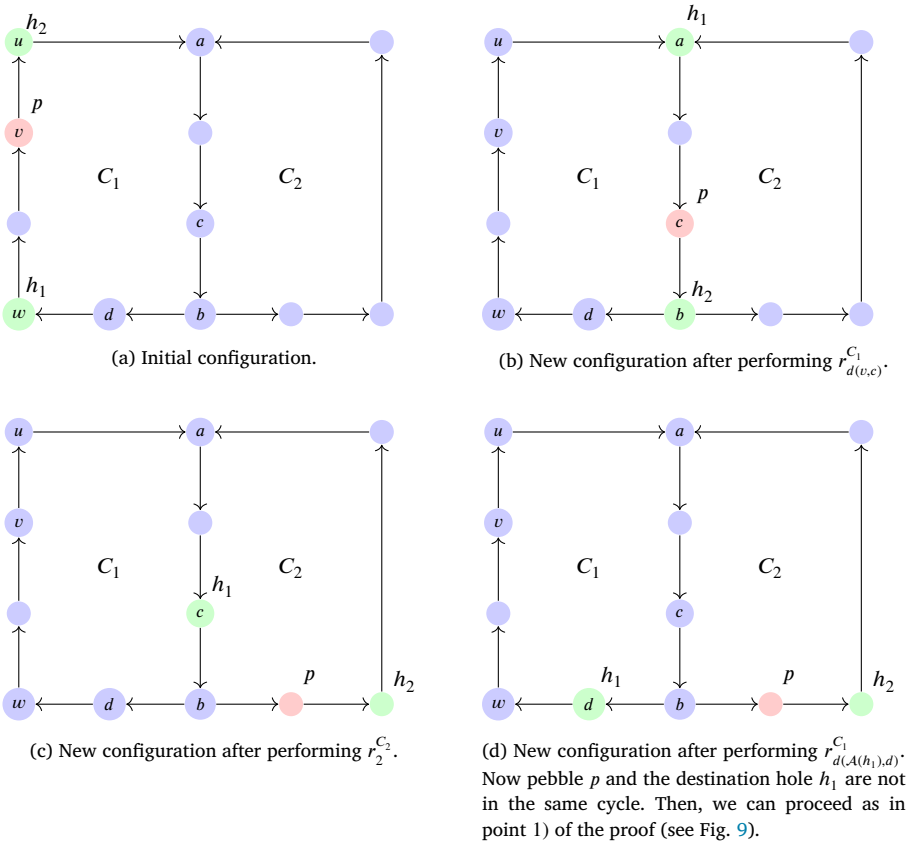
Proof of Go out Lemma 6.4. Let h_1 be the *destination* hole in w ($\mathcal{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$. Without loss of generality we can suppose that exists u a successor of v such that $\mathcal{A}(h_2) = u$. Otherwise, it would be enough to perform at the beginning bring hole from $\mathcal{A}(h_2)$ to a successor of v and at the end we would bring back h_2 to its initial position with bring back hole from a successor of v to $\mathcal{A}(h_2)$. By Observation 4.6, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_m}]$ such that $v, u \in C_{i_1}$ and $y \in C_{i_n}$ and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. Starting from C_{i_1}, we perform the following operations: we perform a $d(u, b_{i_1})$-Cycle Rotation over cycle C_{i_1}; for all $j = 2, \dots, n-1$ we perform a $d(b_{i_j}, b_{i_{j+1}})$-Cycle Rotation over cycle C_{i_j}; we perform a $d(b_{i_{n-1}}, y)$-Cycle Rotation over cycle C_{i_n}. Setting $k = (d(u, b_{i_1}), d(b_{i_2}, b_{i_3}), \dots, d(b_{i_{n-1}}, y))$, this sequence of rotations corresponds to R^C_k. After these rotations, p is on $x \in V$, a predecessor of y. Then, we move p to y and then w and perform the inverse sequence $(R^C_k)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions. Hence, the overall plan that allows us to prove the thesis is $f_{vw} = R^C_k(x \to y)(y \to w)(R^C_k)^+ h^{-1}_{hs(w)}$. □

Proof of Attached-Edge Lemma 6.5. Fig. 8 illustrates this proof. If the strongly biconnected component of D has a *r-oed*, the proof follows from Lemma 6.6. Otherwise, the strongly biconnected component is a partially-bidirectional cycle $G = (V, E)$. We consider the cycle C_G contained in G, which has length l_{C_G}. Let h_1 be the *destination* hole on w and h_2 a *transport* hole (which exists, since $|H| \geq 2$). Without loss of generality, suppose that $\mathcal{A}(h_2) = v$. Indeed, if this were not the case, we can bring hole from $\mathcal{A}(h_2)$ to v and finally bring back it to its initial position. In this case, the new initial configuration is $\bar{A} = \rho(\mathcal{A}, h_{\mathcal{A}(h_2)v})$ and we have to consider $\bar{u} = \mathcal{A}(p)$ and $\bar{w} = \bar{\mathcal{A}}(h_1)$. Let $y \in V$ be the cycle node that shares and arc with v, and consider the distances from u and w to y, $d_1 := d(u, y)$ and $d_2 := d(w, y)$. Performing a d_1-Cycle Rotation over C_G, we move p from w to y. Then, we move p on v with $y \to v$. Now, let

$$k = \begin{cases} d_2 - d_1 & if\ d_2 \geq d_1, \\ l + d_2 - d_1 & if\ d_2 < d_1. \end{cases}$$

We perform a k-Cycle Rotation over C_G, so that we move the hole h_1 from w to y. Next, we move p from v to y with $v \to y$. Finally, we perform a $d(y, w)$-Cycle Rotation over C_G to move p on w. To conclude, $f_{uw} = r^{C_G}_{d(u,y)}(y \to v) r^{C_G}_k (v \to y) r^{C_G}_{d(y,w)}$. □

Proof of Stay in Lemma 6.6. Fig. 9 illustrates this proof. Let h_1 be the *destination* hole in w ($\mathcal{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$. The initial position of pebble p is v ($\mathcal{A}_p = v$). Without loss of generality we can suppose that exists u a successor of v



(a) Initial configuration.

(b) New configuration after performing $r^{C_1}_{d(v,c)}$.

(c) New configuration after performing $r^{C_2}_{2}$.

(d) New configuration after performing $r^{C_1}_{d(\mathcal{A}(h_1),d)}$. Now pebble p and the destination hole h_1 are not in the same cycle. Then, we can proceed as in point 1) of the proof (see Fig. 9).

Fig. B.22. An example of situation of Stay in Lemma 6.6 in which the initial position of the pebble $v = \mathcal{A}(p)$ and the target $w = \mathcal{A}(h_1)$ belong to the same cycle C_1. In this case we move p on another cycle and then we proceed as in Fig. 9.

such that $\mathcal{A}(h_2) = u$ (see Fig. 9a). Otherwise, it would be enough to perform at the beginning BRING HOLE FROM $\mathcal{A}(h_2)$ TO A SUCCESSOR OF v and at the end we would bring back h_2 to its initial position with BRING BACK HOLE FROM A SUCCESSOR OF v TO $\mathcal{A}(h_2)$.

There are two cases:

1. the initial position $\mathcal{A}(p) = v$ of the pebble and the target position $\mathcal{A}(h_1) = w$ does not belong to the same cycle. By Observation 4.6, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_m}]$ such that $v \in C_{i_1}$ and $w \in C_{i_n}$ and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. In particular, choose $a_{i_{m-1}}$ such that his predecessor node x in $C_{i_{m-1}}$ does not belong to C_{i_m}.
   Now, starting from C_{i_1}, we perform the following operations: we perform a $d(v, a_{i_1})$-CYCLE ROTATION over cycle C_{i_1}; for all $j = 2, \dots, m-3$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-CYCLE ROTATION over cycle C_{i_j}; we perform a $d(a_{i_{m-2}}, x)$-CYCLE ROTATION over cycle $C_{i_{m-1}}$ (Fig. 9b) and then a $d(w, a_{i_{m-1}})$-CYCLE ROTATION over cycle C_{i_m}. Setting $k = (d(v, a_{i_1}), d(a_{i_2}, a_{i_3}), \dots, d(a_{i_{m-2}}, x), d(w, a_{i_{m-1}}))$, this sequence of rotations corresponds to R^C_k. Then, we move p to $a_{i_{m-1}}$ (Fig. 9c) and perform the inverse sequence $(R^C_k)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions (Fig. 9e). Hence, the overall plan that allows us to prove the thesis is $f_{vw} = R^C_k(x \to a_{i_{n-1}})(R^C_k)^+$.
2. $\mathcal{A}(p) = v$ and $\mathcal{A}(h_1) = w$ belongs to the same cycle C_1. In this case, the idea is to move the pebble to another cycle and then use the procedure described at point 1) (see Fig. B.22). Let C_2 a cycle such that $|C_1 \cap C_2| \geq 2$.
   We denote with a and b the first and the last nodes that belong to the intersection, with c the predecessor of b in $C_1 \cap C_2$ and with d the successor of b in C_1. We perform a $d(v, c)$-CYCLE ROTATION over cycle C_1 and then a 2-CYCLE ROTATION over cycle C_2. Now, if the destination hole belongs to the intersection between the two cycles (i.e., $\mathcal{A}(h_1) \in C_1 \cap C_2$) we perform a $d(\mathcal{A}(h_1), d)$-CYCLE ROTATION of C_1 in order to move h_1 out of C_2. Thus, setting $k = (d(v, c), 2, d(\mathcal{A}(h_1), d))$ and $C = [C_1, C_2, C_1]$, so far we have performed R^C_k. Now pebble p is in $C_2 \setminus C_1$ and h_1 in $C_1 \setminus C_2$, therefore we can proceed as in point 1). Finally we perform $(R^C_k)^+$. □

Proof of Two Biconnected Components Lemma 6.7. Let $B_1 = (V_1, E_1)$ and $B_2 = (V_2, E_2)$ be the two biconnected components and v be the articulation point, i.e., $V_1 \cup V_2 = V$, $E_1 \cup E_2 = E$, $V_1 \cap V_2 = \{v\}$, and $E_1 \cap E_2 = \emptyset$. Let h_1 be the *destination* hole on b, and let h_2 be a *transport* hole. We discuss different cases.

1. $a \in V_1 \setminus \{v\}$ and $b \in V_2 \setminus \{v\}$.

Without loss of generality, we assume that $\mathcal{A}(h_2) = v$. Indeed, if this is not the case, we can BRING HOLE FROM $\mathcal{A}(h_2)$ TO v, and the new initial configuration is $\bar{\mathcal{A}} = \rho(\mathcal{A}, h_{\mathcal{A}(h_2)v})$. Note that $h_{\mathcal{A}(h_2)v}$ could change the position either of h_1 or of p, i.e., either $\bar{a} = \bar{\mathcal{A}}(p) \neq a$ or $\bar{b} = \bar{\mathcal{A}}(h_1) \neq b$. By the procedure described below, we will reach $\bar{\mathcal{A}}[\bar{a}, \bar{b}]$, and at that point we will need to perform BRING BACK HOLE FROM v TO $\mathcal{A}(h_2)$ in order to obtain, by Observation 6.1, $\mathcal{A}[a, b] = \rho(\bar{\mathcal{A}}[\bar{a}, \bar{b}], h^{-1}_{v,\mathcal{A}(h_2)})$.

Assuming $\mathcal{A}(h_2) = v$, let $y \in V_1$ be such that $(y, v) \in E_1$, i.e., y is a predecessor of v. Now, by *Entry Lemma* 6.3, there exists a plan g such that, setting $\mathcal{A}^1 = \rho(\mathcal{A}, g)$, $\mathcal{A}^1(p) = b$ and, for all $q \in H \cup P \setminus \{p\}$ such that $\mathcal{A}^1(q) \in V_2$, $\mathcal{A}^1(q) = \mathcal{A}(q)$ holds. Plan g is defined as follows

$$g = h_{v,s(b)} R^C_k t_{a,v} (R^C_k)^+ h^{-1}_{v,s(b)},$$

where C is a sequence of cycles, k a vector, and $t_{a,v}$ is a plan which moves pebble p to the unoccupied vertex v. In particular, if B_1:

- has a *r-oed*: $t_{a,v} = a \Rightarrow v$, (see Definition 2.3) which by Theorem 2.4 exists since in B_1 there is at least one hole (h_1);
- is a partially-bidirectional cycle: $t_{a,v} = r^{C_{B_1}}_{d(a,v)}$.

After performing g, both holes h_1 and h_2 are in B_1 (in particular, $\mathcal{A}^1(h_2) = v$). If B_1 has a *r-oed*, by Lemma 6.6 there exists a plan f so that $\mathcal{A}^2 = \rho(\mathcal{A}^1, f)$ is such that $\mathcal{A}^2(h_1) = a$ and for all $q \in P \cup H \setminus \{h_1\}$, $\mathcal{A}^2(q) \in V_1$, $\mathcal{A}^2(q) = \mathcal{A}(q)$. So, finally $\mathcal{A}^2 = \mathcal{A}[a, b]$.

If B_1 is a partially-bidirectional cycle, performing $r^{C_{B_1}}_{d(v,a)}$ is sufficient to bring p on a and the other pebbles of B_1 on their initial positions.

2. Suppose that $a, b \in V_1$.
   - B_1 has a *r-oed*. We can assume without loss of generality that $\mathcal{A}(h_2) \in V_1$ (if not, it would be enough to BRING HOLE FROM $\mathcal{A}(h_2)$ TO v and finally BRING BACK HOLE FROM v TO $\mathcal{A}(h_2)$). Since in B_1 there are two holes, by Lemma 6.6 we can move p to b without changing the final position of the other pebbles.
   - B_1 is a cycle. We can assume without loss of generality that $\mathcal{A}(h_2) \in V_2 \setminus \{v\}$. Then, by point 1) of this proof, we can first move p from a to $\mathcal{A}(h_2)$ and then from $\mathcal{A}(h_2)$ to b, without changing the final position of the other pebbles. □

B.1. *Proof of Theorem 6.8*

In this appendix we show the proof of the Theorem 6.8, which provides the sequence of instructions for the *CONVERT-PATH* algorithm.

Proof of Theorem 6.8. Let f_{ab} be a plan on D. For each single move $m = u \to v$ in this plan, recalling Observation 4.9 and noting that u, v either belong to the same biconnected component or to the same corridor, there are two cases. In the first case there exists a non-trivial strongly biconnected component $B = (V_B, E_B)$ such that $u, v \in V_B$. Then, we define m', a corresponding plan on T, as follows: let S be the star in T corresponding to B, and let s be the trans-shipment vertex of S; then, $m' = (u \to s)(s \to v)$. In the second case, i.e., u, v belong to the same corridor, then $m' = m$. f'_{ab} is defined as the composition of all the moves m' just described.

Conversely, let f'_{ab} be a plan on T. Then, by Observation 6.2

1. If a and b are not in the same star on T, then they are not in the same strongly biconnected component on D. By Observation 6.2 f_{ab} will be composed by movements: from/to a corridor to/from a strongly biconnected component (f_{ab} exists by *Attached-Edge Lemma* 6.5); from a strongly biconnected component to another one, connected by an articulation point (f_{ab} exists by *Two Biconnected Components Lemma* 6.7); from a node to another one of the same corridor ($f'_{ab} = f_{ab}$).
2. If a and b belong to the same "star" on T, then a and b belong to the same strongly biconnected component $B_1 = (W_1, F_1)$ on D. There are two possibilities:
   (a) B_1 has at least two holes. In this case, if B_1 has a *r-oed*, by Lemma 6.6 $f_{a,b}$ exists. If B_1 is a partially-bidirectional cycle there are two cases:
      - there is another biconnected component $B_2 = (W_2, F_2)$ such that B_1 and B_2 are joined by an articulation point. In this case existence of $f_{a,b}$ follows from Lemma 6.7;
      - there is a node $v \in V \setminus W_1$ such that a node $w \in W_1$ with $(v, w), (w, v) \in E$ exists. Therefore, $G = (W_1 \cup \{v\}, F_1 \cup \{(v, w), (w, v)\})$ is a cycle with an attached edge, and existence of $f_{a,b}$ follows from Lemma 6.5;

   (b) B_1 has only one hole. In this case, let h_1 be the hole on b and h_2 a hole such that $\mathcal{A}(h_2) = w \notin W_1$ (which exists since $|H| \geq 2$) and $u \in W_1$ a node different from a and b (which exists since non-trivial biconnected components have at least three nodes). Let $h_{w,u}$ be the plan BRING HOLE FROM w TO u which moves the hole, and $\bar{\mathcal{A}} = (\mathcal{A}, h_{w,u})$. Then:
      a') if $\bar{\mathcal{A}}(p), \bar{\mathcal{A}}(h_1) \in W_1$, then we do perform BRING HOLE FROM w TO u, we replace a and b with $\bar{\mathcal{A}}(p)$ and $\bar{\mathcal{A}}(h_1)$, and by point *a*) we find the plan $f_{\bar{\mathcal{A}}(p)\bar{\mathcal{A}}(h_1)}$; finally we perform $h^{-1}_{w,u}$ and h_2 returns to its initial position.
      b') if $\bar{\mathcal{A}}(h_1) \in W_1$ but $\bar{\mathcal{A}}(p) \notin W_1$, we do perform BRING HOLE FROM w TO u and we fall into the case where start and final position of the pebble are not in the same biconnected component. Therefore, $f_{ab} = h_{w,u} f_{\bar{\mathcal{A}}(p)\bar{\mathcal{A}}(h_1)} h^{-1}_{w,u}$, where $f_{\bar{\mathcal{A}}(p)\bar{\mathcal{A}}(h_1)}$ exists by point 1).
      c') if $\bar{\mathcal{A}}(p) \in W_1$ but $\bar{\mathcal{A}}(h_1) \notin W_1$, we do not perform BRING HOLE FROM w TO u. First, we move h_1 away from b by BRING HOLE FROM b TO u. Then, we perform BRING HOLE FROM w TO b. Then, we can proceed as in a') or b') with u replaced

by b. Finally, we will need to perform BRING BACK HOLE FROM u TO b. In formulas, given $\tilde{\mathcal{A}} = \rho(\mathcal{A}, h_{b,u}h_{w,b})$ the final plan is $f_{ab} = h_{b,u}h_{w,b}f_{\tilde{\mathcal{A}}(p),\tilde{\mathcal{A}}(h_1)}h_{w,b}^{-1}h_{b,u}^{-1}$. □

Data availability

No data was used for the research described in the article.

References

[1] Ebtehal Turki, Saho Alotaibi, Hisham Al-Rawi, Push and spin: a complete multi-robot path planning algorithm, in: 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE, 2016, pp. 1–8.
[2] S. Ardizzoni, I. Saccani, L. Consolini, M. Locatelli, Multi-agent path finding on strongly connected digraphs, in: 2022 IEEE 61st Conference on Decision and Control (CDC), IEEE, 2022, pp. 7194–7199.
[3] Stefano Ardizzoni, Luca Consolini, Marco Locatelli, Irene Saccani, Constrained motion planning and multi-agent path finding on directed graphs, Automatica 165 (2024) 111593.
[4] Stefano Ardizzoni, Irene Saccani, Luca Consolini, Marco Locatelli, Local optimization of mapf solutions on directed graphs, in: 2023 62nd IEEE Conference on Decision and Control (CDC), IEEE, 2023, pp. 8081–8086.
[5] Stefano Ardizzoni, Irene Saccani, Luca Consolini, Marco Locatelli, Bernhard Nebel, An algorithm with improved complexity for pebble motion/multi-agent path finding on trees, J. Artif. Intell. Res. 79 (2024) 483–514.
[6] Vincenzo Auletta, Angelo Monti, Mimmo Parente, Pino Persiano, A linear-time algorithm for the feasibility of pebble motion on trees, Algorithmica 23 (3) (1999) 223–245.
[7] Vincenzo Auletta, Pino Persiano, Optimal pebble motion on a tree, Inf. Comput. 165 (1) (2001) 42–68.
[8] Jørgen Bang-Jensen, Gregory Z. Gutin, Digraphs: Theory, Algorithms and Applications, Springer Science & Business Media, 2008.
[9] Edouard Bonnet, Tillmann Miltzow, Paweł Rzazewski, Complexity of token swapping and its variants, Algorithmica 80 (2018) 2656–2682.
[10] Adi Botea, Pavel Surynek, Multi-agent path finding on strongly biconnected digraphs, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, 2015.
[11] Boris De Wilde, Adriaan W. Ter Mors, Cees Witteveen, Push and rotate: a complete multi-agent pathfinding algorithm, J. Artif. Intell. Res. 51 (2014) 443–492.
[12] Gilad Goraly, Refael Hassin, Multi-color pebble motion on graphs, Algorithmica 58 (3) (2010) 610–636.
[13] Mehmet Hakan Karaata, A stabilizing algorithm for finding biconnected components, J. Parallel Distrib. Comput. 62 (5) (2002) 982–999.
[14] Mokhtar M. Khorshid, Robert C. Holte, Nathan R. Sturtevant, A polynomial-time algorithm for non-optimal multi-agent pathfinding, in: Fourth Annual Symposium on Combinatorial Search, 2011.
[15] Daniel Martin Kornhauser, Gary Miller, Paul Spirakis, Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications, PhD thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.
[16] Athanasios Krontiris, Ryan Luna, Kostas Bekris, From feasibility tests to path planners for multi-agent pathfinding, in: International Symposium on Combinatorial Search, vol. 4, 2013.
[17] Pierre Le Bodic, Edward Lam, Optimal unlabeled pebble motion on trees, in: Proceedings of the International Symposium on Combinatorial Search, vol. 17, 2024, pp. 218–222.
[18] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, Sven Koenig, Anytime multi-agent path finding via large neighborhood search, in: International Joint Conference on Artificial Intelligence 2021, Association for the Advancement of Artificial Intelligence (AAAI), 2021, pp. 4127–4135.
[19] Bernhard Nebel, On the computational complexity of multi-agent pathfinding on directed graphs, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, 2020, pp. 212–216.
[20] Bernhard Nebel, The small solution hypothesis for mapf on strongly connected directed graphs is true, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 33, 2023, pp. 304–313.
[21] Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, Hisao Tamaki, Motion planning on a graph, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, IEEE, 1994, pp. 511–520.
[22] Gabriele Röger, Malte Helmert, Non-optimal multi-agent pathfinding is solved (since 1984), in: Proceedings of the International Symposium on Combinatorial Search, vol. 3, 2012, pp. 173–174.
[23] Guni Sharon, Roni Stern, Ariel Felner, Nathan R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66.
[24] David Silver, Cooperative pathfinding, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 1, 2005, pp. 117–122.
[25] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T.K. Kumar, et al., Multi-agent pathfinding: definitions, variants, and benchmarks, in: Proceedings of the International Symposium on Combinatorial Search, vol. 10, 2019, pp. 151–158.
[26] Pavel Surynek, Lazy modeling of variants of token swapping problem and multi-agent path finding through combination of satisfiability modulo theories and conflict-based search, arXiv preprint, arXiv:1809.05959, 2018.
[27] Douglas Brent West, et al., Introduction to Graph Theory, vol. 2, Prentice Hall, Upper Saddle River, 2001.
[28] Zhilin Wu, Stéphane Grumbach, Feasibility of motion planning on acyclic and strongly connected directed graphs, Discrete Appl. Math. 158 (9) (2010) 1017–1028.
[29] Zhilin Wu, Stéphane Grumbach, Feasibility of motion planning on directed graphs, in: Theory and Applications of Models of Computation: 6th Annual Conference, TAMC 2009, Changsha, China, May 18-22, 2009. Proceedings 6, Springer, 2009, pp. 430–439.
[30] Jingjin Yu, Steven M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.