Multi-robot Path Planning

Pavel Surynek
Charles University in Prague
Czech Republic

1. Introduction

This chapter is devoted to a problem of *path planning for multiple robots* (Ryan, 2008; Surynek, 2010a). Consider a group of mobile robots that can move in some environment (for example in the 2-dimensional plane with obstacles). Each robot of the group is given an initial and a goal position in the environment. The question of our interest is how to determine a sequence of motions for each robot of the group such that all the robots reach their goal positions starting from the initial ones. Physical limitations must be respected by robots: robots must not collide with each other and they must avoid obstacles in the environment during their movements.

The problem of multi-robot path planning is motivated by many practical tasks. Various problems of navigating a group of mobile robots can be formulated as multi-robot path planning. However, the primary motivations for the problem are tasks of moving certain entities within an environment with a limited free space. Hence, the formulation of the problem is not limited to the case where robots are actually represented by mobile robots. The constraint of limited free space represents a key aspect that makes the problem interesting and non-trivial. It is quite intuitive to see that the problem becomes easier with a lot of free space in the environment. The interaction between robots (which corresponds to the probability of collisions) is low in such a case. Thus, it is possible to plan movements of each robot relatively independently with respect to other robots. Then the problem reduces almost to the series of simple single source path finding (Cormen et al., 2001) for each robot; potentially with finding alternative paths if a collision still occurs.

Fig. 1. *An illustration of shipping container rearranging.* This problem can be formulated as path planning for multiple robots where robots are represented by containers.

On the other hand, if there is limited free space in the environment the problem becomes harder. Consider the situation where the space occupied by robots is comparable to the free space or even the situation where robots occupy larger space than the space that remains unoccupied in the environment. The interaction between robots (that is, the probability of collisions) is so high in such a case that finding a path for each robot independently no longer works. Therefore, different methods must be used.

One of the aims of this chapter is to explain solving methods for these cases. Notice, that it is desirable to reason about the case with limited free space since practical tasks typically has this property. Such real-life examples include rearranging of shipping containers in warehouses (a robot is represented by a shipping container – see Fig. 1) or coordination of vehicles in dense traffic (robot = vehicle). Additionally, our reasoning should not be limited to physical entities only. A robot may be represented by a virtual entity or by a piece of commodity too. Thus, many tasks such as planning of data transfer between communication nodes with limited storage capacity (robot = data packet), commodity transportation in the commodity transportation network (robot = certain amount of commodity), or even the motion planning of large groups of virtual agents in the computer-generated imagery can be expressed as the problem of multi-robot path planning.

We would like also to emphasize that an approach to solving tasks of multi-robot path planning presented in this chapter is substantially different from the multi-agent approach where the final plan is constructed in a distributed manner through communication between individual agents. A robot in our case is not regarded as an autonomous agent with communication ability. The plan for individual robots is constructed by the centralized mechanism (that is represented by a solving method in our case), which is capable of observing the whole environment and all the robots. The individual robots merely execute the centrally created plan.

2. Formal Abstraction

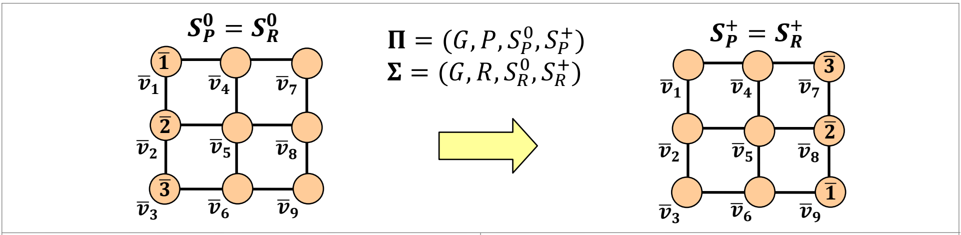
When dealing with the problem of multi-robot path planning it will soon turn out to be necessary to work with the problem at the highly abstract level. Nevertheless, even with the high abstraction the problem remains computationally hard. Let us now introduce abstract formulations of the problem where the environment is modeled as a graph. Vertices of the graph represent locations in the environment. Time is modeled as a structure of discrete time steps isomorphic to the structure of natural numbers (including zero). Robots are placed in vertices of the graph while there is at most one robot in each vertex. Robots can move between neighboring vertices in a single step. That is, an edge represents an unobstructed way between locations in the environment represented by connected vertices that can be travelled by a robot in a single time step. At least one vertex is left unoccupied to allow robots to move. An initial and a goal arrangement of robots are also given to specify the task.

There are two different approaches to the dynamicity of the problem. One approach is that a move is allowed to a currently unoccupied vertex only – this variant of the problem is known as coordinating *pebble motion on a graph* (Wilson, 1974; Kornhauser et al., 1984) . Another approach is to allow moves into currently vacated vertices. This allows a group of robots to move like a train where only the leading robot must move into an unoccupied vertex; the other robots directly follow it. As this variant of the problem is closer to the reality of how robots are required to move, it is called *multi-robot path planning* (Surynek, 2010a). Following definitions describes both variants of the problem formally.

Definition 1 ***(problem of pebble motion on a graph).*** Let $G = (V, E)$ be an undirected graph. Next, let $P = \{\bar{p}_1, \bar{p}_2, \dots, \bar{p}_\mu\}$ where $\mu < |V|$ be a set of pebbles. The graph models an environment in which the pebbles are moving. An ***initial arrangement*** of the pebbles is defined by a uniquely invertible function $S_P^0 \colon P \longrightarrow V$ (that is $S_P^0(p) \neq S_P^0(q)$ for every $p, q \in P \in$ with $p \neq q$). A ***goal arrangement*** of the pebbles is defined by another uniquely invertible function $S_P^+ \colon P \longrightarrow V$ (that is $S_P^+(p) \neq S_P^+(q)$ for $p, q \in P$ with $p \neq q$). A problem of ***pebble motion on a graph*** is the task to find a number ξ and a sequence $\mathcal{S}_P = [S_P^0, S_P^1, \dots, S_P^\xi]$ where $S_P^k \colon P \longrightarrow V$ is a uniquely invertible function for every $k = 1,2,\dots,\xi$. Additionally, the following conditions must hold for the sequence $\mathcal{S}_P$:

- **(i)** $S_P^\xi = S_P^+$; that is, all the pebble reaches their destination vertices.
- **(ii)** Either $S_P^k(p) = S_P^{k+1}(p)$ or $\{S_P^k(p), S_P^{k+1}(p)\} \in E$ for every $p \in P$ and $k = 1,2,\dots,\xi - 1$; that is, a pebble can either stay in a vertex or move into the neighboring vertex between each two successive time steps.
- **(iii)** If $S_P^k(p) \neq S_P^{k+1}(p)$ (that is, the pebble p moves between time steps k and $k + 1$) then $S_P^k(q) \neq S_P^{k+1}(p)$ $\forall q \in P$ such that $q \neq p$; must hold for every $p \in P$ and $k = 1,2,\dots,\xi - 1$; that is, a pebble can move into an unoccupied neighboring vertex only. This constraint together with unique invertibility of functions forming $\mathcal{S}_P$ implies that no two pebbles can enter the same target vertex at the same time step.

The instance of the problem of pebble motion on a graph is formally a quadruple $\Pi = (G, P, S_P^0, S_P^+)$. Sometimes, the solution of the problem Π will be denoted as $\mathcal{S}_P(\Pi) = [S_P^0, S_P^1, \dots, S_P^\xi]$. □



Solution of the problem of *pebble motion on a graph* $\mathbf{\Pi}$ with $\boldsymbol{P} = \{\bar{\mathbf{1}}, \bar{\mathbf{2}}, \bar{\mathbf{3}}\}$

$\xi = 6$ $\boldsymbol{P}$	$\boldsymbol{S_P^0}$	$\boldsymbol{S_P^1}$	$\boldsymbol{S_P^2}$	$\boldsymbol{S_P^3}$	$\boldsymbol{S_P^4}$	$\boldsymbol{S_P^5}$	$\boldsymbol{S_P^6 = S_P^+}$
$\bar{1}$	$\bar{\boldsymbol{v}}_\mathbf{1}$	$\bar{v}_4$	$\bar{v}_7$	$\bar{v}_8$	$\bar{v}_9$	$\bar{v}_9$	$\bar{\boldsymbol{v}}_\mathbf{9}$
$\bar{2}$	$\bar{\boldsymbol{v}}_\mathbf{2}$	$\bar{v}_2$	$\bar{v}_1$	$\bar{v}_4$	$\bar{v}_7$	$\bar{v}_8$	$\bar{\boldsymbol{v}}_\mathbf{8}$
$\bar{3}$	$\bar{\boldsymbol{v}}_\mathbf{3}$	$\bar{v}_3$	$\bar{v}_3$	$\bar{v}_2$	$\bar{v}_1$	$\bar{v}_4$	$\bar{\boldsymbol{v}}_\mathbf{7}$

Solution of the problem of *path planning for multiple robots* $\mathbf{\Sigma}$ with $\boldsymbol{R} = \{\bar{\mathbf{1}}, \bar{\mathbf{2}}, \bar{\mathbf{3}}\}$

$\zeta = 4$ $\boldsymbol{R}$	$\boldsymbol{S_R^0}$	$\boldsymbol{S_R^1}$	$\boldsymbol{S_R^2}$	$\boldsymbol{S_R^3}$	$\boldsymbol{S_R^4 = S_R^+}$
$\bar{1}$	$\bar{\boldsymbol{v}}_\mathbf{1}$	$\bar{v}_4$	$\bar{v}_7$	$\bar{v}_8$	$\bar{\boldsymbol{v}}_\mathbf{9}$
$\bar{2}$	$\bar{\boldsymbol{v}}_\mathbf{2}$	$\bar{v}_1$	$\bar{v}_4$	$\bar{v}_7$	$\bar{\boldsymbol{v}}_\mathbf{8}$
$\bar{3}$	$\bar{\boldsymbol{v}}_\mathbf{3}$	$\bar{v}_2$	$\bar{v}_1$	$\bar{v}_4$	$\bar{\boldsymbol{v}}_\mathbf{7}$

Fig. 2. *An illustration of problems of* ***pebble motion on a graph*** *and* ***multi-robot path planning***. Both problems are illustrated on the same graph with the same initial and goal positions. The task is to move pebbles/robots from their initial positions specified by S_P^0/S_R^0 to the goal positions specified by S_P^+/S_R^+. A solution of the makespan 6 ($\xi = 6$) is shown for the problem of pebble motion on a graph and a solution of the makespan 4 ($\zeta = 4$) is shown for the problem of multi-robot path planning. Notice the differences in parallelism between both solutions.

The notation with a stripe above the symbol is used to distinguish a constant from a variable (for example, $p \in P$ is a variable while $\bar{p}_2$ is a constant; sometimes a constant parameterized by a variable or by an expression will be used – for example $\bar{p}_i$ denotes a constant parameterized by an index $i \in \mathbb{N}$; the parameterization by an expression will be clear from the context).

When speaking about a move at time step k, it is referred to the time step of commencing the move (exactly, the move is performed between time steps k and $k+1$). A problem of multi-robot path planning is a relaxation of the problem of pebble motion on a graph. The condition that the target vertex of a pebble/robot must be vacated in the previous time step is relaxed. Thus, the motion of a robot entering the target vertex that is simultaneously vacated by another robot is allowed in multi-robot path planning. However, there must be some leading robot initiating such chain of moves by moving into an unoccupied vertex.

Definition 2 ***(problem of multi-robot path planning).*** Again, let $G = (V, E)$ be an undirected graph. Now a set of robots $R = \{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_\nu\}$ where $\nu < |V|$ is given instead of the set of pebbles. Similarly, the graph models an environment in which the robots are moving. The ***initial arrangement*** of the robots is defined by a uniquely invertible function $S_R^0: R \longrightarrow V$ (that is $S_R^0(r) \neq S_R^0(s)$ for $\forall r, s \in R$ with $r \neq s$). The ***goal arrangement*** of the robots is defined by another uniquely invertible function $S_R^+: R \longrightarrow V$ (that is $S_R^+(r) \neq S_R^+(s)$ for $\forall r, s \in R$ with $r \neq s$). A problem of ***multi-robot path planning*** is the task to find a number ζ and a sequence $\mathcal{S}_R = [S_R^0, S_R^1, \dots, S_R^\zeta]$ where $S_R^k: R \longrightarrow V$ is a uniquely invertible function for every $k = 1,2, \dots, \zeta$. The following conditions must hold for the sequence $\mathcal{S}_R$:

- **(i)** $S_R^\zeta = S_R^+$; that is, all the robots reaches their destination vertices.
- **(ii)** Either $S_R^k(r) = S_P^{k+1}(r)$ or $\{S_R^k(r), S_R^{k+1}(r)\} \in E$ for every $r \in R$ and $k = 1,2, \dots, \zeta - 1$; that is, a robot can either stay in a vertex or move to the neighboring vertex at each time step.
- **(iii)** If $S_R^k(r) \neq S_R^{k+1}(r)$ (that is, the robot r moves between time steps k and $k+1$) and $S_R^k(s) \neq S_R^{k+1}(r)$ $\forall s \in R$ such that $s \neq r$ (that is, no other robot s occupies the target vertex at time step k), then the move of r at the time step k is called to be ***allowed*** (that is, the robot r moves into an unoccupied neighboring vertex – a ***leading*** robot). If $S_R^k(r) \neq S_R^{k+1}(r)$ and there is $s \in R$ such that $s \neq r \wedge S_R^k(s) = S_R^{k+1}(r) \wedge S_R^k(s) \neq S_R^{k+1}(s)$ (that is, the robot r moves into a vertex that is being left by the robot s) and the move of s at the time step k is allowed, then the move of r at the time step k is also ***allowed***. All the moves of robots at all the time steps **must be allowed**. And analogically, this condition together with the requirement on unique invertibility of functions forming $\mathcal{S}_R$ implies that no two robots can enter the same target vertex at the same time step.

The instance of the problem of multi-robot path planning is formally a quadruple $\Sigma = (G, R, S_R^0, S_R^+)$. The solution of the problem Σ will be sometimes denoted as $\mathcal{S}_R(\Sigma) = [S_R^0, S_R^1, \dots, S_R^\zeta]$. □

The numbers ξ and ζ are called ***makespan*** of the solution of pebble motion on a graph and multi-robot path planning respectively. The makespan needs to be distinguished from the ***size*** of the solution, which is the total number of moves performed by pebbles/robots.

3. Properties of Problems and Complexity Issues

Let us now summarize several basic properties of the solutions of instances of problems of pebble motion on graphs and multi-robot path planning.

Notice that a solution of the problem of pebble motion on a graph as well as a solution of the problem of multi-robot path planning allows a pebble/robot to stay in a vertex for more than a single time step. It is also possible that a pebble/robot visits the same vertex several times within the solution. Hence, a sequence of moves for a single pebble/robot does not necessarily form a simple path in the input graph. Notice further that both problems intrinsically allow parallel movements of pebbles/robots. That is, more than one pebble/robot can perform a move in a single time step. However, multi-robot path planning allows higher motion parallelism due to its weaker requirements on robot movements (the target vertex is required to be unoccupied only for the leading robot in the previous time step – see Fig. 2). More than one unoccupied vertex is necessary to obtain parallelism in the problem of pebble motion on a graph. On the other hand, it is sufficient to have a single unoccupied vertex to obtain parallelism within the solution of multi-robot path planning problem (consider for example robots moving around a cycle).

The following proposition puts in relation the solution of an instance of multi-robot path planning and the solution of the corresponding instance of pebble motion on a graph (the instance of multi-robot path planning problem consists of the same graph, the set of robots is represented by the set of pebbles, and the initial/goal positions of robots are the same as in the case of pebbles). As the proof is easy, it is left as an exercise.

Proposition 1 ***(problem correspondence).*** Let $\Pi = (G, P, S_P^0, S_P^+)$ be an instance of the problem of pebble motion on a graph and let $\mathcal{S}_P(\Pi) = [S_P^0, S_P^1, \ldots, S_P^\xi]$ be its solution. Then $\mathcal{S}_R(\Sigma) = \mathcal{S}_P(\Pi)$ is a solution of an instance of the problem of path planning for multiple robots $\Sigma = (G, P, S_P^0, S_P^+)$. ■

There is a variety of modifications of the defined problems. A natural additional requirement is to produce solutions with **makespan** as small as possible (that is, the numbers ξ or ζ respectively are required to be as small as possible). Unfortunately, this requirement makes the problem of pebble motion on a graph intractable. It is shown in (Ratner & Warmuth, 1986) that the optimization variant of a special case of the problem of pebble motion on a graph is NP-hard (Garey & Johnson, 1979). The restriction forming the special case adopted in (Ratner & Warmuth, 1986) works with a graph that can be embedded in plane as a square grid and where there is a single unoccupied vertex - this case is known as $N \times N$ *puzzle* (also known as $N^2 - 1$ *puzzle*). Hence, the general optimization variant of the problem of pebble motion on a graph is NP-hard as well.

A restriction of both types of problems on *bi-connected graphs* (West, 2000) (for the precise definitions see Section 4.1) represents important subclass with respect to the existence of a solution. Hence, it is a reasonable question what is the complexity of these classes of problems. Since the grid graph forming the mentioned $N \times N$ puzzle is bi-connected, the immediate answer is that the optimization variant of the problem of pebble motion on a bi-connected graph with a single unoccupied vertex is again NP-hard.

However, it is not possible to simply make any similar statement about the complexity of the optimization variant of multi-robot path planning based on the above facts. The situation here is complicated by the inherent parallelism, which can reduce the makespan of the solution significantly. Constructions used for the $N \times N$ puzzle in (Ratner & Warmuth, 1986) thus no longer work. It has been shown recently in (Surynek, 2010a) that the optimization variant of multi-robot path planning is NP-hard as well. As the proof of this result is technically complicated and not all the details fit into the conference paper (Surynek, 2010a) we refer the reader to technical report (Surynek, 2010b) for further reading.

Observe that above difficult cases of the problem of pebble motion on a graph have a single unoccupied vertex. This fact may raise the question how the situation is changed when there are more than one unoccupied vertices. More unoccupied vertices may simplify the problem. Unfortunately, it is not the case. The pebble motion problem on a general graph with the fixed number of unoccupied vertices is still NP-hard since multiple copies of the $N \times N$ puzzle can be used to add as many unoccupied vertices as needed.

Without the requirement on the optimality of the makespan of solutions the situation is much easier; the problem of pebble motion on a graph is in the P class as it is shown in (Wilson, 1974; Kornhauser et al., 1984). Due to Proposition 1, the problem of path planning for multiple robots is in the P class as well. Moreover, it is shown in (Kornhauser et al., 1984) that a solution of the size of $O(|V|^3)$ can be generated for $\Pi = (G = (V, E), P, S_P^0, S_P^+)$. Hence, it provides us a polynomial upper bound on size of the oracle in the content to guess in non-deterministic model. Thus, it is possible to conclude that decision versions of optimization variants of both pebble motion on a graph as well as of multi-robot path planning are NP-complete problems (Garey & Johnson, 1979). By the decision version it is meant a yes/no question whether there is a solution of Π/Σ of the makespan smaller than the given bound. As constructions proving the membership of the problem of pebble motion on a graph into the P class used in (Wilson, 1974; Kornhauser et al., 1984) generate solutions that are too long, we will show an alternative solving algorithm proposed in (Surynek, 2009a, 2009b).

4. Solving Algorithm

This section is devoted to a sub-optimal algorithm for solving problems of motion on a graph in polynomial time. The algorithm presented in the following text is designed for the problem of pebble motion on a graph. As we have Proposition 1, algorithm for pebble motion on a graph applies also for multi-robot path planning. However, the practice of solving multi-robot path planning problems using algorithms for pebble motion on a graph does not reflect the possibility of higher parallelism in multi-robot path planning. Particularly, parallelism in the form of the "train like" movement of a queue of robots is never produced in this way. This drawback can be augmented by a post-processing step that increases parallelism which is discussed in Section 5.

The $BIBOX$ algorithm that will be presented below comes from (Surynek, 2009a). The input instance of the problem of pebble motion on a graph should consist of a so called *non-trivial bi-connected graph* with exactly **two unoccupied** vertices. Overcoming some of these assumptions is discussed in Section 4.4.

4.1 Graph-theoretical Preliminaries

Several notions from graph theory (Tarjan, 1972; West, 2000) are introduced in this section. Following definitions establish the notion of bi-connectivity upon that the $BIBOX$ algorithm is built. Several useful properties of bi-connected graph will be also discussed.

Definition 3 *(connected graph).* An undirected graph $G = (V, E)$ is ***connected*** if $|V| \geq 2$ for any two vertices $u, v \in V$ such that $u \neq v$ there is an undirected path consisting of edges from E connecting u and v. □

Definition 4 (*bi-connected graph, non-trivial).* An undirected graph $G = (V, E)$ is ***bi-connected*** if $|V| \geq 3$ and the graph $G' = (V', E')$, where $V' = V \setminus \{v\}$ and $E' = \{\{u, w\} | u, w \in V \wedge u \neq v \wedge w \neq v\}$, is connected for every $v \in V$. A bi-connected graph not isomorphic to a cycle will be called ***non-trivial*** bi-connected graph. □

Observe that, if a graph is bi-connected, then every two distinct vertices are connected by at least two *vertex disjoint paths* (equivalently, there is a cycle containing both vertices; only internal vertices of paths are considered when speaking about vertex disjoint paths - vertex disjoint paths can intersect in their *start points* and *endpoints*). If a graph is not bi-connected then it is either disconnected or there exists a vertex which removal partitions the graph into at least two connected components – this vertex is called an *articulation point*. Several examples of bi-connected graphs are shown in Fig. 3.

Bi-connected graphs have an important property, which is exploited within the algorithm. Each bi-connected graph can be constructed from a cycle by an operation of *adding a handle* to the graph. Consider a graph $G = (V, E)$; the new handle with respect to G is a sequence $L = [u, w_1, w_2, \ldots, w_l, v]$ where $l \in \mathbb{N}_0$, $u, v \in V$ (called *connection vertices*) and $w_i \notin V$ for $i = 1,2, \ldots, l$ (w_i are new vertices). The result of the addition of the handle L to the graph G is a new graph $G' = (V', E')$ where $V' = V \cup \{w_1, w_2, \ldots, w_l\}$ and either $E' = E \cup \{\{u, v\}\}$ in the case of $l = 0$ of $E' = E \cup \{\{u, w_1\}, \{w_1, w_2\}, \ldots, \{w_{l-1}, w_l\}, \{w_l, v\}\}$ in the case of $l \geq 0$. Let the sequence of handles together with the initial cycle be called a *handle decomposition* of the given graph. See Fig. 3 for examples.

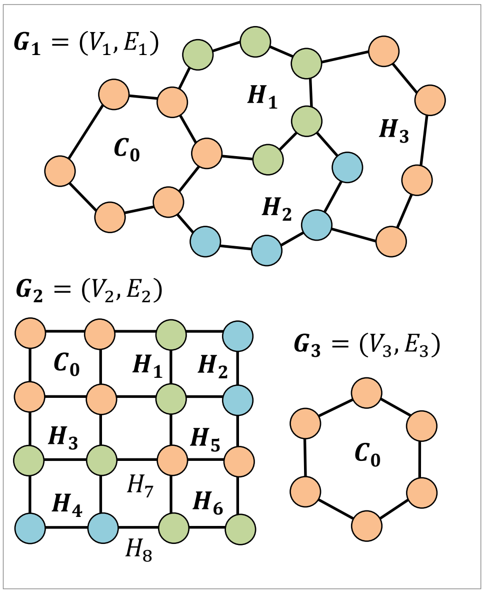


Fig. 3. *Examples of* ***bi-connected graphs.*** Three bi-connected graphs G_1, G_2, and G_3 and their handle decompositions are shown using colors (handles H_7 and H_8 of the decomposition of G_2 consist of a single edge).

Lemma 1 *(handle decomposition)* (Tarjan, 1972). Any bi-connected $G = (V, E)$ graph can be obtained from a cycle by a sequence of operations of adding a handle. Moreover, the corresponding handle decomposition of the graph G can be effectively found in the worst case time of $O(|V| + |E|)$ and worst case space of $O(|V| + |E|)$. ■

An important property of the construction of a bi-connected graph according to its handle decomposition is that the currently constructed graph is bi-connected at any stage of the construction. This property is substantially exploited in the design of the *BIBOX* solving.

The *BIBOX* algorithm is presented using a pseudo-code as Algorithm 1 (the algorithm is illustrated with pictures for easier understanding). The algorithm starts with the last handle of the handle decomposition and proceeds to the original cycle. Pebbles, which goal positions are within the last handle, are moved to their goal positions within this handle. The instance of the problem now reduces to the instance of the same type indeed on a smaller bi-connected graph. That is, the last handle is not considered any more since its pebbles do not need to move any more. This process is repeated until the original cycle of the decomposition remains.

Let $\Pi = (G = (V, E), P, S_P^0, S_P^+)$ be an instance of the pebble motion problem. The following notation is used in the formalization of the algorithm. The handle decomposition of the graph G is formally a sequence $\mathcal{D} = [C_0, H_1, H_2, \dots, H_d]$, where C_0 is the initial cycle and H_c is a handle for $c = 1, 2, \dots, d$. The order of handle additions in construction of G corresponds to their positions in the sequence (that is, H_1 is added to C_0 first; and H_d is added as the last to the currently constructed graph). A handle $H_c = [u^c, w_1^c, w_2^c, \dots, w_{h_i}^c, v^c]$ for $c \in \{1, 2, \dots, d\}$ can be assigned a cycle $C(H_i)$ if the input graph G is connected. The cycle $C(H_c)$ consists of the sequence of vertices on a path connecting v^c and u^c in a graph before the addition of H_c followed by vertices $w_1^c, w_2^c, \dots, w_{h_i}^c$. Specially, it is defined that $C(C_0) = C_0$. The following lemma is important for the design of the algorithm.

Lemma 2 ***(two paths existence).*** Let $G = (V, E)$ be a bi-connected graph and let $u_1, u_2 \in V$ and $v_1, v_2 \in V$, where u_1, u_2, v_1, v_2 are pair-wise distinct, be two pairs of vertices. Then either the first or the second of the following claims holds:

(a) There exist two vertex disjoint paths φ and χ such that they connect u_1 with v_1 and u_2 with v_2 in G respectively.

(b) There exist two vertex disjoint paths φ and χ such that they connect u_1 with v_2 and u_2 with v_1 in G respectively. ■

Notice that the above lemma states that individual vertices in the input pair of vertices are indifferent with respect to connecting by vertex disjoint paths.

Proof. The idea of the proof is to proceed inductively according to the size of the handle decomposition of the graph $G = (V, E)$. Let $\mathcal{D} = [C_0, H_1, H_2, \dots, H_d]$ be a handle decomposition of the graph G. A function $c_{\mathcal{D}}: V \longrightarrow \mathbb{N}_0$ is defined as follows: $c_{\mathcal{D}}(v) = 0$ if $v \in C_0$ and $c_{\mathcal{D}}(v) = c$ if $v \in H_c$ for some $c \in \{1, 2, \dots, d\}$ (v is one of the internal vertices of the handle L_c). Observe, that $c_{\mathcal{D}}$ is a correctly defined function.

A given 4-tuple of vertices (u_1, u_2, v_1, v_2) is assigned a 4-tuple of integers defined using the function $c_{\mathcal{D}}$: $(c_{\mathcal{D}}(u_1), c_{\mathcal{D}}(u_2), c_{\mathcal{D}}(v_1), c_{\mathcal{D}}(v_2))$. The mathematical induction will proceed according to the lexicographic ordering of the 4-tupules assigned using the function $c_{\mathcal{D}}$ sorted in descending order. Several cases must be distinguished.

Case (i): Let the 4-tuple of vertices (u_1, u_2, v_1, v_2) is assigned a 4-tuple of numbers $(1,1,1,1)$, that is, all the vertices u_1, u_2, v_1, v_2 are located within the initial cycle C_0. Then the following juxtapositions of vertices u_1, u_2, v_1, and v_2 within C_0 with respect to the positive orientation of the cycle can occur: (u_1, v_1, v_2, u_2), (u_1, v_2, v_1, u_2), (v_1, u_1, v_2, u_2), (v_1, v_2, u_1, u_2), (v_2, u_1, v_1, u_2), and (v_2, v_1, u_1, u_2) (vertices are listed according to the positive orientation of the cycle; there is in total $4! = 24$ candidates for juxtapositions of 4 vertices; however, the remaining juxtapositions are isomorphic to the listed ones using a rotation along the cycle). In all the cases either the claim (a) or the claim (b) holds. See Fig. 4 for detailed case analysis

– for example in the juxtaposition (u_1, v_1, v_2, u_2), u_1 should be connected in positive orientation with v_1 and u_2 should be connected in negative orientation with v_2.

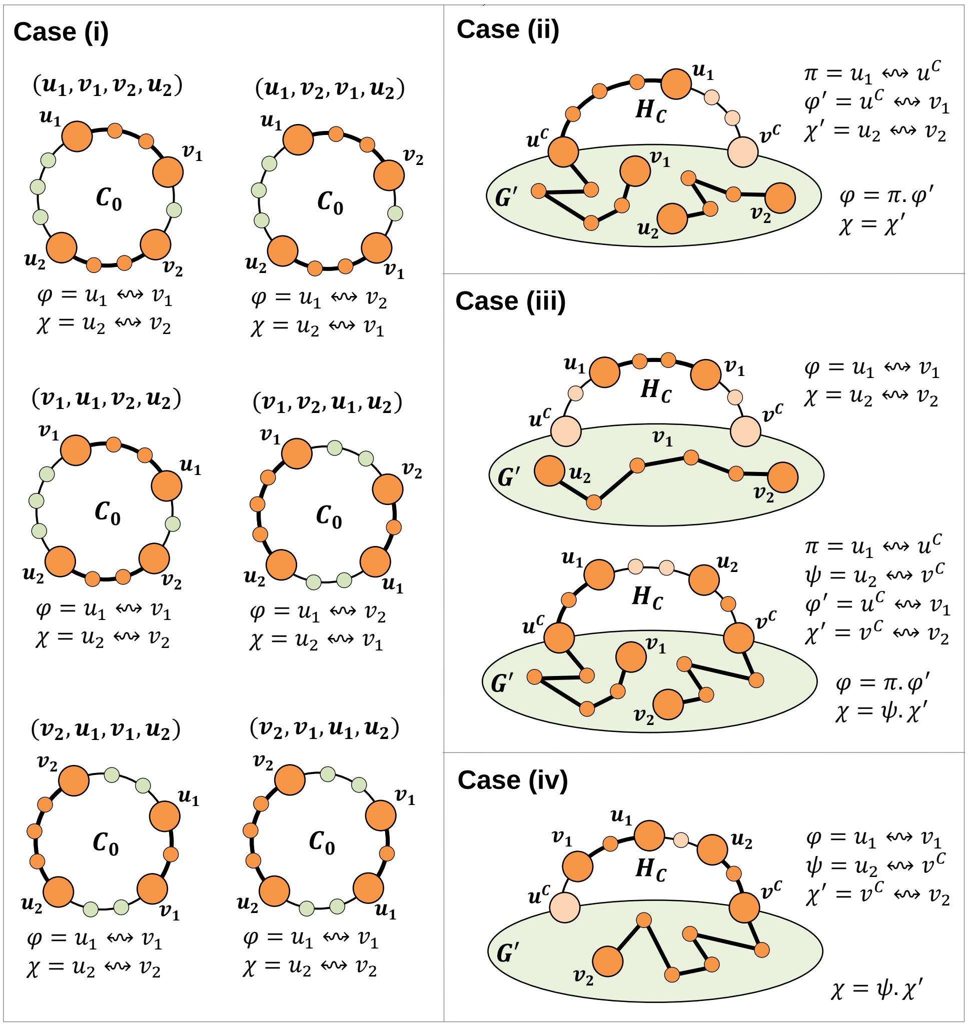

Case (i)
(u_1, v_1, v_2, u_2)
C_0
$\varphi = u_1 \leftrightsquigarrow v_1$
$\chi = u_2 \leftrightsquigarrow v_2$
(u_1, v_2, v_1, u_2)
$\varphi = u_1 \leftrightsquigarrow v_2$
$\chi = u_2 \leftrightsquigarrow v_1$
(v_1, u_1, v_2, u_2)
$\varphi = u_1 \leftrightsquigarrow v_1$
$\chi = u_2 \leftrightsquigarrow v_2$
(v_1, v_2, u_1, u_2)
$\varphi = u_1 \leftrightsquigarrow v_2$
$\chi = u_2 \leftrightsquigarrow v_1$
(v_2, u_1, v_1, u_2)
$\varphi = u_1 \leftrightsquigarrow v_1$
$\chi = u_2 \leftrightsquigarrow v_2$
(v_2, v_1, u_1, u_2)
$\varphi = u_1 \leftrightsquigarrow v_1$
$\chi = u_2 \leftrightsquigarrow v_2$
Case (ii)
H_C
G'
u^C
v^C
$\pi = u_1 \leftrightsquigarrow u^C$
$\varphi' = u^C \leftrightsquigarrow v_1$
$\chi' = u_2 \leftrightsquigarrow v_2$
$\varphi = \pi.\varphi'$
$\chi = \chi'$
Case (iii)
$\varphi = u_1 \leftrightsquigarrow v_1$
$\chi = u_2 \leftrightsquigarrow v_2$
$\pi = u_1 \leftrightsquigarrow u^C$
$\psi = u_2 \leftrightsquigarrow v^C$
$\varphi' = u^C \leftrightsquigarrow v_1$
$\chi' = v^C \leftrightsquigarrow v_2$
$\varphi = \pi.\varphi'$
$\chi = \psi.\chi'$
Case (iv)
$\varphi = u_1 \leftrightsquigarrow v_1$
$\psi = u_2 \leftrightsquigarrow v^C$
$\chi' = v^C \leftrightsquigarrow v_2$
$\chi = \psi.\chi'$

Fig. 4. *An illustration of the existence of* ***two vertex disjoint paths*** *connecting two pairs of vertices in a bi-connected graph.* The figure illustrates the case analysis from the proof of Lemma 2 which states that there exist two vertex disjoint paths φ and χ connecting a pair of vertices u_1 and u_2 with a pair of vertices v_1 and v_2 in a bi-connected graph G. The proof proceeds as mathematical induction according to the size of the handle decomposition of the graph G.

Case (ii): Let the 4-tuple of vertices (u_1, u_2, v_1, v_2) is assigned a sorted 4-tuple (C, c_2, c_3, c_4) where $C > c_2 \wedge C > c_3 \wedge C > c_4$. Using the interchangeability of vertices u_1, u_2, v_1, v_2, it is possible to suppose that $c_{\mathcal{D}}(u_1) = C$ without loss of generality. Let $H_C =$

$[u^C, w_1^C, w_2^C, \ldots, w_{l_C}^C, v^C]$, then there exists a path π connecting u_1 and u_C consisting of the internal vertices of L_C. Since the sorted 4-tuple $(c_{\mathcal{D}}(u^C), c_{\mathcal{D}}(u_2), c_{\mathcal{D}}(v_1), c_{\mathcal{D}}(v_2))$ is lexicographically strictly less than (C, c_2, c_3, c_4), the induction hypothesis implies that the lemma holds for the 4-tuple of vertices u^C, u_2, v_1, v_2 and the graph G without the internal vertices of the handle H_C; let this smaller graph be denoted as G'. That is either (a) or (b) holds in G'. Without loss of generality, suppose that (a) holds. Then there exist vertex disjoint paths φ' and χ' connecting u^C with v_1 and u_2 with v_2 in G' respectively. The path π is vertex disjoint with χ' and it shares exactly one vertex u^C with φ. Let φ be a path formed by the concatenation of π with φ' (the vertex u^C is used only once) and let $\chi = \chi'$. Then φ and χ are vertex disjoint paths substantiating the claim (a) for 4-tuple of vertices u_1, u_2, v_1, v_2 in G. See Fig. 4 for detailed illustration of the case.

Case (iii): The next case is that the 4-tuple of vertices (u_1, u_2, v_1, v_2) is assigned a sorted 4-tuple (C, C, c_3, c_4) where $C > c_3 \wedge C > c_4$. Again using the interchangeability of vertices only some of all the case are interesting. The first case is that $c_{\mathcal{D}}(u_1) = C$ and $c_{\mathcal{D}}(v_1) = C$ (that is, a pair of vertices to connect is within the handle H_C) and the second case is that $c_{\mathcal{D}}(u_1) = C$ and $c_{\mathcal{D}}(u_2) = C$ (that is, one vertex of a pair to connect is within the handle and the other is outside the internal vertices of the handle). In the first case, it is sufficient to construct a path φ connecting u_1 and v_1 consisting of the internal vertices of H_C and a path χ connecting u_2 and v_2 in G' (G' is a connected graph). The constructed paths φ and χ are vertex disjoint and hence they substantiate the claim (a) of the lemma. In the second case, it is necessary to distinguish between two juxtapositions of u_1 and u_2 within H_C with respect to the positive orientation of the handle: (u_1, u_2) and (u_2, u_1). In the case of juxtaposition (u_1, u_2), a path π connecting u_1 and u^C and a path ψ connecting u_2 and v^C are constructed (with the exception of u^C and v^C only the internal vertices of L_C are used). The second juxtaposition just interchanges u_1 and u_2. The sorted 4-tuple $(c_{\mathcal{D}}(u^C), c_{\mathcal{D}}(v_C), c_{\mathcal{D}}(v_1), c_{\mathcal{D}}(v_2))$ is lexicographically strictly less than (C, C, c_3, c_4), hence the lemma holds for the 4-tuple of vertices u^C, v^C, v_1, v_2 in the graph G'. Without loss of generality suppose that the case (a) holds; that is, there exists a path φ' that connects u^C with v_1 in G' and a path χ' that connects v^C with v_2 in G' while φ' and χ' are vertex disjoint. Observe, that π and ψ are vertex disjoint as well. It is sufficient to set a path φ to be a concatenation of π and φ' and a path χ to be a concatenation of ψ and χ'. Then φ and χ are the paths substantiating the claim (a) of the lemma for the 4-tuple of vertices u_1, u_2, v_1, v_2 in G. Again, see Fig. 4 for the detailed illustration of the case.

Case (iv): Let the 4-tuple of vertices (u_1, u_2, v_1, v_2) is assigned a sorted 4-tuple (C, C, C, c_4) where $C > c_4$. Without loss of generality, suppose that $c_{\mathcal{D}}(v_2) = c_4$. Then the following interesting juxtapositions of vertices u_1, u_2, and v_1 within the handle L_C with respect to the positive orientation can occur: (v_1, u_1, u_2), (u_1, v_1, u_2), and (u_1, u_2, v_1) (interchangeability of u_1 and u_2 is used to rule out the second half of juxtapositions). All the cases can be treated in the same way, thus it is sufficient to show only one case – for example the case of (v_1, u_1, u_2). Let φ be a path connecting v_1 and u_1 consisting of the internal vertices of the handle H_C. Next, let ψ be a path connecting u_2 with v^C that uses internal vertices of the handle H_C and the vertex v^C. Let χ' be a path connecting v^C and v_2 in G' (such a path exists since G' is a connected graph). Observe, that φ is vertex disjoint with ψ as well as with χ'. Thus, if χ is set to be a concatenation of ψ and χ', then φ and χ substantiate the claim (a) of the lemma for the 4-tuple of vertices u_1, u_2, v_1, v_2 and the graph G. Again, see Fig.4 for illustration of the case.

Case (v): The last case occurs if a sorted 4-tuple (C, C, C, C) where $C > 1$ is assigned to the 4-tuple of vertices (u_1, u_2, v_1, v_2). This case reduces to the case with all the vertices of the input 4-tuple located within the original cycle of the handle decomposition. However, instead of the original cycle a cycle $C(H_C)$ should be used. ∎

4.2 Pseudo-code of the *BIBOX* Algorithm

Several primitives are introduced to express the *BIBOX* algorithm in an easier way. Except functions S_P^0 and S_P^+ there is a function $S_P: P \longrightarrow V$ that represents the current arrangement of pebbles in the graph. Additionally, there are functions $\Phi_P^0: V \longrightarrow P \cup \{\perp\}$, $\Phi_P^+: V \longrightarrow P \cup \{\perp\}$, and $\Phi_P: V \longrightarrow P \cup \{\perp\}$ which are generalized inverses of S_P^0, S_P^+, and S_P respectively with the symbol $\perp$ is used to represent an unoccupied vertex (that is, $(\forall p \in P)\Phi_P(S_P(p)) = p$ and $\Phi_P(\perp) = \perp$ if $(\forall p \in P)S_P(p) \neq v$). Next, each undirected cycle appearing in the handle decomposition of the input graph is assigned a fixed orientation. Let C be an undirected cycle (a set of vertices of the cycle), then the orientation of C is expressed by functions $next_{\circlearrowright}$ and $prev_{\circlearrowright}$ where $next_{\circlearrowright}(C, v)$ for $v \in C$ is the vertex following v (with respect to positive orientation) in the cycle C and $prev_{\circlearrowright}(C, v)$ is the vertex preceding v (with respect to positive orientation). The orientation of a cycle given by $next_{\circlearrowright}$ and $prev_{\circlearrowright}$ is respected as well when vertices of the cycle are explicitly enumerated in the code. Auxiliary operations *Lock* (X) and *Unlock*(X) locks or unlocks a set of vertices X. Each vertex of the input graph is either locked or unlocked. The state of a vertex is used to determine whether a pebble can move into a vertex. Typically, a pebble is not allowed to enter a locked vertex (see the pseudo-code for details). Finally, there is assumed a potentially infinite sequence of functions $S_P^0, S_P^1, S_P^2, \ldots$ which finite prefix is used to form a solution. Actually, these variables are not needed to be stored in the memory; the output solution can be directly printed to the output. For convenience, several variables such as those representing handle decomposition are global; that is, they are shared among all the functions and procedures in the pseudo-code.

It is assumed that for the number of pebbles it holds that $|V| = \mu - 2$, where $|P| = \mu$ (that is, there are exactly two unoccupied vertices in the graph G). Furthermore, it is required for the successful progression of the algorithm that the unoccupied vertices within the goal arrangement are located in the first two vertices of the original cycle (according to the positive orientation) of the handle decomposition. This requirement is treated by a function *Transform-Goal* and a procedure *Finish-Solution*. The function *Transform-Goal* determines two vertex disjoint paths from unoccupied vertices in the goal arrangement to first two vertices in the original cycle of the handle decomposition. Since the unoccupied vertices are indifferent, it does not matter what unoccupied vertex is associated with the first or with the second vertex of the initial cycle. Thus, preconditions of Lemma 2 are satisfied and hence the existence of mentioned two vertex disjoint paths is ensured.

The goal arrangement is changed by the function *Transform-Goal* so that finally unoccupied vertices are located in the original cycle. This is done by shifting pebbles within the goal arrangement along the two determined paths. After the modified instance is solved, the function *Finish-Solution* moves unoccupied vertices back to their goal positions given by the original unmodified goal arrangement. This final placement of unoccupied vertices is done by shifting pebbles along the two paths determined by the function *Transform-Goal* in the opposite direction.

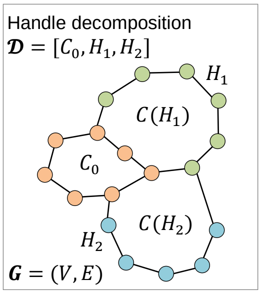
Several upper level primitives are exploited by the *BIBOX* algorithm. It is possible to make any vertex unoccupied in a connected graph (especially in a bi-connected graph).

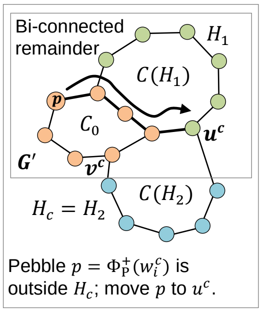
The process of making a given vertex unoccupied is implemented by a procedure *Make-Unoccupied*. Let v be a vertex to be made unoccupied. A path ϕ connecting v and some of the unoccupied vertices avoiding the locked vertices is found. Then pebbles along the path ϕ are shifted using swapping pebbles towards the currently unoccupied vertex.

Algorithm 1. *The* ***BIBOX*** *algorithm.* It solves a given pebble motion problem on a **non-trivial bi-connected** graph with exactly **two unoccupied** vertices. The algorithm proceeds inductively according to the handle decomposition of the graph of the input instance. The two unoccupied vertices are necessary for arranging pebbles within the original cycle of the handle decomposition.

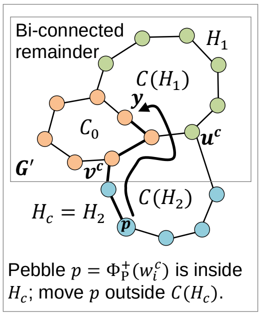
```
function BIBOX-Solve($G = (V, E), P, S_P^0, S_P^+$) : pair
    /* Top level function of the BIBOX algorithm; solves
    a given problem of pebble motion on a graph.
    Parameters:    $G$ - a graph modeling the environment,
                   $P$ - a set of pebbles,
                   $S_P^0$ - a initial arrangement of pebbles,
                   $S_P^+$ - a goal arrangement of pebbles. */
1:  let $\mathcal{D} = [C_0, H_1, H_2, \ldots, H_d]$ be a handle decomposition of $G$
2:  $(S_P^+, \varphi, \chi) \leftarrow$ Transform-Goal$(G, P, S_P^+)$
3:  $S_P \leftarrow S_P^0$
4:  $\xi \leftarrow 1$
5:  for $c = d, d-1, \ldots, 1$ do
6:  |   if $|H_c| > 2$ then
7:  |   |   Solve-Regular-Handle$(c)$
8:  Solve-Original-Cycle
9:  Finish-Solution$(\varphi, \chi)$
10: return $(\xi, [S_P^0, S_P^1, \ldots, S_P^\xi])$

procedure Solve-Regular-Handle($c$)
    /* Places pebbles which destinations are within a
    handle $H_c$; pebbles placed in the handle $H_c$ are finally
    locked so they cannot move any more.
    Parameters:    $c$ – the index of a handle */
1:  let $[u^j, w_1^j, w_2^j, \ldots, w_{h_c}^j, v^j] = H_j \ \forall j \in \{1,2,\ldots,d\}$
    /* Both unoccupied vertices must be located
    outside the currently solved handle. */
2:  let $w, z \in V \setminus \bigcup_{j=c}^{d}(H_j \setminus \{u^j, v^j\})$ such that $w \neq z$
3:  Make-Unoccupied$(w)$
4:  Lock $(\{w\})$
5:  Make-Unoccupied$(z)$
6:  Unlock $(\{w\})$
7:  for $i = h_c, h_c - 1, \ldots, 1$ do
8:  |   Lock$(H_c \setminus \{u^c, v^c\})$
    |   /* A pebble to be placed is outside the handle $H_c$. */
9:  |   if $S_P(\Phi_P^+(w_i^c)) \notin (H_c \setminus \{u^c, v^c\})$ then
10: |   |   Move-Pebble$(\Phi_P^+(w_i^c), u^c)$
11: |   |   Lock$(\{u^c\})$
```



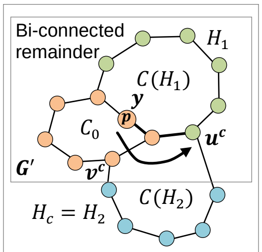


Pebble $p = \Phi_P^+(w_i^c)$ is outside H_c; move p to u^c.

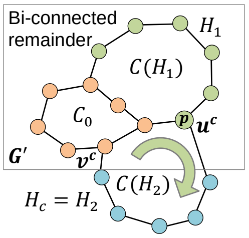


Pebble $p = \Phi_P^+(w_i^c)$ is inside H_c; move p outside $C(H_c)$.

```
12: | | Make-Unoccupied($v^c$)
13: | | Unlock($H_c$)
14: | | Rotate-Cycle$^+$($C(H_c)$)
    | /* A pebble to be placed is inside the handle $H_c$. */
15: | else
16: | | Make-Unoccupied($u^c$)
17: | | Unlock($H_c$)
18: | | $\rho \leftarrow 0$
19: | | while $S_P(\Phi_P^+(w_i^c)) \neq v^c$ do
20: | | | Rotate-Cycle$^+$($C(H_c)$)
21: | | | $\rho \leftarrow \rho + 1$
22: | | Lock($H_c \setminus \{u^c, v^c\}$)
23: | | let $y \in V \setminus (\bigcup_{j=c+1}^{d}(H_j \setminus \{u^j, v^j\}) \cup C(H_c))$
24: | | Move-Pebble($\Phi_P^+(w_i^c), y$)
25: | | Lock ($\{y\}$)
26: | | Make-Unoccupied($u^c$)
27: | | Unlock($H_c$)
28: | | while $\rho > 0$ do
29: | | | Rotate-Cycle$^-$($C(H_c)$)
30: | | | $\rho \leftarrow \rho - 1$
31: | | Unlock($\{y\}$)
32: | | Lock($H_c \setminus \{u^c, v^c\}$)
33: | | Move-Pebble($\Phi_P^+(w_i^c), u^c$)
34: | | Lock ($\{u^c\}$)
35: | | Make-Unoccupied($v^c$)
36: | | Unlock($H_c$)
37: | | Rotate-Cycle$^+$($C(H_c)$)
38: Lock($H_c \setminus \{u^c, v^c\}$)
```

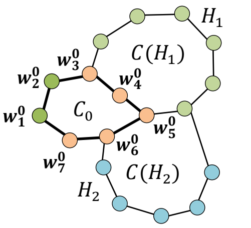


Pebble $p = \Phi_P^+(w_i^c)$ outside $C(H_c)$; $C(H_c)$ rotated back; move p into connection u^c.



Pebble $p = \Phi_P^+(w_i^c)$ in u^c; rotate $C(H_c)$ once forward.

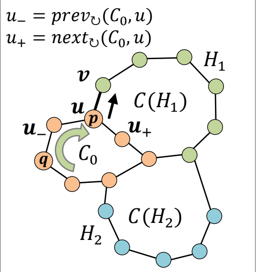
```
procedure Solve-Original-Cycle
    /* Places pebbles which destinations are within the original cycle;
    it is assumed that unoccupied vertices of the goal arrangement of
    pebbles are located within the original cycle. */
1:  let $u \in C_0$ and $v \in V \setminus C_0$ such that $\{u, v\} \in E$
2:  let $[w_1^0, w_2^0, \ldots, w_l^0] = C_0$
    /* According to the assumption on the goal arrangement
    it holds that $\Phi_P^+(w_1^0) = \perp$ and $\Phi_P^+(w_2^0) = \perp$. */
3:  for $i = 3, 4, \ldots, l$ do
4:  | Make-Unoccupied($w_1^0$)
5:  | Lock($\{w_1^0\}$)
6:  | Make-Unoccupied($w_2^0$)
7:  | Unlock($\{w_1^0\}$)
8:  | if $\Phi_P^+(w_i^0) \neq \Phi_P(w_i^0)$ then
9:  | | Exchange-Pebbles ($\Phi_P^+(w_i^0), \Phi_P(w_i^0), u, v$)
10: Make-Unoccupied($w_1^0$)
11: Lock($\{w_1^0\}$)
12: Make-Unoccupied($w_2^0$)
13: Unlock($\{w_1^0\}$)
```

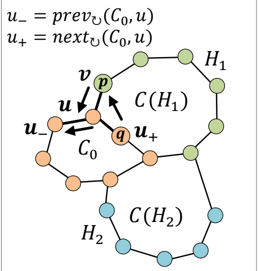


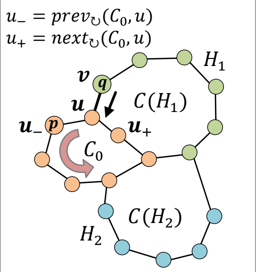
Move w_1^0 and w_2^0 are made unoccupied.

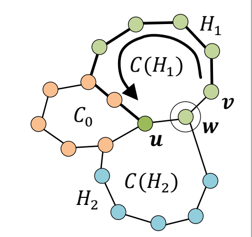
```
**procedure** *Exchange-Pebbles*$(p, q, u, v)$
    /* Exchanges a pair of pebbles within the initial
    cycle of the handle decomposition.
    Parameters:    $p, q$ - a pair of pebbles to be exchanged,
                   $u, v$ - a pair of neighboring vertices where
                   $v$ is used as a storage space. */
1:  $r \leftarrow \Phi_P(v)$
2:  Make-Unoccupied$(u)$
3:  Swap-Pebbles-Unoccupied$(v, u)$
4:  **while** $S_P(p) \neq u$ **do**
5:  |   Rotate-Cycle$^+(C_0)$
6:  Swap-Pebbles-Unoccupied$(u, v)$
7:  Lock $(\{u\})$
8:  Make-Unoccupied$(next_{\circlearrowright}(C_0, u))$
9:  Unlock $(\{u\})$
    /* Subsequent rotation must use $u$
    as the unoccupied vertex. */
10: Lock$(C_0 \setminus \{u\})$
11: $\rho \leftarrow 0$
12: **while** $S_P(q) \neq next_{\circlearrowright}(C_0, u)$ **do**
13: |   Rotate-Cycle$^+(C_0)$
14: |   $\rho \leftarrow \rho + 1$
15: Swap-Pebbles-Unoccupied$(v, u)$
16: Unlock $(C_0 \setminus \{u\})$
17: Make-Unoccupied$(prev_{\circlearrowright}(C_0, u))$
18: Swap-Pebbles-Unoccupied$(u, prev_{\circlearrowright}(C_0, u))$
19: Swap-Pebbles-Unoccupied$(next_{\circlearrowright}(C_0, u), u)$
20: Swap-Pebbles-Unoccupied$(u, v)$
21: Unlock$(\{u\})$
22: Lock$(C_0 \setminus \{u\})$
23: **while** $\rho > 0$ **do**
24: |   Rotate-Cycle$^-(C_0)$
25: |   $\rho \leftarrow \rho - 1$
26: Swap-Pebbles-Unoccupied$(v, u)$
27: **while** $S_P(r) \neq u$ **do**
28: |   Rotate-Cycle$^+(C_0)$
29: Swap-Pebbles-Unoccupied$(u, v)$
30: Unlock$(C_0)$
```

```
**procedure** *Make-Unoccupied*$(v)$
    /* Makes a vertex $v$ unoccupied while locked
    vertices remain untouched.
    Parameters:    $v$ - a vertex to be made unoccupied. */
1:  **let** $u \in V$ such that $\Phi_P(u) = \perp$ and $u$ is not locked
2:  **let** $\phi = [u = w_1, w_2, \dots, w_j = v]$ be a (shortest) path
3:  |   connecting $u$ and $v$ in $G$ not containing locked vertices
4:  **for** $i = 1, 2, \dots, j - 1$ **do**
5:  |   Swap-Pebbles-Unoccupied$(w_{i+1}, w_i)$
```


$u_- = prev_{\circlearrowright}(C_0, u)$
$u_+ = next_{\circlearrowright}(C_0, u)$
H_1
v
u
p
$C(H_1)$
u_-
u_+
q
C_0
$C(H_2)$
H_2
Move p into v; rotate C_0 forward such that q appears in u_+.

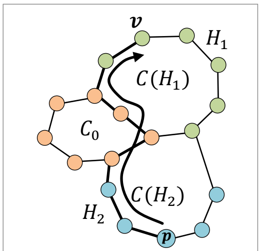

$u_- = prev_{\circlearrowright}(C_0, u)$
$u_+ = next_{\circlearrowright}(C_0, u)$
H_1
v
p
u
$C(H_1)$
u_-
q
u_+
C_0
$C(H_2)$
H_2
Exchange p and q (p appears in u_-).


$u_- = prev_{\circlearrowright}(C_0, u)$
$u_+ = next_{\circlearrowright}(C_0, u)$
H_1
v
q
u
$C(H_1)$
u_-
p
u_+
C_0
$C(H_2)$
H_2
Rotate C_0 back; move q from v to u.



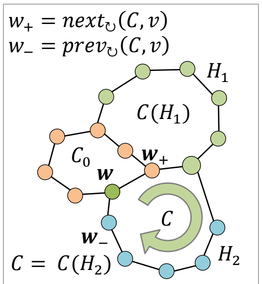
Vertex w is locked; u is unoccupied; pebbles are shifted along cycle $C(H_1)$ to make v unoccupied.

```
procedure Move-Pebble$(p,v)$
    /* Moves a pebble $p$ into a vertex $v$ avoiding locked vertices.
    Parameters:   $p$ - a pebble to move,
                  $v$ - a target vertex.*/
    /* complexity issues impose special selection of $\varphi$ */
1:  let $\varphi = [S_P(p) = w_1^\varphi, w_2^\varphi, \dots, w_{j_\varphi}^\varphi = v]$ be a path
2:  |   connecting $S_P(p)$ and $v$ in $G$ not containing
3:  |   locked vertices such that an alternative vertex
4:  |   disjoint path $\chi = [S_P(p) = w_1^\chi, w_2^\chi, \dots, w_{j_\chi}^\chi = v]$
5:  |   not containing locked vertices exists
6:  for $i = 1,2,\dots,j_\varphi - 1$ do
7:  |   Lock $(\{w_i^\varphi\})$
8:  |   Make-Unoccupied$(w_{i+1}^\varphi)$
9:  |   Unlock$(\{w_i^\varphi\})$
10: |   Swap-Pebbles-Unoccupied$(w_i^\varphi, w_{i+1}^\varphi)$
```

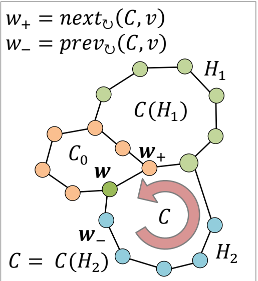


Pebble p is moved to v through cycles $C(H_2)$, C_0, and $C(H_1)$.

```
procedure Rotate-Cycle$^+(C)$
    /* Rotates pebbles in a cycle $C$ in the positive direction;
    the vertex locking mechanism allows to select which one of
    unoccupied vertices should be used. At least one unlocked
    unoccupied vertex must be located in $C$.
    Parameters:   $C$ - a cycle to rotate. */
1:  let $w \in C$ such that $\Phi_P(w) = \perp$ and $w$ is not locked
2:  for $i = 1,2,\dots,|C|$ do
3:  |   Swap-Pebbles-Unoccupied$(prev_\circlearrowright(C,w), w)$
4:  |   $w \leftarrow prev_\circlearrowright(C,w)$
```


$w_+ = next_\circlearrowright(C,v)$
$w_- = prev_\circlearrowright(C,v)$
H_1
$C(H_1)$
C_0
w_+
w
C
w_-
H_2
$C = C(H_2)$
Vertex w is unoccupied; $C(H_2)$ is rotated in the positive direction.

```
procedure Rotate-Cycle$^-(C)$
    /* Rotates pebbles in the cycle $C$ in the negative direction;
    again an unoccupied vertex to use can be selected by the
    vertex locking mechanism. At least one unlocked
    unoccupied vertex must be located in $C$.
    Parameters:   $C$ - a cycle to rotate. */
1:  let $w \in C$ such that $\Phi_P(w) = \perp$ and $w$ is not locked
2:  for $i = 1,2,\dots,|C|$ do
3:  |   Swap-Pebbles-Unoccupied$(next_\circlearrowright(C,w), w)$
4:  |   $w \leftarrow next_\circlearrowright(C,w)$
```


$w_+ = next_\circlearrowright(C,v)$
$w_- = prev_\circlearrowright(C,v)$
H_1
$C(H_1)$
C_0
w_+
w
C
w_-
H_2
$C = C(H_2)$
Vertex w is unoccupied; $C(H_2)$ is rotated in the negative direction.

```
procedure Swap-Pebbles-Unoccupied$(u,v)$
    /* Swaps pebbles in vertices $u$ and $v$;
    vertex $v$ is supposed to be unoccupied.
    Parameters:   $u, v$ – vertices in which pebbles
                  are swapped. */
1:  $S_P(\Phi_P(u)) \leftarrow v$
2:  $\Phi_P(v) \leftarrow \Phi_P(u)$
3:  $\Phi_P(u) \leftarrow \perp$
4:  $S_P^\xi \leftarrow S_P$
5:  $\xi \leftarrow \xi + 1$
```

An operation of swapping pebbles itself is implemented using a procedure *Swap-Pebbles-Unoccupied*. The procedure moves a pebble into a neighboring unoccupied vertex and the next member S_P^{ξ} of the output solution sequence is constructed together with the update of functions S_P and Φ_P according to the new arrangement of pebbles.

The next important process is moving a pebble into a given target vertex. This is implemented by a procedure *Move-Pebble*. Let a pebble p is moved to a vertex v. A path φ is found such that is connects vertices $S_P(p)$ (which is a vertex currently occupied by p) and v and there exists an alternative vertex disjoint path χ connecting the same pair of vertices. The existence of the alternative is ensured by Lemma 2. Indeed, the proof of Lemma 2 provides the construction such a path. Parameters of the lemma should be set as follows: u_1 is $S_P(p)$, u_2 is a neighboring vertex to u_1 in the same bi-connected component, v_1 is v, and v_2 is a neighboring vertex to v_1 in the same bi-connected component (it can be determined which of the neighboring vertices belong into the same bi-connected component from the knowledge of the handle decomposition). Vertex disjoint paths φ' and χ' resulting from Lemma 2 together with edges $\{u_1, u_2\}$ and $\{v_1, v_2\}$ form the required φ and χ. In case (a): $\varphi = \varphi'$ and $\chi = [u_1].\chi.[u_2]$; in case (b): $\varphi = [u_1].\chi'$ and $\chi = \varphi'.[u_2]$.

Subsequently, edges of φ are traversed in the following way. The first vertex of the edge is locked so paths to be searched must avoid this vertex. An invariant holds, that p is located in the first vertex of the edge at the beginning of each traversal step and thus it cannot move. Then the second vertex of the edge is made unoccupied (the alternative path χ is used for this task); the first vertex of the edge is unlocked and the pebble p is moved to the second vertex of the edge which is now unoccupied.

The last basic operation exploited by the algorithm is a rotation of pebbles along a cycle. This operation is implemented by procedures *Rotate-Cycle*$^+$ and *Rotate-Cycle*$^-$. The former rotates pebbles in the positive direction and the latter rotates pebbles in the negative direction. It supposed the at least one vertex in the given input cycle is unoccupied. The rotation is done using an unlocked unoccupied vertex located in the cycle.

During movement of an unoccupied vertex and during movement of a pebble to another vertex, the arrangement of pebbles located in vertices that are locked is preserved while the arrangement of pebbles located in unlocked vertices is generally not preserved. This behavior helps to control finished parts of the goal arrangement. On the other hand, moving pebbles must be done in a precise way so that required unlocked paths always exist.

The process of placing pebbles according to the given goal arrangement will be described now using the primitives discussed above. Pebbles, which goal positions are within the currently solved handle, are placed in a stack like manner. This process is carried out by a procedure *Solve-Regular-Handle* (iteration through the handle is at lines 7-37). Let $H_c = [u^c, w_1^c, w_2^c, \ldots, w_{h_c}^c, v^c]$ for $c \in \{1,2,\ldots,d\}$ be a current handle. Suppose that a pebble which goal position is in w_i^c for $i \in \{1,2,\ldots,h_c\}$, that is a pebble $\Phi_P^+(w_i^c)$, is processed in the current iteration. Inductively suppose that pebbles $\Phi_P^+(w_{h_c}^c), \Phi_P^+(w_{h_c-1}^c), \ldots, \Phi_P^+(w_{i+1}^c)$ are located in vertices $w_{h_c-i-1}^c, w_{h_c-i-2}^c, \ldots, w_1^c$ respectively. An analogical situation for the next pebble $\Phi_P^+(w_{i+1}^c)$ must be produced at the end of the iteration.

The pebble $\Phi_P^+(w_i^c)$ is moved to the vertex u^c and then the cycle $C(H_c)$ is positively rotated once which causes that the pebble $\Phi_P^+(w_i^c)$ moves to w_1^c and pebbles $\Phi_P^+(w_{h_c}^c), \Phi_P^+(w_{h_c-1}^c), \ldots, \Phi_P^+(w_{i+1}^c)$ plunge in the cycle so that they are located in $w_{h_c-i}^c$, $w_{h_c-i-1}^c, \ldots, w_2^c$. The described process represents one iteration of stacking pebbles into the handle H_c. However, the process is not that easy. At least, two major cases must be distinguished within this process. In both cases, the first step is that internal vertices of the handle H_c are locked (line 8 of *Solve-Regular-Handle*).

If the pebble $\Phi_P^+(w_i^c)$ is not located in the internal vertices of the handle H_c (line 9-14 of *Solve-Regular-Handle*), it is just moved to u^c. This is possible since an invariant holds that both unoccupied vertices are located outside the internal vertices of the handle and the graph without the internal vertices of the handle is connected. This holds at the beginning, since both unoccupied vertices are explicitly moved outside the handle H_c (lines 2-6 of *Solve-Regular-Handle*) and it is preserved through all the iterations. Observe that these movements do not affect pebbles already stacked in the handle. The pebble $\Phi_P^+(w_{h_c}^c)$ is fixed in u^c by locking u^c and then an unoccupied vertex is moved to v^c which makes the rotation of the cycle $C(H_c)$ possible. The positive rotation of $C(H_c)$ finishes the iteration.

If the pebble $\Phi_P^+(w_i^c)$ is already located in some of the internal vertices of the handle H_c (lines 15-37 of *Solve-Regular-Handle*), the above process is reused but it must be preceded by getting the pebble $\Phi_P^+(w_{h_c}^c)$ outside the handle. Notice, that it is not possible for the pebble $\Phi_P^+(w_i^c)$ to intermix with already stacked pebbles $\Phi_P^+(w_{h_c}^c), \Phi_P^+(w_{h_c-1}^c), \ldots, \Phi_P^+(w_{i+1}^c)$. The vertex u^c is made unoccupied and the cycle $C(H_c)$ is positively rotated until the pebble $\Phi_P^+(w_i^c)$ gets outside the internal nodes of H_c; that is, $\Phi_P^+(w_i^c)$ appears in v^c. This series of rotations preserves the order of the already stacked pebbles. To restore the situation however, the cycle must be rotated back the same number of times. A vertex w outside the already finished part of the graph (that is outside $C(H_c)$ and outside H_j for $j > c$) is selected; the pebble $\Phi_P^+(w_i^c)$ is moved into w and it is fixed there by locking. The vertex u^c is made unoccupied again since the preceding process may move some pebble into it (this is possible since w alone cannot rule out the existence of a path from an unoccupied vertex to u^c in the bi-connected graph; there is always an alternative path). The cycle is rotated back so that inductively supposed placement of $\Phi_P^+(w_{h_c}^c), \Phi_P^+(w_{h_c-1}^c), \ldots, \Phi_P^+(w_{i+1}^c)$ is restored. The situation is now the same as in the previous case with $\Phi_P^+(w_i^c)$ outside the handle.

After the last iteration within the handle H_c it holds that the pebbles $\Phi_P^+(w_{h_c}^c), \Phi_P^+(w_{h_c-1}^c), \ldots, \Phi_P^+(w_1^c)$ are located in vertices $w_{h_c}^c$, $w_{h_c-1}^c, \ldots, w_1^c$ respectively. Moreover it holds that unoccupied vertices are both outside the internal vertices of H_c. Thus, the solving process can continue with the next handle in the same way while the already solved handles remain unaffected by subsequent steps. Notice, that only one unoccupied vertex is sufficient for stacking pebbles into handles.

The initial cycle C_0 of the handle decomposition must be treated in a different way. Here, the second unoccupied vertex is utilized. An arrangement of pebbles within C_0 can be regarded as a permutation. The task is to obtain the right permutation corresponding to the goal arrangement. This can be achieved by exchanging several pairs of pebbles. More precisely, if a pebble residing in a vertex of C_0 differs from a pebble that should reside in this vertex in the goal arrangement, this pair of pebbles is exchanged. The process is implemented by a procedure *Solve-Original-Cycle* and by auxiliary procedure *Exchange-Pebbles* for exchanging a pair of pebbles.

The procedure *Exchange-Pebbles* expects that first two vertices of the initial cycle are unoccupied in the current arrangement. However, the function generally does not preserve this property. Hence, the vacancy of the first two vertices of the initial cycle must be repeatedly restored (lines 4-7 and 10-13 of *Solve-Original-Cycle*). The process of exchanging a pair of pebbles p and q itself exploits a pair of vertices u and v which are connected by an edge and $u \in C_0 \wedge v \notin C_0$. The vertex v is used as a storage place. The need of two unoccupied vertices is imposed by the fact that a pebble from C_0 to be stored in v must be rotated into u first. During this process, some vertex of the cycle must be unoccupied to make the rotation possible and the vertex v must be unoccupied as well to make storing possible.

When exchanging the pair o pebbles p and q it is necessary to preserve ordering of the remaining pebbles. First, a pebble occupying the vertex v is moved into the cycle C_0 in order to make v vacant (lines 1-3 of *Exchange-Pebbles*). Then the cycle is rotated until the pebble p appears in u (since there was a pebble in u at the beginning of the rotation, there is always some pebble in u after all the rotations) and the pebble p is stored in v (lines 4-6 of *Exchange-Pebbles*). Next, the cycle C_0 is rotated positively so that q appears in $next_{\circlearrowright}(C_0, u)$ (the next vertex to u with respect to the positive orientation) while the number of rotations is recorded (lines 7-14 of *Exchange-Pebbles*). However, the second unoccupied vertex must not interfere with counting of rotations, thus it is located $next_{\circlearrowright}(C_0, u)$ at the beginning (that is, outside the sequence of pebbles between p and q which length is being counted in fact) and then moved to $prev_{\circlearrowright}(C_0, u)$ in the positive direction (the movement of the second unoccupied in the negative direction is not possible here, since u is locked at the moment). At this moment, pebbles p and q are exchanged using two unoccupied vertices so that ordering of p in the cycle C_0 is the same as of q before the exchange (lines 15-20 of *Exchange-Pebbles*). Then, the cycle is rotated in the negative direction recorded number of times so that place within the cycle where p was originally ordered appears in u; thus q is ordered here (lines 21-26 of *Exchange-Pebbles*). Finally, the pebble that was located in v before the exchange of pebbles p and q has been commenced is put back into v (lines 27-30 of *Exchange-Pebbles*). Since the process of exchange of a pair of pebbles is quite complicated, the detailed case analysis is given within the proof of correctness of the process in (Surynek, 2010c).

4.3 Summary of Properties of the *BIBOX* Algorithm

Several theoretical properties of the *BIBOX* algorithm regarding the time and space complexity are summarized in the following propositions. Propositions are presented without proofs. The detailed proofs can be found in (Surynek, 2010c). Nevertheless, it can be briefly stated that the algorithm is polynomial in all the aspects.

Proposition 2 *(BIBOX - soundness and completeness).* The *BIBOX* algorithm is sound and complete. That is, the algorithm always terminates and produces a solution of a given input instance of the problem of pebble motion on a graph $\Pi = (G = (V, E), P, S_P^0, S_P^+)$. ■

Proposition 3 *(BIBOX – worst case time complexity).* The worst case time complexity of the *BIBOX* algorithm is $O(|V|^3)$ with respect to an input instance of the problem pebble motion on a graph $\Pi = (G = (V, E), P, S_P^0, S_P^+)$. ■

Proposition 4 *(BIBOX – makespan of the solution).* The makespan of a solution in the worst case produced by the *BIBOX* algorithm (that is, the number ξ) for an input instance of the problem of pebble motion on a graph $\Pi = (G = (V, E), P, S_P^0, S_P^+)$ is $O(|V|^3)$. ■

Proposition 5 *(BIBOX – worst case space complexity).* The worst case space complexity of the *BIBOX* algorithm is $O(|V| + |E|)$ with respect to an input instance of the problem pebble motion on a graph $\Pi = (G = (V, E), P, S_P^0, S_P^+)$. ■

4.4 Extensions and the Real-life Implementation

The natural question is how to apply the *BIBOX* algorithm if there are more than two unoccupied vertices in input instance (that is, $\mu - 2 \leq |V|$). The algorithm can be used directly if the graph is filled by dummy pebbles. The instance with dummy pebbles is solved by the algorithm as it is and finally movements of dummy pebbles are filtered out from the solution in an additional post-processing step. An adaptation of the solving algorithm for sparse instances of the pebble motion problem is out of scope of this chapter. Nevertheless, a straightforward adaptation is to replace the non-deterministic selection of an unlocked unoccupied vertex (such as that at line 1 of *Make-Unoccupied*) by the selection of the most promising one. For example, an unlocked unoccupied vertex that is nearest to the vertex that is to be made unoccupied can be selected.

Some further optimizations should be used in the real-life implementation to reduce the makespan of the produced solution. Various preconditions are explicitly enforced in order to make the presented pseudo-code simpler (for example, the precondition of having first two vertices of the initial cycle of the handle decomposition unoccupied before a pair of vertices is exchanged within the cycle - lines 4-6 of *Solve-Original-Cycle*). This approach should be avoided and a lazier approach should be adopted in the real-life implementation (in the case of exchanging pebbles, locations of unoccupied vertices should be detected implicitly in subsequent steps by more sophisticated branching of the code).

The real-life implementation of procedures *Solve-Regular-Handle* and *Solve-Original-Cycle* should use opportunistic selection of vertices to store pebbles (vertex y - line 23 of *Solve-Regular-Handle* and vertices u, v - line 1 of *Solve-Original-Cycle*). The nearest vertex to the target pebble should be always selected. Moreover, selection of these vertices within the procedure *Solve-Original-Cycle* should be done not only at the beginning but also in every iteration of its main loop.

5. Improving Makespan by Increasing Parallelism

This section is devoted to a method for increasing parallelism of solutions. In fact, this method represents a major technique how to utilize parallelism allowed by the definition of the problem of multi-robot path planning. The presented algorithm does not utilize the possibility of parallel movements. It solves the problem of pebble motion on a graph in fact. The method presented below is intended as a post-processing technique that should be applied on a solution produced by the *BIBOX* algorithm.

Definition 5 ***(sequential solution).*** A solution $\mathcal{S}_R(\Sigma) = [S_R^0, S_R^1, \dots, S_R^\zeta]$ of multi-robot path planning problem $\Sigma = (G = (V, E), R = \{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_\nu\}, S_R^0, S_R^+)$ is called ***sequential*** (ζ is the length of the solution) if for each $k = 1,2, \dots, \zeta - 1$ there exists $j \in \{1,2, \dots, \nu\}$ such that $S_R^k(\bar{r}_j) \neq S_R^{k+1}(\bar{r}_j)$ and $S_R^k(\bar{r}_l) \neq S_R^{k+1}(\bar{r}_l)$ for each $l = 1,2, \dots, \nu \wedge l \neq j$ (at time step k a robot $\bar{r}_j$ is moved; all the other robots do not move at the time step). □

A move of a robot r from a vertex u to a vertex v will be denoted using the notation $r: u \rightarrow v$. The sequential solution of multi-robot path planning problem can be equivalently represented as a sequence of moves of the form $r: u \rightarrow v$. That is, a sequence $\mathcal{S}_R^{\prec}(\Sigma) = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \dots, r_\zeta: u_\zeta \rightarrow v_\zeta]$ determines sequential solution (r_i are variables with the domain $\{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_\nu\}$; $\bar{r}_i$ are constants). Notice, that $u_k \neq v_k$ for each $k = 1,2, \dots, \zeta$ which is ensured by the definition of the sequential solution. In other words, a solution is sequential if there is just one move at each step. This, however, may prolong makespan significantly, which is not desirable.

Suppose a sequential solution $\mathcal{S}_R^{\prec}(\Sigma) = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \dots, r_\zeta: u_\zeta \rightarrow v_\zeta]$ of an instance of multi-robot path planning $\Sigma = (G = (V, E), R = \{\bar{r}_1, \bar{r}_2, \dots, \bar{r}_\nu\}, S_R^0, S_R^+)$. This form of the solution of the problem will be more convenient for reasoning about the possible parallelism. The following definitions refer will to $\mathcal{S}_R^{\prec}(\Sigma)$.

Definition 6 ***(interfering moves).*** A move $r_h: u_h \rightarrow v_h;\ h \in \{1,2,\dots,\zeta - 1\}$ is ***interfering*** with a move $r_k: u_k \rightarrow v_k;\ k \in \{1,2,\dots,\zeta - 1\}$ if $|\{u_h, v_h\} \cap \{u_k, v_k\}| \geq 1$. □

Typically, interfering moves cannot be executed in parallel. However, the situation is not so straightforward. Following definitions are trying to capture which pairs of interfering moves can be undoubtedly executed in parallel and which not.

Definition 7 ***(potentially concurrent moves).*** A move $r_k: u_k \rightarrow v_k;\ k \in \{1,2,\dots,\zeta\}$ is ***potentially concurrent*** with a move $r_h: u_h \rightarrow v_h;\ h \in \{1,2,\dots,\zeta\}$ with $h < k$ if $r_h \neq r_k,\ u_h = v_k \wedge v_h \neq u_k$, and there is no other move $r_\hbar: u_\hbar \rightarrow v_\hbar$ in $\mathcal{S}_R^{\prec}(\Sigma)$ such that $h < \hbar < k$ interfering with $r_h: u_h \rightarrow v_h$ or $r_k: u_k \rightarrow v_k$. The notation is that $r_h: u_h \rightarrow v_h \preccurlyeq r_k: u_k \rightarrow v_k$. □

The definition captures the fact that although the moves are interfering they can be executed at the same time step according to the definition of a solution of the instance of multi-robot path planning problem. The relation of potential concurrence is anti-reflexive due to the requirement on different robots involved ($r_h \neq r_k$) and anti-symmetric due to the ordering of moves within the sequential solution ($h < k$).

Definition 8 ***(trivially dependent moves).*** A move $r_k: u_k \rightarrow v_k;\ k \in \{1,2,\dots,\zeta\}$ is ***trivially dependent*** on a move $r_h: u_h \rightarrow v_h;\ h \in \{1,2,\dots,\zeta\}$ with $h < k$ if these moves are interfering, $r_h = r_k$ or $u_h \neq v_k \vee v_h = u_k$, and there is no other move $r_\hbar: u_\hbar \rightarrow v_\hbar$ in $\mathcal{S}_R^{\prec}(\Sigma)$ such that $h < \hbar < k$ interfering with $r_h: u_h \rightarrow v_h$ or $r_k: u_k \rightarrow v_k$. The notation is that $r_h: u_h \rightarrow v_h \prec r_k: u_k \rightarrow v_k$. □

The definition captures the fact that trivially dependent moves cannot be executed at the same time step. Notice, that the condition $r_h = r_k$ or $u_h \neq v_k \vee v_h = u_k$ is the negation of the condition $r_h \neq r_k$ and $u_h = v_k \wedge v_h \neq u_k$ from the definition of the potential concurrence. Observe that, when $|\{u_h, v_h\} \cap \{u_k, v_k\}| \geq 1$ (interfering moves), the condition $u_h \neq v_k \vee v_h = u_k$ can be equivalently expressed as a disjunction of several cases as follows: $(u_h \neq u_k \wedge v_h = v_k)$ or $(u_h = u_k \wedge v_h \neq v_k)$ or $(u_h = u_k \wedge v_h = v_k)$ or $(u_h = v_k \wedge v_h = u_k)$ or $(u_h \neq v_k \wedge v_h = u_k)$ (original and target vertices of each move are different; thus, each of the conjunctions defines the situation unambiguously with respect to involved vertices). Observe, that none of the cases is actually possible if $r_h = r_k$ and with no middle move $r_\hbar: u_\hbar \rightarrow v_\hbar$ allowed. The relation of trivial dependence of moves is reflexive and anti-symmetric due to the ordering of moves within the sequential solution ($h < k$).

The notions of potential concurrence and trivial dependence are to be used as building blocks of a process that constructs parallel solution of the instance of the problem of multi-robot path planning.

Proposition 6 ***(execution order).*** Let each move of a sequential solution $\mathcal{S}_R^{\prec}(\Sigma)$ is assigned a time step of its execution by a function $t: \bigcup \mathcal{S}_R^{\prec}(\Sigma) \longrightarrow \{1,2,\dots,\zeta - 1\}$. Let t satisfies the following constraint: if $r_h: u_h \rightarrow v_h \prec r_k: u_k \rightarrow v_k$ then $t(r_h: u_h \rightarrow v_h) < t(r_k: u_k \rightarrow v_k)$ and if

$r_h: u_h \rightarrow v_h \preccurlyeq r_k: u_k \rightarrow v_k$ then $t(r_h: u_h \rightarrow v_h) \leq t(r_k: u_k \rightarrow v_k)$. Then a standard (parallel) solution $\mathcal{S}_R(\Sigma)$ constructed from $\mathcal{S}_R^{\prec}(\Sigma)$ using the function t forms a (correct) solution of Σ (sequence of arrangements of robots in $\mathcal{S}_R(\Sigma)$ reflects changes induced by moves at time steps determined by the function t). ■

Proof. The proof will proceed by induction according to the length of the sequential solution $\mathcal{S}_R^{\prec}(\Sigma)$. If the sequence $\mathcal{S}_R^{\prec}(\Sigma)$ consists of a single element, the proposition holds. Suppose that $\mathcal{S}_R^{\prec}(\Sigma) = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \ldots, r_\zeta: u_\zeta \rightarrow v_\zeta]$ is of non-trivial length. From induction hypothesis, the proposition holds for the sequence of moves $\mathcal{S}_R^{\prec}(\Sigma') = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \ldots, r_{\zeta-1}: u_{\zeta-1} \rightarrow v_{\zeta-1}]$. In other words, there is a function $t': \bigcup \mathcal{S}_R^{\prec}(\Sigma') \longrightarrow \{1,2,\ldots,\zeta-1\}$ such that it determines a correct parallel solution of an instance Σ' which is almost the same as Σ except the goal arrangement which differs by the last move $r_\zeta: u_\zeta \rightarrow v_\zeta$.

If there is some move $r_i: u_i \rightarrow v_i$ with $1 \leq i \leq \zeta$ such that $r_\zeta: u_\zeta \rightarrow v_\zeta$ is trivially dependent on it, then t should satisfy that $t(r_i: u_i \rightarrow v_i) < t(r_\zeta: u_\zeta \rightarrow v_\zeta)$. Properties of trivial dependency ensure that execution of $r_\zeta: u_\zeta \rightarrow v_\zeta$ after $r_i: u_i \rightarrow v_i$ does not violate correctness of the solution.

If there is some move $r_j: u_j \rightarrow v_j$ with $1 \leq j \leq \zeta$ such that $r_\zeta: u_\zeta \rightarrow v_\zeta$ is potentially concurrent with it, then t should satisfy that $t(r_j: u_j \rightarrow v_j) \leq t(r_\zeta: u_\zeta \rightarrow v_\zeta)$. The relation of potential concurrence ensures that execution of $r_\zeta: u_\zeta \rightarrow v_\zeta$ at the same time step or after the time step with $r_i: u_i \rightarrow v_i$ does not violate correctness of the solution.

Let $t(r_h: u_h \rightarrow v_h) = t'(r_h: u_h \rightarrow v_h)$ for $h = 1,2,\ldots,\zeta-1$. The function will be defined for the last element of $\mathcal{S}_R^{\prec}(\Sigma)$ specially to satisfy above inequalities with respect to all the trivially dependent and potentially concurrent moves with respect to $r_\zeta: u_\zeta \rightarrow v_\zeta$. Let $t'_{\prec} = \max\{t'(r_i: u_i \rightarrow v_i) \mid r_i: u_i \rightarrow v_i \prec r_\zeta: u_\zeta \rightarrow v_\zeta \wedge i \in \{1,2,\ldots,\zeta-1\}\}$ be the time step assigned to the last trivially dependent move. Similarly, let $t'_{\preccurlyeq} = \max\{t'(r_j: u_j \rightarrow v_j) \mid r_j: u_j \rightarrow v_\mathrm{j} \preccurlyeq r_\zeta: u_\zeta \rightarrow v_\zeta \wedge j \in \{1,2,\ldots,\zeta-1\}\}$ be the time step assigned to the last potentially concurrent move. Let $t(r_\zeta: u_\zeta \rightarrow v_\zeta) \geq \max\{t'_{\prec}+1, t'_{\preccurlyeq}\}$. The function t defined as above satisfies the proposition. ■

Algorithm 2. *The* ***parallelism increasing*** *algorithm.* The algorithm produces a parallelized solution of an instance of multi-robot path planning problem from the given sequential solution. The idea of the algorithm is inspired by the **critical path method** (Russel & Norvig, 2003).

```
function Increase-Parallelism($\mathcal{S}_R^{\prec}(\Sigma), \mathcal{S}_R^0$) : pair
    /* A function for producing standard solution of
    multi-robot path planning problem instance from the
    sequential one.
    Parameters:   $\mathcal{S}_R^{\prec}(\Sigma)$ - a sequential solution of $\Sigma$,
                  $\mathcal{S}_R^0$ - a initial arrangement of robots. */
1: let $\mathcal{S}_R^{\prec}(\Sigma) = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \ldots, r_\zeta: u_\zeta \rightarrow v_\zeta]$
2: $step[1] \leftarrow \{r_1: u_1 \rightarrow v_1\}$
3: $\omega \leftarrow 1$
4: for $k = 2,3,\ldots,\zeta-1$ do
5:  |  $t'_{\prec} \leftarrow$ Earliest-Execution-Time$^{\prec}(r_k: u_k \rightarrow v_k)$
6:  |  $t'_{\preccurlyeq} \leftarrow$ Earliest-Execution-Time$^{\preccurlyeq}(r_k: u_k \rightarrow v_k)$
7:  |  $t \leftarrow \max\{t'_{\prec}+1, t'_{\preccurlyeq}\}$
8:  |  $step[t] \leftarrow step[t] \cup \{r_1: u_1 \rightarrow v_1\}$
```

```
9:  |    $\omega \leftarrow \max\{\omega, t\}$
10: $S_R \leftarrow S_R^0$
11: $\zeta \leftarrow 1$
12: **for** $i = 1,2,\dots,\omega$ **do**
13: |    **for each** $(r: u \rightarrow v) \in$ *step*$[i]$ **do**
14: |    |    $S_R(r) \leftarrow v$
15: |    |    $S_R^i \leftarrow S_R$
16: **return** $(\omega, [S_R^0, S_R^1, \dots, S_R^\omega])$

**function** *Earliest-Execution-Time* $^{\prec}(r_k: u_k \rightarrow v_k, \omega)$ : **integer**
    /* Calculates earliest execution time for a given move
    with respect to the relation of trivial dependency.
    Parameters:    $r_k: u_k \rightarrow v_k$ - a move for that
                   a time step is calculated,
                   $\omega$ – currently last time step. */
1:  **for** $i = \omega, \omega - 1, \dots, 1$ **do**
2:  |    **for each** $(r: u \rightarrow v) \in$ *step*$[i]$ **do**
3:  |    |    **if** $r: u \rightarrow v \prec r_k: u_k \rightarrow v_k$ **then**
4:  |    |    |    **return** $k$
5:  **return** 1

**function** *Earliest-Execution-Time* $^{\preccurlyeq}(r_k: u_k \rightarrow v_k, \omega)$ : **integer**
    /* Calculates earliest execution time for a given move
    with respect to the relation of potential concurrence.
    Parameters:    $r_k: u_k \rightarrow v_k$ - a move for that
                   a time step is calculated,
                   $\omega$ – currently last time step. */
1:  **for** $i = \omega, \omega - 1, \dots, 1$ **do**
2:  |    **for each** $(r: u \rightarrow v) \in$ *step*$[i]$ **do**
3:  |    |    **if** $r: u \rightarrow v \preccurlyeq r_k: u_k \rightarrow v_k$ **then**
4:  |    |    |    **return** $k$
5:  **return** 1
```

The parallelized solution will be constructed according to Proposition 6. To obtain small makespan and high parallelism of the solution, low execution times for execution should be assigned to the individual moves. Thus, it is recommended to assign the time step for the execution of the newly added move in the proposition as follows: $t(r_\zeta: u_\zeta \rightarrow v_\zeta) = \max\{t'_{\prec} + 1, t'_{\preccurlyeq}\}$.

The process is formalized in pseudo-code as Algorithm 2. The method described above is also known as *critical path method* in different contexts (Russel & Norvig, 2003). The algorithm consists of three functions: *Increase-Parallelism*, *Earliest-Execution-Time* $^{\prec}$, and *Earliest-Execution-Time* $^{\preccurlyeq}$. The main framework of the algorithm is represented by the function *Increase-Parallelism*. The function successively includes moves into the constructed parallel solution while trivial dependency and potential concurrence with respect to already included moves is calculated. The function is build over the array *step* which is indexed by time steps. The cell *step*$[t]$ contains a set of moves that are to be executed at the time step t. Functions *Earliest-Execution-Time* $^{\prec}$ and *Earliest-Execution-Time* $^{\preccurlyeq}$ calculates earliest execution time for the newly included move with respect to already included trivially dependent moves and potentially concurrent moves.

Proposition 7 ***(increasing parallelism).*** The algorithm for increasing parallelism has the worst case time complexity of $\mathcal{O}(|\mathcal{S}_R^{\prec}(\Sigma)|^2)$ for the input sequential solution $\mathcal{S}_R^{\prec}(\Sigma) = [r_1: u_1 \rightarrow v_1, r_2: u_2 \rightarrow v_2, \dots, r_\zeta: u_\zeta \rightarrow v_\zeta]$. The worst case space complexity of the algorithm is $\mathcal{O}(|\mathcal{S}_R^{\prec}(\Sigma)|)$. ■

Proof. Each call of *Earliest-Execution-Time*$^{\prec}$ and *Earliest-Execution-Time*$^{\preccurlyeq}$ requires time of $\mathcal{O}(|\mathcal{S}_R^{\prec}(\Sigma)|)$. Both functions are called $|\mathcal{S}_R^{\prec}(\Sigma)|$ times, thus the overall worst case time complexity is $\mathcal{O}(|\mathcal{S}_R^{\prec}(\Sigma)|^2)$. A space of the size $\mathcal{O}(|\mathcal{S}_R^{\prec}(\Sigma)|)$ is required to store the input sequential solution $\mathcal{S}_R^{\prec}(\Sigma)$. A space of the same size is necessary for storing the array *step*. ■

6. Discussion and Conclusions

The graph theoretical abstraction of the problem of path planning for multiple robots has been introduced in this chapter. The abstraction consists in modeling the environment where robots are moving as an undirected graph with vertices standing for locations and edges representing an unblocked way from one location to another. At most one robot is placed in each vertex and at least one vertex remains unoccupied to allow robots to move.

The solving algorithm called *BIBOX* has been shown. It can be used to solve instances of the problem over bi-connected graphs with at least two unoccupied vertices. The algorithm produces a solution of the cubic makespan with respect to the number of vertices of the input graph. The technique how to increase the parallelism and consequently the makespan of solutions has been also presented. A more sophisticated algorithm called *BIBOX-θ* has been developed in (Surynek, 2010c). It is again designed for solving multi-robot path planning over bi-connected graphs. Contrary to the *BIBOX* algorithm, it suffices with just one unoccupied vertex.

There are still some open questions for the future work. The method represented by the *BIBOX* algorithm is suitable for instances with relatively small number of unoccupied vertices where there is high probability of collisions. On the other hand there exists methods suitable for instances with lot of unoccupied space (Wang & Botea, 2008; Wang, 2009). These methods are based on search for shortest paths between initial and goal positions of individual robots. Eventual collisions between robots are resolved by search for alternative paths. The interesting question is under what circumstances one or the other of these two approaches is more advantageous.

Another interesting question regarding the complexity of the optimization variant of the multi-robot path planning problem is whether it is possible to construct a solution with the makespan constant time worse than the optimum in pseudo-polynomial time (= polynomial time with respect to the size of the input and the given constant).

7. References

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001). *Introduction to Algorithms, Second Edition.* The MIT Press, ISBN 978-0-262-03293-3.

Garey, M. R.; Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP Completeness.* W. H. Freeman & Co., ISBN: 978-0716710455.

Kornhauser, D.; Miller, G. L.; Spirakis, P. G. (1984). *Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications.* Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), pp. 241-250, West Palm Beach, FL, USA, IEEE Press, 1984.

Ratner, D.; Warmuth, M. K. (1986). *Finding a Shortest Solution for the N×N Extension of the 15-PUZZLE Is Intractable.* Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), pp. 168-172, Philadelphia, PA, USA, Morgan Kaufmann Publisher.

Russell, S.; Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (second edition).* Prentice Hall, ISBN: 978-0137903955

Ryan, M. R. K. (2008). *Exploiting Subgraph Structure in Multi-Robot Path Planning.* Journal of Artificial Intelligence Research (JAIR), Volume 31, pp. 497-542, AAAI Press.

Surynek, P. (2009a). *A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs.* Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), pp. 3613-3619, ISBN 978-1-4244-2789-5, Kobe, Japan, IEEE Press.

Surynek, P. (2009b). *An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning.* Proceedings of the 21st International Conference on Tools with Artificial Intelligence (ICTAI 2009), pp. 151-158, ISBN 978-0-7695-3920-1, Newark, NJ, USA , IEEE Press.

Surynek, P. (2009c). *Towards Shorter Solutions for Problems of Path Planning for Multiple Robots in Theta-like Environments.* Proceedings of the 22nd International FLAIRS Conference (FLAIRS 2009), pp. 207-212, ISBN 978-1-57735-419-2, Sanibel Island, FL, USA, AAAI Press.

Surynek, P. (2010a). *An Optimization Variant of Multi-Robot Path Planning is Intractable.* Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010), pp. 1261-1263, ISBN 978-1-57735-463-5, Atlanta, GA, USA, AAAI Press.

Surynek, P. (2010b*). Abstract Path Planning for Multiple Robots: A Theoretical Study.* Technical Report, http://ktiml.mff.cuni.cz/~surynek/index.html.php?select=publications, Charles University in Prague, Czech Republic.

Surynek, P. (2010c*). Abstract Path Planning for Multiple Robots: An Empirical Study.* Technical Report, http://ktiml.mff.cuni.cz/~surynek/index.html.php?select=publications, Charles University in Prague, Czech Republic.

Tarjan, R. E. (1972). *Depth-First Search and Linear Graph Algorithms.* SIAM Journal on Computing, Volume 1 (2), pp. 146-160, Society for Industrial and Applied Mathematics.

Wang, K. C. (2009). *Bridging the Gap between Centralised and Decentralised Multi-Agent Path-finding.* Proceedings of the 14th Annual AAAI/SIGART Doctoral Consortium (AAAI-DC 2009), pp. 23-24, AAAI Press, 2009.

Wang, K. C.; Botea, A. (2008). *Fast and Memory-Efficient Multi-Agent Path-finding.* Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), pp. 380-387, ISBN 978-1-57735-386-7, Australia, AAAI Press, 2008.

West, D. B. (2000). *Introduction to Graph Theory, second edition.* Prentice-Hall, ISBN 978-0130144003.

Wilson, R. M. (1974). *Graph Puzzles, Homotopy, and the Alternating Group.* Journal of Combinatorial Theory, Ser. B 16, pp. 86-96, Elsevier.