Algorithmica (2010) 58: 610–636
DOI 10.1007/s00453-009-9290-7

Multi-Color Pebble Motion on Graphs

Gilad Goraly · Refael Hassin

Received: 12 February 2008 / Accepted: 3 February 2009 / Published online: 25 February 2009

Abstract We consider a graph with n vertices, and $p < n$ pebbles of m colors. A pebble move consists of transferring a pebble from its current host vertex to an adjacent unoccupied vertex. The problem is to move the pebbles to a given new color arrangement.

We study the feasibility version of the problem—does a given instance have a solution? We use an algorithm of Auletta et al. (Algorithmica 23:223–245, 1999) for the problem where each pebble has a distinct color to give a linear time algorithm for the feasibility decision problem on a general graph.

Keywords Motion planning · Pebble motion

1 Introduction

We consider the following pebble motion problem, which we call the m COLORS PEBBLE MOTION PROBLEM (m-PM). Let G be a graph with n vertices and $p < n$ pebbles of m colors. Each vertex is either colored in one of the colors, or not colored. Each vertex can host at most one pebble. A *pebble move* consists of transferring a pebble from its current host vertex to an adjacent unoccupied vertex. The problem is to move the pebbles to a prespecified new arrangement.

We also consider a more general version. In this version, which we call T-m-PM, some of the vertices are *transshipment vertices* which cannot host pebbles. A pebble can move from a neighbor of a transshipment vertex to another unoccupied neighbor, but without stopping at the transshipment vertex. A transshipment vertex with its

G. Goraly · R. Hassin (✉)
Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv 69978, Israel
e-mail: hassin@math.tau.ac.il

G. Goraly
e-mail: gilad@goraly.com

neighbors are equivalent (in respect to the pebble movement possibilities) to a clique consisting of the neighbors of the transshipment vertex.

We use an algorithm of [3] for the problem where each pebble has a distinct color on a tree, and the T-m-PM to give an algorithm for the feasibility decision problem on a general graph.

Two major variants of pebble motion problems have been considered in the literature. In the PERMUTATION PEBBLE MOTION (PPM) every pebble is distinct ($m = p$). In the other variant, there are two types of pebbles. Pebbles of the first type, often called *robots*, are required to move to given terminal vertices, whereas the pebbles of the other type do not have terminals and serve as *obstacles* for the first set. The latter problem is sometimes called the ROBOT MOTION PROBLEM.

Two questions are asked regarding these problems: The *feasibility question*—is there a feasible solution for a specific instance? The *optimality question*—given that a feasible solution exists, find one with a minimum number of moves.

There are applications of pebble motion on graphs in several areas. Most of the research considers problems where every robot is unique [3, 5, 11, 15], and some also study the case where all robots are identical [5, 6]. We generalize these models and extend the study assuming the objects to be identical within classes, and each class is distinct.

In the next section we review some related literature. Section 3 contains definitions and notation. In Sect. 4 we describe known results on which we base our study, and in Sect. 5 we highlight our new results.

We show that the feasibility of m-PM on a tree can be decided in linear time. In Sect. 6 we apply the analysis of PPM on a tree made by Auletta et al. [3], to a tree with transshipment vertices. In Sect. 7 we apply this result to T-m-PM on a tree. In Sect. 8 we address m-PM on vertex-connected graphs. Our approach to solve the feasibility question, is to transform the problem to a T-m-PM problem on a tree.

2 Literature on Pebble Motion

Some puzzles can be modeled as pebble motion problems. A nice example is KNIGHTS EXCHANGE [16]. Here, several knights of two colors are located on a fragment of a chessboard. The goal is to interchange their positions using legal knight moves. Modeling the game on a graph, makes the puzzle easier. See Fig. 1 where the graph is a tree and the feasibility question can be decided according to the analysis is Sect. 7. Here the initial location of one class of pebbles is the set of terminals of the other class.

Christofides and Colloff [4] derive an algorithm for a version of PPM. The items are rearranged by a 'vehicle', and the cost of moving an item between two locations may be different from the cost of moving the unloaded vehicle between the same pair of points. There is also a temporary auxiliary storage space of limited capacity, and the goal is to perform the task at minimum cost.

PPM generalizes Sam Loyd's famous "15-Puzzle" [13]—a PPM on a 4×4 lattice with 15 pebbles. Archer [1] shows that the feasible instances of the 15-puzzle are those with arrangement which is an even permutation of the standard arrangement. He

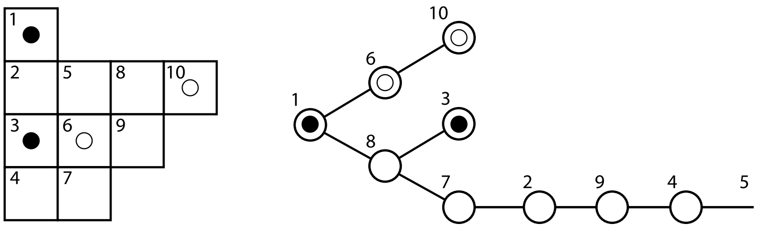

Fig. 1 Knight Exchange game modeled as a 2-PM on a tree. It is required to exchange the black pebbles with the white pebbles

uses nine simple permutations along a Hamiltonian path to generate the alternating group A_{15}. This method is applicable to any graph containing a Hamiltonian path. The only full description of an $n \times n$-puzzle is for $n = 3$. Reinefeld [18] shows a complete description of the 8-puzzle.

Wilson [19] generalizes the 15-puzzle and proves a feasibility criterion for PPM on 2-vertex-connected graphs where the number of pebbles is $p = n - 1$. Kornhauser et al. [11] follow Wilson's work and show that feasibility for PPM on general graphs can be decided in polynomial time. If the instance is feasible then their algorithm produces a feasible sequence of moves. They prove that the number of moves is $O(n^3)$ with a matching lower bound. They decompose the instance to subgraphs where each subgraph is 1-transitive, i.e., each pebble can move to any part of the subgraph. The instance on the original graph is feasible if and only if the instance induced on every subgraph, is feasible. They check the feasibility of each 1-transitive subgraph by pruning vertices and pebbles that have no effect on the feasibility. This is done by identifying intermediate vertices, that can be removed with the pebbles they host and thus reducing the size of the problem. This solution can be performed in $O(n^3)$ time.

Auletta, Monti, Parente, and Persiano [3] present a linear time algorithm for PPM feasibility on trees. They reduce the problem to a problem of swapping two pebbles at a time, then they divide the tree into equivalence classes—in each class all the pebbles can be swapped. We give more details on their work in Sect. 4.

Papadimitriou, Raghavan, Sudan, and Tamaki [15] prove a simple feasibility criterion for the ROBOTS MOTION PROBLEM. On a biconnected graph, it is sufficient to have two holes in order to move the robot to any vertex on the graph. On a tree, a necessary condition is to have $l + 3$ holes where l is the number of vertices on the longest path with vertices of degree 2, the robot needs to pass. A necessary and sufficient condition is that the robot will be able to reach the nearest vertex with degree greater than 2.

Călinescu, Dumitrescu, and Pach [5] consider an optimality versions of PPM and 1-PM in which moving a pebble along an arbitrary long path is considered a single move. They prove hardness of these versions and provide bounds and an approximation algorithm.

Several optimality problems have been studied, and most of them are hard. Goldreich [7] proves that computing the shortest plan for an $n \times n$ lattice PPM with $(n^2 - 1)$ pebbles, a generalization of the "15-puzzle", is NP-hard. This is later extended by

Ratner and Warmuth [17] to grid graphs. Papadimitriou et al. [15] show that finding the shortest plan for the ROBOT MOTION PROBLEM is NP-hard.

There are several techniques to solve the 15-puzzle and its generalizations. Parberry [14] gives an algorithm for the (n^2-1)-puzzle, which uses $5n^3$ moves at the most. The algorithm first orders the first row and the first column in place, then moves to solve the $(n-1)\times(n-1)$ grid. The last 3×3 grid is solved by brute force. Hayes [8] adapts Parberry's scheme and obtains a better result for the worst case analysis.

There are some bounds on the number of steps needed for solution. Kornhauser et al. [11] show an upper and lower bound of $\Theta(n^3)$ on the number of moves required for a solution of PPM on a general graph. Auletta et al. [3] give a bound of $O(p^2(n-p))$ on the number of moves on a tree, where p is the number of pebbles, and a sequence of swaps of two pebbles is used to solve the instance. Papadimitriou et al. [15] give a polynomial time algorithm for the optimal solution of the robot motion on trees. Their algorithm solves $O(n^6)$ min-cost-flow problems. They also give a 7-approximation algorithm that solves $O(n)$ min-cost-flow problems. Auletta et al. [2] improve these results to $O(n^5)$ for solving the robot problem on trees.

In a *graph pebbling* model (Milans and Clark [12]), a $u \to v$ pebble move consists of removing two pebbles from u and adding one pebble to v. The pebbles are of the same color and the vertices have no hosting limit. The *pebbling number* $\pi(G)$ is the minimum k such that for all arrangements of k pebbles and for any target vertex t, there is a sequence of moves which places a pebble on t.

An application that uses a pebbling model is described by Jacobs, Rader, Kuhn and Thorpe [10]. They model the strength of the connection within the protein based on the 3-D Pebble Game [9], where the pebbles represent the degrees of freedom.

3 Preliminaries

Let $G=(V,E)$ be a connected undirected graph with vertex set V, $|V|=n$, and edge set E. For a subgraph H denote its vertex set by $V(H)$, and its edge set by $E(H)$.

Let f be a function acting on a set of vertices. As a convention, we denote $f(v)=f(\{v\})$.

Definition 3.1 There is a set of colored pebbles placed on V so that each $v\in V$ hosts at most one pebble. *Color arrangement*[1] $A=(S_0(A),S_1(A),\dots,S_m(A))$ is a partition of the vertices in V according to the color of the pebble they host, at a certain moment. $S_c(A)$ is the set of vertices which host pebbles of color c in the color arrangement A. In particular, $c=0$ is not a valid color for a pebble and $S_0(A)$ is the set of unoccupied vertices or *holes*. Clearly, $\bigcup_{c=0}^{m} S_c(A)=V$. We omit the '$A$' in $S_c(A)$ when it is clear from the context. $A^{(0)}$ denotes the *initial color arrangement* of the pebbles. Denote $s_c=|S_c|$ and let $s=(s_0,s_1,\dots,s_m)$ be the vector indicating the number of pebbles of each color. In particular, s_0 is the number of unoccupied vertices and m is the number of different colors.

[1] This is called *configuration* in [3].

Definition 3.2 A *pebble move* is a transfer of a pebble from its current location to an adjacent hole. Let $e = (u, v) \in E$. Given a color arrangement A, a $u \to v$ move is *feasible* if $u \in S_c$ for some $c > 0$, and $v \in S_0$. A *plan*, denoted by f, is a sequence of feasible moves. $T(f, A)$ denotes the color arrangement achieved by executing f on the color arrangement A.

Definition 3.3 In an instance, I, of the m-COLORS PEBBLE MOTION PROBLEM (m-PM) we are given a graph $G(I)$, an initial color arrangement $A^{(0)}(I)$ of pebbles of m colors, and a final required color arrangement $A^{(*)} = (S_0(A^{(*)}), S_1(A^{(*)}), \dots, S_m(A^{(*)}))$. It requires to compute a plan f, for which $T(f, A^{(0)}) = A^{(*)}$. I is denoted by $I = (G(I), A^{(0)}(I), A^{(*)}(I))$. When I is clear from the context we omit it and write $I = (G, A^{(0)}, A^{(*)})$.

We also present the problem on a graph with *transshipment vertices*—vertices that cannot host pebbles. We define PPM and m-PM on this kind of graph in Sect. 6.

Let f, g be two plans. Denote by gf the plan created by executing g after f, $T(gf, A) = T(g, T(f, A))$. A plan f has a *reverse plan*, f^{-1}, in which the sequence of moves is executed backwards, $T(f^{-1}f, A) = T(\mathcal{I}, A) = A$, where $\mathcal{I}$ is the *identity plan*. We denote by f^k a plan f executed k times, $f^{-k} = (f^{-1})^k$, $f^0 = \mathcal{I}$.

Without loss of generality, we consider simple graphs: a pebble move $u \to v$ requires an edge $e = (u, v)$ and a hole on v, parallel edges and loops do not affect the feasibility. A $u \to v$ pebble move can be seen as a $v \to u$ *hole move*. In a connected graph the holes can be moved to any set of vertices.

Let $I = (G, A^{(0)}, A^{(*)})$, where $A^{(0)}$ is an initial color arrangement with pebbles on vertices in T_0. We can compose an equivalent problem $I' = (G, A'^{(0)}, A^{(*)})$ where $A'^{(0)}$ is an arrangement for which all the holes are only on vertices of T_0. Since the holes can be moved freely, we can define g to be a plan for which $T(g, A^{(0)}) = A'^{(0)}$. We claim that I is feasible if an only if I' is feasible. If I is feasible, there is a plan f that solves it, hence fg^{-1} solves I'. If I' is feasible, there is a plan, f', that solves it, hence $f'g$ solves I. Therefore we consider only plans for which the initial color arrangement $A^{(0)}$, and final color arrangement $T(f, A^{(0)})$ have the same set of holes, as stated in Observation 3.4:

Observation 3.4 *We can assume without loss of generality that* $S_0(A^{(0)}) = T_0$.

Let $A = (S_0, S_1, \dots, S_m)$ be a color arrangement. Let $v \in S_i$ and $u \in S_j$, $i \neq j$, $0 < i, j$. Suppose there is a plan f such that $T(f, A) = (S_0, S_1, \dots, (S_i \setminus \{v\}) \cup \{u\}, \dots, (S_j \setminus \{u\}) \cup \{v\}, \dots, S_m)$. Then u and v are *equivalent* with respect to A, f is called a *color swap*, and we say that f swaps the colors on u and v.

The equivalence property is transitive. Let f, g be two plans, if u is equivalent to v by f, and v is equivalent to w by g, then u is equivalent to w by fgf or gfg.

The equivalence property is symmetric since every plan f has a reverse plan f^{-1}, and it is also reflexive by an identity plan (e.g. an empty plan). Since the equivalence property is reflexive, symmetric and transitive, it induces a decomposition of V into *equivalence classes*.

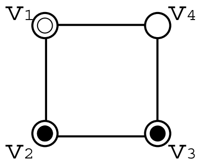

Fig. 2 A cyclic pebble swap on (v_1, v_2, v_3) is a (v_1, v_2) color swap

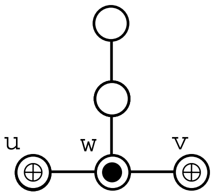

Fig. 3 The graph induced by an equivalence class may be disconnected—u and v are in the same equivalence class, different from the one w belongs to

Definition 3.5 A *cyclic pebble swap* on $\{u_1, \ldots, u_k\}$ is a plan f that swaps the pebbles located on a set of distinct vertices $\{u_1, \ldots, u_k\} \subseteq V \backslash S_0$, in a cyclic order: the pebble located on u_i moves to u_{i+1}, $i = 1, \ldots, k-1$, and the pebble on u_k moves to u_1. A cyclic pebble swap does not change the other pebble locations on the graph. When $k = 2$ we simply refer to it as a *pebble swap*.

A simple example emphasizes the difference between 2-PM and PPM:

Example 3.6 Let $G = (V, E)$ be a cycle with four vertices $V = \{v_1, v_2, v_3, v_4\}$, assume $S_0 = \{v_4\}$, $S_1 = \{v_1\}$, and $S_2 = \{v_2, v_3\}$, as in Fig. 2. There is no (v_1, v_3) pebble swap, but there is a (v_1, v_3) color swap: execution of a cyclic pebble swap on (v_1, v_3, v_2) result in $S_0 = \{v_4\}$, $S_1 = \{v_3\}$, and $S_2 = \{v_2, v_1\}$

4 Known Results

Auletta et al. [3] study PPM on a tree. They show that feasibility can be decided in linear time by partitioning $V(G)$ into equivalence classes. Any pair of pebbles located in the same equivalence class can be swapped. Pebbles from different equivalence classes cannot be swapped. Hence, the partition to equivalence classes is sufficient for deciding the feasibility of the instance I. They show that this condition is necessary—if there is a solution, there is a solution using swaps. An interesting observation is that the graph induced by an equivalence class may be disconnected. See Fig. 3 for example: the pebbles on u and on v can be swapped, but the pebble on w cannot be swapped with neither the pebble on u nor the pebble on v. Thus u and v are in the same equivalence class while w is in another equivalence class.

5 New Results

We generalize the result of [3] on a tree by allowing transshipment vertices. Our motivation is to use it for solving m-PM on general graphs.

Next we show that when G is a tree, the results of T-PPM can be used for T-m-PM and show that the feasibility of T-m-PM on a tree is decidable in linear time.

When we consider general graphs, we reduce the problem to T-m-PM and use the former results to show that the feasibility of m-PM on a connected general graph is decidable in linear time. This result is valid also when $m = p$ and thus give a linear time decision algorithm for PPM on a general graph. This problem was described as an open problem in [3].

6 T-PPM on a Tree

We modify the algorithm given in [3] to work on a tree G with transshipment vertices. The work of [3] is brought almost as is, we follow their theorem and lemmas, and modify the proofs of some of the lemmas.

6.1 Notation and Definitions

Definition 6.1 A *transshipment vertex* is a vertex that cannot host a pebble, other vertices are called *regular vertices*. Define $V(G) = V_T(G) \cup V_R(G)$, where $V_T(G)$ is the set of transshipment vertices and $V_R(G)$ is the set of regular vertices. We omit G when it is clear from the context.

A color arrangement includes only vertices of V_R, $w \in V_T \Rightarrow w \notin S_c\ \forall c, 0 \leq c \leq m$. In particular $w \in V_T$ is not a hole even though it does not host a pebble.

Definition 6.2 For each $u, v \in V$ denote by P_{uv} the unique path between u and v. P_{uv} includes all edges and vertices on the path, excluding u and v, and we define $P_{uv} = \{(u, v)\}$ when $(u, v) \in E(G)$.

We say that the path between v and u is *clear* when it is either empty and/or contain transshipment vertices, i.e. and $V(P_{uv}) = \emptyset$ or $V(P_{uv}) \subseteq V_T$. Let $u, v \in V_R$, a $u \to v$ move is feasible if u hosts a pebble ($u \notin S_0$), v is unoccupied ($v \in S_0$), and P_{uv} is clear. $v \in V_R$ appears in the figures as a circle and $w \in V_T$ appears in the figures as a square. See Fig. 4 for an example.

Definition 6.3 Define T-PPM to be a PPM on a graph with transshipment vertices. Define T-m-PPM to be a m-PPM on a graph with transshipment vertices. In both cases, V_R takes the role of V in the original problem.

Assumption 6.4 Without loss of generality the following hold in T-PPM:

- There are no two adjacent transshipment vertices.
- Any transshipment vertex has a degree greater than 2.

There is no loss of generality in the first item of Assumption 6.4 since two adjacent transshipment vertices can be merged into a single transshipment vertex. This merge will not affect the pebble motion since the movement between two transshipment

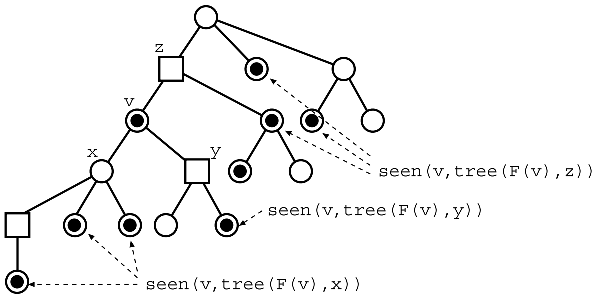

Fig. 4 seen(v)

vertices is always feasible. It will also not affect the number of holes in the graph since the transshipment vertices are not holes. Finally, since the graph is a tree it will not affect the movement possibilities between other pebbles, and the resulting graph is still a tree.

There is no loss of generality in the second item of Assumption 6.4 since a transshipment vertex with a degree 2 or less can be removed from the graph (with the necessary changes to keep the graph connected), without affecting the movement of pebbles. Removing it does not change the number of holes and since the degree is 2 or less, it does not change the feasible moves between its adjacent vertices.

Definition 6.5 For each $v \in V$ denote by $deg(v)$ the number of edges connected to v.

Definition 6.6 We root the tree G at a vertex that hosts a pebble, call this vertex *root*. For each vertex u we denote by $F(u)$ the forest obtained from G by removing u and the edges connected to it. For a vertex v in a forest F denote by $\mathrm{tree}(F, v)$ the tree of F containing v, and by $R(F, v)$ the remaining forest $F \backslash \mathrm{tree}(F, v)$.

Definition 6.7 Let $u \in V_R$ be a regular vertex adjacent to $v \in V$. Define $\mathrm{dist}(u, v) = 0.5$ if $v \in V_T$ and $\mathrm{dist}(u, v) = 1$ if $v \in V_R$. For $a, b \in V$ define the distance $\mathrm{dist}(a, b)$ as the sum of $\mathrm{dist}(u, v)$ over the edges (u, v) of the induced path P_{ab} in G. We call a vertex with degree greater than 2 a *branch*.

Definition 6.8 For a given $v \in V_R \setminus S_0$ and tree $Q \in F(v)$. Denote by $\mathrm{holes}(Q) = |S_0 \cap V(Q)|$ the number of holes in Q. Q is *full* if $\mathrm{holes}(Q) = 0$. Define $\mathrm{seen}(v, Q)$ as the set of occupied vertices in $V_R(Q)$ for which there is a clear path from v

$$\mathrm{seen}(v, Q) = \{u \in V_R(Q) \setminus S_0 \mid Q \in F(v), V(P_{vu}) \subseteq S_0\}.$$

Define $\mathrm{seen}(v)$ as the set of occupied vertices in $V_R(G)$ for which there is a clear path from v

$$\mathrm{seen}(v) = \bigcup_{Q \in F(v)} \mathrm{seen}(v, Q),$$

in particular $v \notin \mathrm{seen}(v)$. See Fig. 4 for an example.

The values of $\mathrm{seen}(v, Q)$, $\mathrm{holes}(Q)$, for each vertex $v \in V_R \setminus S_0$ and for each subtree $Q \in F(v)$ are computed in $O(n)$ time during a preprocessing phase (see more details in the proof of Theorem 6.16).

6.2 The Algorithm

In this section we present the algorithm MARK (see Fig. 12) which on an instance $I = (G, A^{(0)}, A^{(*)})$ of T-PPM, partitions the vertices of $V_R \setminus S_0$ into equivalence classes (recall the definition in Sect. 3).

Lemma 6.9 *Consider $u, v \in V_R$, $a, b \in V_R(P_{uv}) \cup \{u, v\}$. Let A_{ab} be an arrangement where*

$$V_R(P_{uv}) \cup \{u, v\} \setminus \{a, b\} \subseteq S_0, \quad a, b \notin S_0.$$

Let A_{uv} be the arrangement obtained from A_{ab} by moving the pebbles from a and b to u and v. Then, for any selection of a and b, u and v are equivalent with respect to A_{uv} if and only if a and b are equivalent with respect to A_{ab}.

Proof Assume a is closer to u. Assume a and b are equivalent, it means that the pebbles they host can be swapped. Since the pebbles can move freely on P_{uv}, they can be moved to any other pair of vertices, which means that any pair of vertices on $V_R(P_{uv}) \cup \{u, v\}$ is equivalent with respect to the new arrangement. Since a and b can be also u and v, this proves both sides of the claim. □

The following lemma identifies the conditions sufficient for two vertices to be not equivalent. A similar lemma was proved in [3] for PPM, we show the same logic holds also for T-PPM.

Lemma 6.10 *Consider $u, v \in V_R \setminus S_0$, and suppose $(u, v) \in E(G)$ or that every $w \in P_{uv}$ has degree 2 and $V_R(P_{uv}) \subseteq S_0$. Let $F_1 = R(F(u), v)$ and $F_2 = R(F(v), u)$. Suppose that for all branch vertices $w_1 \in F_1$*

$$\mathrm{holes}(F_1) \leq \lfloor \mathrm{dist}(u, w_1) \rfloor + 1,$$

and that for all branch vertices $w_2 \in F_2$

$$\mathrm{holes}(F_2) \leq \lfloor \mathrm{dist}(v, w_2) \rfloor + 1.$$

Then u and v are not equivalent.

Proof Call the original instance I, see Fig. 5 as an example. Create a new instance I', by replacing each transshipment vertex with a hole. We claim that the conditions of the lemma hold for vertices u and v in I if and only if they hold for them also in I'. Suppose there exists a transshipment branch vertex $w_1 \in F_1$ and $\mathrm{holes}(F_1) = x \leq \lfloor \mathrm{dist}(u, w_1) \rfloor + 1 = y + 1$. Replace w_1 with a hole, and call it w_1'. Now $\mathrm{holes}(F_1) = x + 1$ and in the new graph $\mathrm{dist}(u, w_1') + 1 = (y + 1) + 1$. Since $\lfloor \mathrm{dist}(u, w_1') \rfloor =$

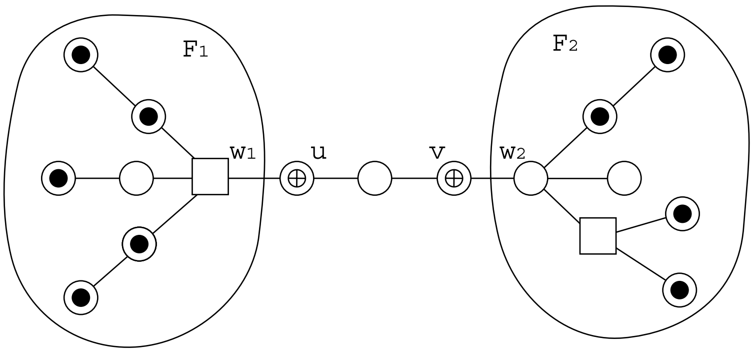

Fig. 5 u and v are not equivalent

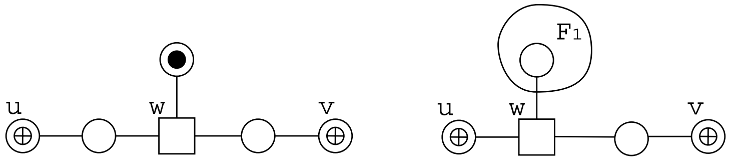

Fig. 6 Lemma 6.11, condition 1. $F_1 = R(F(w), u) \cap R(F(w), v)$

$\mathrm{dist}(u, w'_1)$, $\mathrm{holes}(F_1) \leq \mathrm{dist}(u, w'_1) + 1$ if and only if $\mathrm{holes}(F_1) \leq \lfloor \mathrm{dist}(u, w_1) \rfloor + 1$. This argument holds also for $w_2 \in F_2$.

By Corollary 3 in [3] $\mathrm{holes}(F_i) \leq \mathrm{dist}(u, w'_i) + 1$, $i = 1, 2$, implies that u and v are not equivalent in I'. Since a hole can be used as a transshipment vertex and more, it is clear that if there is no possible (u, v) swap in I', then there cannot be a (u, v) swap in I. Thus u and v are not equivalent also in I. □

The following lemma gives a necessary and sufficient condition for the equivalence of a vertex $v \in V_R \setminus S_0$ and a vertex $u \in \mathrm{seen}(v)$.

Lemma 6.11 *Consider $u, v \in V_R \setminus S_0$, and suppose $(u, v) \in E(G)$ or that $V_R(P_{uv}) \subseteq S_0$. Then u and v are equivalent if an only if at least one of the following conditions is met*:

1. *There exists a branch $w \in P_{uv}$ such that*:

$$\min\{\mathrm{holes}(\mathrm{tree}(F(w), u)), \mathrm{holes}(\mathrm{tree}(F(w), v))\} > 0$$

or

$$\mathrm{holes}(R(F(w), u) \cap R(F(w), v)) > 0$$

(*see Fig.* 6).

2. *There exists a branch vertex w such that $u \in P_{wv}$ and*

$$\mathrm{holes}(\mathrm{tree}(F(u), w)) \geq \lfloor \mathrm{dist}(w, u) \rfloor + 2$$

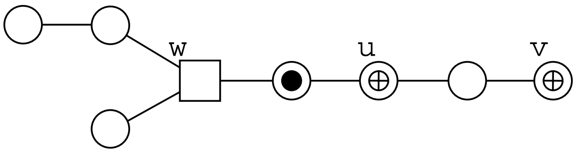

Fig. 7 Lemma 6.11, condition 2. $\mathrm{dist}(w,u) = 1\frac{1}{2}$, $\mathrm{holes}(\mathrm{tree}(F(u),w)) = 3 \geq \lfloor 1\frac{1}{2} \rfloor + 2$

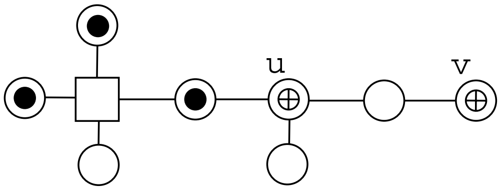

Fig. 8 Lemma 6.11, condition 4

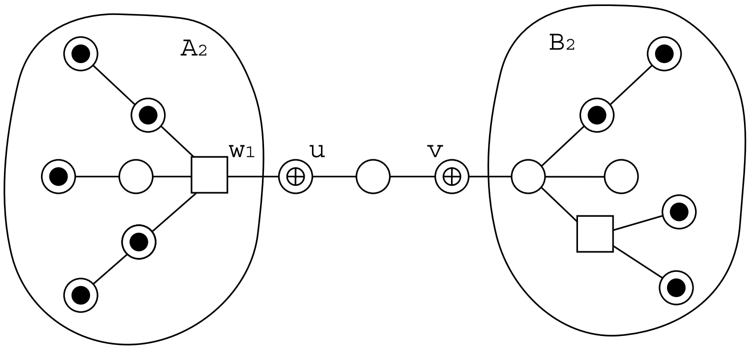

Fig. 9 Example for the proof of Lemma 6.11, case 1 (a)

(*see Fig*. 7).

3. *As in* 2. *with the roles of u and v interchanged.*
4. $deg(u) > 2$, *at least two trees of* $F(u)$ *are not full*, *and at least two holes are outside of* $\mathrm{tree}(F(u), v)$ (*see Fig*. 8).
5. *As in* 4. *with the roles of u and v interchanged.*

Proof The sufficiency can be proved by constructing a (u, v) swap for each of the cases described in these conditions. The necessity is proved by showing that in each possible arrangement, if none of the conditions hold, then u and v are not equivalent.

1. First consider the case where P_{uv} does not contain a branch vertex (thus condition 1 does not hold).
   (a) Suppose that u and v are not branch vertices (thus conditions 4 and 5 do not hold). Now $F(u)$ consists of two trees A_1 and A_2, and similarly, $F(v)$ consists of two trees B_1 and B_2. Assume $v \in A_1$ and $u \in B_1$ (A_2 and B_2 may be empty). See example in Fig. 9. Since conditions 2 and 3 are not met, then for all branch vertices $w_1 \in A_2$ we have

$$\mathrm{holes}(\mathrm{tree}(F(u), w_1)) < \lfloor \mathrm{dist}(w_1, u) \rfloor + 2,$$

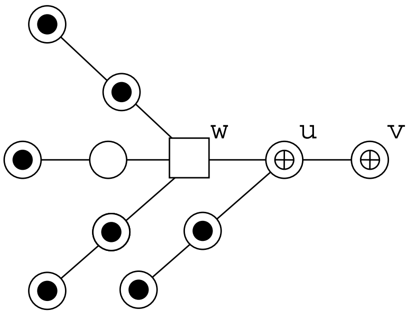

Fig. 10 Example for the proof of Lemma 6.11, case 1 (b) i A

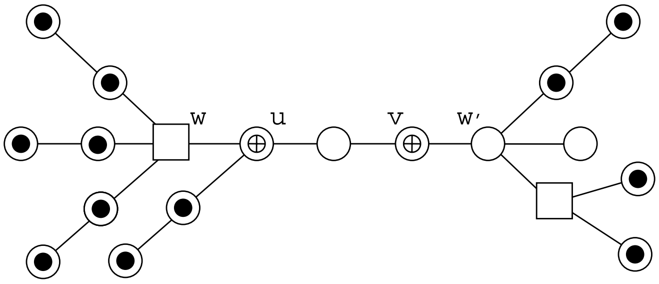

Fig. 11 Example for the proof of Lemma 6.11, case 1 (b) i B

and for all branch vertices $w_2 \in B_2$ we have

$$\text{holes}(\text{tree}(F(v), w_2)) < \lfloor \text{dist}(w_2, v) \rfloor + 2.$$

Therefore by Lemma 6.10 u and v are not equivalent.

(b) Now consider the case in which u is a branch vertex (the case where v is a branch vertex is handled by the same argument). Let w be a branch vertex w closest to u. We have two options to negate condition 4:

i. Only one tree $T' \in F(w)$ has holes.

A. If $v \notin T'$ then u and v are on a tree with no holes so they must be adjacent. See example in Fig. 10. Moreover, since condition 2 does not hold, we get:

$$\text{holes}(T') < \lfloor \text{dist}(w, u) \rfloor + 2.$$

Therefore it is possible to move the pebble from u to a vertex w' adjacent to w so that all the holes from T' are now located on the vertices of $P_{w'v}$. Since only T' has holes holes$(R(F(u), v)) = 0$, then by Lemma 6.10 w' and v are now not equivalent, thus by Lemma 6.9 u and v are not equivalent with respect to the original arrangement.

B. If $v \in T'$ then let w' be the branch vertex of $R(F(v), u)$ closest to v. See example in Fig. 11. Since condition 3 does not hold, we get

$$\text{holes}(\text{tree}(F(v), w')) < \left\lfloor \text{dist}(w', v) \right\rfloor + 2.$$

However, since only T' has holes holes$(R(F(u), v)) = 0$, then u and v meet the conditions of Lemma 6.10, hence they are not equivalent.

ii. There are less than two holes outside $A_1 = \text{tree}(F(u), v)$.

A. If holes$(R(F(u), v)) = 0$, since condition 2 does not hold, we get for the branch vertex $w \in A_1$ closest to v:

$$\text{holes}(A_1) < \lfloor \text{dist}(w, v) \rfloor + 2.$$

Therefore it is possible to move the pebble from v to a vertex w' adjacent to w so that all the holes from A_1 are now located on the vertices of $P_{w'u}$. Since holes$(R(F(u), v)) = 0$, then by Lemma 6.10 w' and u are now not equivalent, thus by Lemma 6.9 u and v are not equivalent with respect to the original arrangement.

B. If holes$(R(F(u), v)) = 1$, since condition 2 does not hold, we get for the branch vertex $w_1 \in A_1$ closest to v:

$$\text{holes}(A_1) < \lfloor \text{dist}(w_1, v) \rfloor + 2,$$

and for the branch vertex $w_2 \in B_1$ closest to u:

$$\text{holes}(B_1) < \lfloor \text{dist}(w_2, u) \rfloor + 2.$$

Therefore it is possible to move the pebble from v to a vertex ${w_1}'$ adjacent to w_1 so that all the holes from A_1 are now located on the vertices of $P_{{w_1}'u}$. Similarly, it is possible to move the pebble from u to a vertex ${w_2}'$ adjacent to w_2 so that all the holes from B_1 are now located on the vertices of $P_{{w_2}'v}$. Then by Lemma 6.10 ${w_1}'$ and ${w_2}'$ are now not equivalent, thus by Lemma 6.9 u and v are not equivalent with respect to the original arrangement.

2. Now suppose there are some branch vertices in P_{uv}. Since condition 1 in not met, we know that holes$(R(F(w), v) \cup R(F(w), u)) = 0$.
   (a) If there are more than two branch vertices in P_{uv}, there is at least one, w, not adjacent to v nor to u. Since two vertices of V_T cannot be adjacent and since P_{uv} has no pebbles, it turns out that two of the trees of $F(w)$ have holes. Thus condition 1 is met, contradicting our hypothesis.
   (b) If there are exactly two branch vertices in P_{uv}, w_1 and w_2, then since condition 1 is not met, they have to be adjacent to u and to v. By Lemma 6.10 w_1 and w_2 are now not equivalent (since there are no holes in the trees containing u and v). Therefore, by Lemma 6.9 u and v are not equivalent with respect to the original arrangement.
   (b) If there is one branch vertex, w, w must be adjacent to u or to v, suppose it is adjacent to u. Thus all the trees of $F(w)$ are full, except for tree$(F(w), v)$. Since condition 2 does not hold and since all the trees of $F(w)$ are full (except for tree$(F(w), v)$), w and v satisfy the conditions of Lemma 6.10, and they are now not equivalent. Therefore, by Lemma 6.9 u and v are not equivalent with respect to the original arrangement. □

Call $T_0(v) = \text{tree}(F(v), root)$, and $T_1(v), T_2(v), \ldots, T_h(v)$ the trees of $F(v)$ rooted at the children of v. Call *bad vertices* the vertices of $\text{seen}(v)$ which are not equivalent to v. The action of marking the bad vertices for equivalence, in Algorithm MARK is based on Lemmas 6.12–6.15, and proved in [3] for PPM (Lemmas 9–12). We generalize the results to the case of T-PPM. The proofs of Lemmas 6.12–6.15 are identical to the proofs of Lemmas 9–12 in [3] and rely on Lemmas 6.10 and 6.11 (which are analogues of Corollary 3 and Lemma 8 in [3]).

Lemma 6.12 *For each $j = 0, \ldots, h$ all the bad vertices of $\text{seen}(v, T_j(v))$ are in the same equivalence class.*

Lemma 6.13 *Suppose that $u \in \text{seen}(v, T_j(v))$ is a bad vertex and it is not equivalent to any vertex of $\text{seen}(v) \setminus T_j(v)$. Then all the vertices equivalent to u belong to $T_j(v)$.*

Lemma 6.14 *If some vertex of $\text{seen}(v)$ is bad, then at most two trees of $F(v)$ are not full, and at most one of them contains more than one hole.*

Lemma 6.15 *Let $u \in \text{seen}(v, T_j(v))$ and $w \in \text{seen}(v, T_l(v))$, with $l \neq j$, be two bad vertices. u and w are equivalent if and only if there exists a tree $T_h(v)$, $h \neq l, j$ such that $\text{holes}(T_h(v)) \geq 2$.*

Algorithm MARK, see Fig. 12, inserts the root vertex in an equivalence class by itself and then traverses the tree by levels. Each time a regular vertex $v \notin S_0$ is visited, algorithm MARK assigns each vertex of $\text{seen}(v)$ that is also a descendant of v to an equivalence class.

It marks for equivalence or non-equivalence (with respect to v) each vertex $u \in \text{seen}(v, T_i(v))$, $i = 1, \ldots, h$. In particular all vertices equivalent to v are marked. Lemma 6.11 gives a necessary and sufficient condition for a vertex $u \in \text{seen}(v)$ to be equivalent to v. This condition can be tested in constant time (since the number of holes for each sub-tree is computed in a preprocessing phase as explained in Theorem 6.16).

By Lemma 6.12, all bad vertices of $\text{seen}(v)$ that belong to the same $T_j(v)$ are equivalent, by Lemma 6.13 if at least one of these vertices is not equivalent to vertices of $\text{seen}(v)$ outside of $T_j(v)$ then all of them are not equivalent to vertices outside of $T_j(v)$. Thus all is left to check is whether bad vertices from different subtrees are equivalent to each other. By Lemma 6.14 if there is any bad vertex then exactly one of the following cases holds. Algorithm MARK splits the case where there is only one tree with at least two holes into two cases, and checks the following four cases:

1. Only one tree, $T_0(v)$ (the tree that contains the root), has at least two holes. By Lemma 6.15 all the bad vertices outside of $T_0(v)$ belong to the same equivalence class, while the bad vertices of $T_0(v)$ belong to their own equivalence class. Therefore all the bad vertices outside $T_0(v)$ belong to a new equivalence class. This case is covered in the first item of the algorithm.
2. There is only one $T_j(v)$ with at least two holes. By Lemma 6.15 all the bad vertices outside of $T_j(v)$ belong to the same equivalence class, while the bad vertices of $T_j(v)$ belong to their own equivalence class. If $j \neq 0$ all the bad vertices outside

```
Mark
  input
  1. A tree $G = (V, E)$, $V = V_R \cup V_T$.
  2. $p$ pebbles in an initial color arrangement $A^{(0)}$.
  returns
  $V_R$, partitioned to equivalence classes.
  begin
  Choose $v_0 \in V_R \setminus S_0$ and set it as the root of $G$, label its descendants in BFS order[a].
  Calculate $\mathrm{seen}(v)$ for each $v \in V$.
  Calculate $\mathrm{holes}(T_j(v))$ for each $v \in V$ and for every $T_j(v) \in F(v)$.
  Put $v_0$ in an equivalence class.
  for $i = 0, \dots, n-1$:
      Put the vertices equivalent to $v_i$ according to Lemma 6.11 in the same
      equivalence class as $v_i$.
      Let $W$ be the set of bad vertices of $v_i$ that do not belong to $T_0(v_i)$ ($T_0(v_0) = \emptyset$).
      1. if $\mathrm{holes}(T_0(v_i)) > 0$
          then
              Put the vertices of $W$ in a new equivalence class.
              return
      end if
      Let $h$ be number of children of $v_i$.
      2. if there is a unique $1 \le j \le h$ such that $holes(T_j(v_i)) > 0$
          then
              Put the vertices of $W \backslash T_j(v_i)$ in the equivalence class of the parent of $v_i$.
              Put the vertices of $W \cap T_j(v_i)$ in a new equivalence class.
              return
      end if
      3. if there are exactly two trees $T_j(v_i)$, $T_l(v_i)$, $1 \le j, l \le h$ with holes and
          $\mathrm{holes}(T_j(v_i)) = \mathrm{holes}(T_l(v_i)) = 1$
          then
              Put the vertices of $W \cap T_j(v_i)$ in a new equivalence class.
              Put the vertices of $W \cap T_l(v_i)$ in a new equivalence class.
              return
      end if
      4. if there are exactly two trees $T_j(v_i)$, $T_l(v_i)$, $1 \le j, l \le h$ with holes and
          $\mathrm{holes}(T_j(v_i)) > 1$
          then
              Put the vertices of $W \cap T_j(v_i)$ in a new equivalence class. [If $v_i$ is a
              branch, $W \subseteq T_j(v_i)$.]
              return
      end if
  end for
  end
```

[a]Begin at the root vertex and label all the adjacent vertices. Then for each of those vertices, label their unlabeled adjacent vertices, and so on, until all vertices are labeled.

Fig. 12 Algorithm MARK

$T_j(v)$ are in the same equivalence class as the parent of v. This case is covered in the second item of the algorithm.

3. There are two non-full trees $T_j(v)$, $T_l(v)$ with one hole each. If $j = 0$ or $l = 0$ then the equivalence class already exists, this case is covered in the first item of

the algorithm. Else two equivalence classes will be created, one contains the bad vertices of $T_j(v)$ and the other one contains the bad vertices of $T_l(v)$. This case is covered in the third item of the algorithm.

4. There are two non-full trees, $T_l(v)$ has one hole and $T_j(v)$ has at least two holes. If v is not a branch vertex, then it is exactly as the former case. Else, only $T_j(v)$ contains bad vertices and they are in an equivalence class by themselves. This case is covered in the fourth item in the algorithm.

Thus, the only nontrivial step in algorithm MARK is the check for every $v \in V_R \setminus S_0$ and $u \in \text{seen}(v)$ whether they are equivalent.

Theorem 6.16 *The feasibility of* T-PPM *on a tree is decidable in linear time.*

Proof Let $I = (G, A^{(0)}, A^{(*)})$ be a T-PPM instance where G is a tree. I is feasible if $A^{(0)}$ is such that every pebble is located on vertex of the same equivalence class as its terminal. The partition of V_R into equivalence classes can be done in linear time using the algorithm MARK.

The Algorithm uses the values of seen(v) and holes$(T_j(v))$, $v \in V_R \setminus S_0$. These values can be calculated in linear time as follows. seen(v) requires to traverse the tree and calculate and remember it for every $v \in V_R \setminus S_0$. For example, starting from the root and going to the leaves, updating the value of the parent when reaching a vertex in $V_R \setminus S_0$. holes$(T_j(v))$ are calculated for each tree $T_j(v) \in F(v)$ and can be calculated in $O(n)$ using a bottom-up approach. Starting from the leaves and moving up, calculate and remember these values for each vertex. This will require $O(n)$ time and $O(n)$ memory space. The values of holes$(T_j(v))$ are also in use when checking the conditions of Lemma 6.11. □

An example that illustrates algorithm Mark is given in Appendix 1.

7 T-m-PM on a Tree

In this section we analyze T-m-PM on a tree, and give a method to decide on the feasibility of a given instance of the problem. Auletta et al. used in [3] a partition of G into equivalence classes to solve the feasibility question of PPM. In Sect. 6 we generalized this result to the feasibility of T-PPM. Here use this result in the more general T-m-PM and show that partitioning $V_R \setminus S_0$ into equivalence classes gives a necessary and sufficient condition for equivalence also in T-m-PM. Algorithm MARK (see Fig. 12) is used for this purpose. The main result of this section is the necessary condition.

Lemma 7.1 *Let $u, v \in V_R \setminus S_0$ satisfy the following condition: there exists $w \in V \setminus \{u, v\}$ such that P_{uv} and P_{vw} share at least one edge, and there exists a plan f that moves the pebble from u to v and the other pebble from v to w (possibly among other changes). Then there exists a (u, v) pebble swap.*

Proof A similar lemma, where there are no transshipment vertices is Lemma 6 in [3]. Its proof holds also for T-m-PM, as we now describe. We only provide a brief review of the needed changes. For the proof to hold we need to show two things:

- The proof holds if all branch vertices are transshipment vertices.
- The proof holds if there are transshipment vertices between the regular vertices that host pebbles participating in proof.

Let $y \in V(P_{uv} \cap P_{vw})$ be the closest branch vertex to w (it is also the closest to u). Let $x \in V_R(P_{yv})$ be the closest vertex to y. Let $z \in V_R(P_{yu})$ be the closest vertex to y. The proof concerns only with the vertices of the paths P_{uv} and P_{vw}, and only the vertices x, y, and z need special care. The rest of the graph can have transshipment vertices without affecting the proof.

For the first item, notice that if a regular vertex is replaced with a transshipment vertex it reduces the number of holes in the graph, thus (or keep the same) the general movement possibilities on the graph. Therefore if the proof holds when all branch vertices are transshipment vertices, it will also hold when some of the branch vertices are transshipment vertices, and some are regular vertices. y is the only branch vertex containing a pebble among the three vertices considered in the proof. All the other branch vertices only let the pebbles pass through them, the same role can be done also by transshipment vertices. However, since y hosts a pebble only for a purpose of moving it to clear the path, y can be replaced with a transshipment vertex without changing the proof—it will not host a pebble at the first place. Moreover, the claim that every plan has a simple plan does not depend on the existence of transshipment vertices. Hence the movement of single pebble in a simple plan is also feasible.

For the second item, notice that transshipment vertices in parts of the graph that are not involved in the proof do not affect the proof since they do not affect the movement possibilities on the graph. x and z are the vertices adjacent to y, so there is a pebble move from one to another when the other is a hole (and y is a hole). Exactly the same conditions holds also if there are transshipment vertices between $x, y, z \in V_R$. □

Lemma 7.2 *Let f be a cyclic pebble swap on $(v_1, \ldots, v_k)$. Then there exists $1 < i \le k$ such that $P_{v_{i-1}v_i}$ and $P_{v_i v_{i+1}}$ share an edge (the indices are considered mod k).*

Proof Consider the tree induced by $\{v_1, \ldots, v_k\}$. Let v_i be a leaf of this tree, then the claim holds for i. □

Lemma 7.3 *Let f be a cyclic pebble swap on $(v_1, \ldots, v_k)$. Then there exists $1 \le j \le k$, for which there is a (v_j, v_{j+1}) pebble swap.*

Proof By Lemma 7.2 there exists j such that the conditions of Lemma 7.1 hold for $u = v_j$ and $v = v_{j+1}$, where $w = v_{j+2}$. Therefore, there exists a (v_j, v_{j+1}) pebble swap. □

Lemma 7.4 *Let f be a cyclic pebble swap on $(v_1, \ldots, v_k)$. Then there is a pebble swap for every two vertices v_i, v_j $i, j \in \{1, \ldots, k\}$.*

Proof By Lemma 7.3 there exists $1 \le j \le k$ for which there is a (v_j, v_{j+1}) pebble swap, g. Call the pebble located on a vertex v, P_v. For swapping the pebbles P_{v_p} and P_{v_q}, $1 \le p < q \le k$ do the following:

1. Execute $f^{(j-p)}$
   The plan $f^{(j-p)}$ moves P_{v_p} to v_j.
2. Execute $(f^{-1}g)^{(q-p-1)}$
   The plan $(f^{-1}g)^{(q-p-1)}$ keeps P_{v_p} on v_j and moves all the other pebbles (according to their original order) so P_{v_p} is on v_j and P_{v_q} is on v_{j+1} when the plan is finished.
3. Execute g
   When g is executed on the third step, P_{v_p} is on v_j and P_{v_q} is on v_{j+1} so they are swapped by g.
4. Execute $(gf)^{(q-p-1)}$
   The plan $(gf)^{(q-p-1)}$ brings all the other pebbles back, i.e. $(f^{-1}g)^{(q-p-1)} \times (gf)^{(q-p-1)} = \mathcal{I}$.
5. Execute $f^{(p-j)}$
   The plan $f^{(p-j)}$ moves back all the pebbles to their original locations, besides P_{v_q} and P_{v_p} that where swapped ($f^{(j-p)} f^{(p-j)} = \mathcal{I}$). When the plan ends P_{v_p} is located on v_q and P_{v_q} is located on v_p.

The process is illustrated in Example 7.5. □

Example 7.5 Consider a cyclic pebble swap $f = (a, b, c, d, e)$ and assume we want to swap the pebbles x and y initially located on vertices b and e. By Lemma 7.3, there is a plan g that swaps two adjacent pebbles, let them be c, d. Table 1 shows the color arrangement after every plan. Using the notation of Lemma 7.4, in this example $j = 3$, $p = 2$ and $q = 5$. The first and the last steps are executed $j - p = 1$ time; the second and fourth steps are executed $q - p - 1 = 2$ times. We mark $x = P_b$ and $y = P_e$ so it will be easier to track the moves.

Corollary 7.6 *The vertices of a cyclic pebble swap are in the same equivalence class.*

We say that f and g are *color equivalent with respect to A* if $T(f, A) = T(g, A)$.

Lemma 7.7 *Let f be a plan, then there exists a color equivalent plan $g = g_r \cdots g_1$, where $g_1, \dots, g_r$ are color swaps.*

Proof By Observation 3.4 f consists of one or more cyclic pebble swaps $f_1, \dots, f_\mu$ (Since the final locations of the pebbles are the same as the initial locations, every pebble either returns to its initial location or takes the initial location of another pebble, which in turn does the same. So the pebbles switch locations in a cycle of one or more pebbles.)

Define $N(f_i)$ as the number of vertices for which the color of the pebble they host is changed by f_i. Let $V_{f_i} = (v_1, \dots, v_{k_i})$ be the set of vertices hosting pebbles involved in the cyclic swap f_i. By Corollary 7.6 every two pebbles located in V_{f_i} can

Table 1 (b, e) pebble swap in a cyclic pebble swap on (a, b, c, d, e)

Algorithm step	Pebble arrangement	a	b	c	d	e
	$A^{(0)}$	P_a	x	P_c	P_d	y
1	$A^{(1)} = T(f, A^{(0)})$	y	P_a	x	P_c	P_d
2	$A^{(2)} = T(g, A^{(1)})$	y	P_a	P_c	x	P_d
2	$A^{(3)} = T(f^{-1}, A^{(2)})$	P_a	P_c	x	P_d	y
2	$A^{(4)} = T(g, A^{(3)})$	P_a	P_c	P_d	x	y
2	$A^{(5)} = T(f^{-1}, A^{(4)})$	P_c	P_d	x	y	P_a
3	$A^{(6)} = T(g, A^{(5)})$	P_c	P_d	y	x	P_a
4	$A^{(7)} = T(f, A^{(6)})$	P_a	P_c	P_d	y	x
4	$A^{(8)} = T(g, A^{(7)})$	P_a	P_c	y	P_d	x
4	$A^{(9)} = T(f, A^{(8)})$	x	P_a	P_c	y	P_d
4	$A^{(10)} = T(g, A^{(9)})$	x	P_a	y	P_c	P_d
5	$A^{(11)} = T(f^{-1}, A^{(10)})$	P_a	y	P_c	P_d	x

be swapped. Hence, in each V_{f_i} every two colors can be swapped. Since f_i is feasible $|S_c \cap V_{f_i}| = |T_c \cap V_{f_i}|$, $c = 1, 2, \ldots, m$.

Each f_i has a color equivalent plan with respect to $A^{(0)}$, $g_i = g_{i_{r_i}} \cdots g_{i_1}$, that consists of color swaps, $r_i = \frac{|N(f_i)|}{2}$. For plans f_i and f_j, the color equivalent plans g_i and g_j are independent since each color swap ends in the same color arrangement besides the swapped colors, $i, j = 1, \ldots, \mu$. Hence, f has a color equivalent plan with respect to $A^{(0)}$, $g = g_{\mu_{r_\mu}} \cdots g_{\mu_1} \cdots g_{1_{r_1}} \cdots g_{1_1}$, where each g_{i_h} is color swap in V_{f_i}, $h = 1, \ldots, r_i$. □

Corollary 7.8 *A (u, v) color swap is possible if and only if u and v are in the same equivalence class.*

Proof If u and v are in the same equivalence class, there is a (u, v) pebble swap, hence a (u, v) color swap. On the other hand, if there is a color swap, by Observation 3.4 it must be between vertices of a cyclic pebble swap. By Corollary 7.6, all the vertices of a cyclic pebble swap are in the same equivalence class. □

The direct implication of Corollary 7.8 is that there is no more strength in swapping two colors than in swapping two pebbles, it does not make any difference regarding the feasibility of the problem. The following theorem uses this property.

Theorem 7.9 *An instance of* T-m-PM *on a tree is feasible if and only if for each equivalence class $Z \subseteq V$, $|Z \cap S_c| = |Z \cap T_c|$, $c = 1, 2, \ldots, m$. The feasibility is decidable in linear time.*

Proof Let $I = (G, A^{(0)}, A^{(*)})$ be a T-m-PM instance where G is a tree. By Lemma 7.7, a plan f for I is color equivalent to a sequence of color swaps. By Corollary 7.8, a (u, v) color swap can be done if and only if u and v are in the same equivalence class. Therefore an instance is feasible if and only if for each equivalence class

$Z \subseteq V$, $|Z \cap S_c| = |Z \cap T_c|$, $c = 1, 2, \ldots, m$. The partition of V into equivalence classes can be done in linear time using Algorithm MARK (see Fig. 12). □

Since m-PM is a special case of T-m-PM we get also the following corollary.

Corollary 7.10 *The feasibility of m-PM on a tree is decidable in linear time.*

8 m-PM on a General Graph

G is 2-*vertex-connected* if $G \setminus v$ is connected for all $v \in V(G)$. (As a convention, if $|V(G)| = 1$ then G is 2-vertex-connected.)

Observation 8.1 *Let I be an m-PM instance where G is a 2-vertex-connected graph with at least one hole, then any pebble can move to any vertex.*

Proof Let $v, u \in V(G)$ be two vertices. Since G is 2-vertex-connected, there is a simple cycle C containing u and v. If C contains a hole then a pebble can move from u to v on this cycle using a series of cyclic pebble swaps. Else, there is a vertex $x_1 \in C$ with degree greater than 2, and a cycle C_1 containing x_1 and a hole. Move the hole on C_1 to x_1 using a series of cyclic pebble swaps. Now, a pebble can move from u to v on C using a series of cyclic pebble swaps. □

Our strategy to decide whether a given m-PM on a vertex-connected graph is feasible, is to reduce it to a T-m-PM on a tree.

On an instance with only one hole, the pebble movement is limited: a pebble cannot move between 2-vertex-connected components (see Lemma 8.2). Swaps of pebbles on different 2-vertex-connected components are not possible with only one hole (see Lemma 8.2). Therefore our main result concerns the case where there are at least two holes ($p < n - 1$). In Lemma 8.4 we show that two holes are enough to swap any two pebbles in a 2-vertex-connected component. It follows that the problem is feasible if it is possible to move the needed pebbles to each 2-vertex-connected component. In each 2-vertex component the movement is not limited. Thus, the feasibility problem is reduced to a problem of moving pebbles between the 2-vertex-connected components of G. This problem is very similar to the problem of moving pebbles on a tree, and we use Theorem 7.9 to decide whether it is feasible.

For any instance $I = (G, A^{(0)}, A^{(*)})$, we define a T-$m$-PM instance $I_T = (G_T, A^{(0)}, A^{(*)})$ on a tree G_T with transshipment vertices, such that I is feasible if and only if I_T is feasible.

Let G be a connected graph. Construct G_T as follows:
For every $H \subseteq G$, a maximal nontrivial 2-vertex-connected component (H is *nontrivial* if $V(H) > 1$):

1. Add a transshipment vertex h.
2. Remove every edge $e \in E(H)$.
3. Add the edges $\{(u, h) \mid u \in V(H)\}$.

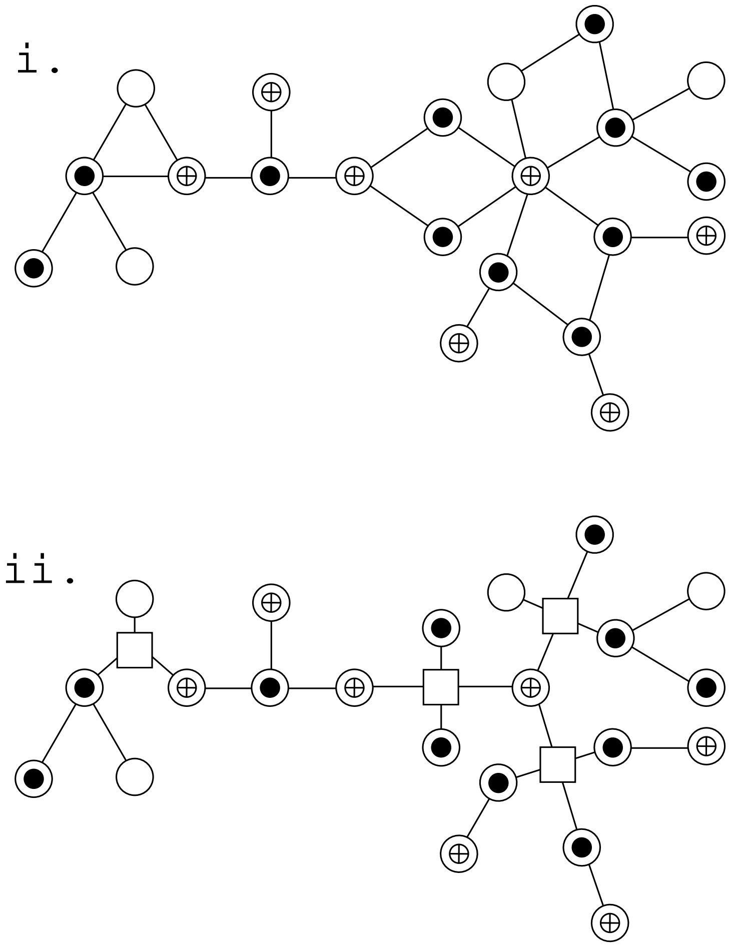

Fig. 13 Converting G (**i**) into G_T (**ii**)

G and G_T have a similar backbone structure where instead of 2-vertex-connected subgraphs in G, G_T has ‘stars’ with transshipment vertices as centers, see Fig. 13 for an example. Notice that G and G_T have the same number of holes and the same number of pebbles. Building I_T from I takes linear time.

Lemma 8.2 *Suppose G is 2-edge-connected and contains a cut vertex, then any pebble can be moved to any vertex if and only if $p < n - 1$.*

Proof Let v_c be a cut vertex. Let C_1 and C_2 be two 2-vertex-connected components such that $v_c \in C_1 \cap C_2$. Without loss of generality, assume that in the initial arrangement a pebble x is in C_1.

First we show that if $p = n - 1$ then x cannot pass from C_1 to C_2. x can move to C_2 only if a color arrangement where x is on v_c and the hole is in C_2, can be reached from the initial color arrangement. However, when moving x to v_c, v_c is a hole and so the hole is in C_1 when x is on v_c. The hole can move to v_c only if x moves back to C_1. Hence, there cannot be a color arrangement in which the hole is in C_2 and x is on v_c.

Now assume $p < n - 1$. For moving x from C_1 to C_2 first move one hole to C_1 and one hole to C_2; next move x to v_c (this is possible according to Observation 8.1). With the hole in C_2, x can be moved to any vertex in C_2. □

Definition 8.3 Let H be a 2-vertex-connected graph. Let G be a graph where $V(G) = V(H) \cup \{v_a\}$ and $E(G) = E(H) \cup \{(v_a, v)\}$, where $v \in V(H)$. G is called *H with an attached edge*.

The following lemma shows that any two pebbles on a cycle can be swapped without changing the locations of the other pebbles, if there are at least two holes and an edge attached to the cycle. This property will be used in Lemma 8.6, when we analyze the moves on general graphs. The other direction, that some pairs of pebbles cannot be swapped if there are less than two holes, is proved in a way similar to the proof of Lemma 8.2, and it is omitted here.

Lemma 8.4 *If $p < n - 1$ and G is a cycle with an attached edge, then all vertices of $V(G)$ are in the same equivalence class.*

Proof We show that any two pebbles can be swapped, and this will prove that all vertices of $V(G)$ are in the same equivalence class. Assume, without loss of generality, that in the initial color arrangement $v_a \in S_0$. Let u be an occupied vertex on the cycle. Let f_u be a plan that moves the pebble located on u to v_a, and then moves the other pebbles back to their locations (f_u moves all the pebbles on the cycle in clockwise direction, until the pebble located on u is on v; then f_u moves this pebble to v_a; finally, it moves all the pebbles on the cycle in counterclockwise direction back to their initial locations). Let g_u be a plan that moves a pebble from v_a to u, and then moves the other pebbles back to their locations (g_u moves the hole to u; then g_u moves all the pebbles on the cycle in clockwise direction, until the hole is located on v; then move the pebble from v_a to v; finally, it moves all the pebbles on the cycle in counterclockwise direction back to their initial locations, and the pebble that was on v_a is on u). Let u and w be two arbitrary vertices on the cycle and call the vertex next to w, in clockwise direction, w_l. For a (u, w) pebble swap do as follows:

1. Execute f_u.
2. Move all the pebbles from w to u, one vertex in counterclockwise direction.
3. Execute g_w.
4. Execute f_{w_l}.
5. Move all the pebbles from u to w_l, one vertex clockwise direction.
6. Execute g_u.

Assume in the initial color arrangement pebble 1 is on u and pebble 2 is on w. Figure 14 is an example of such a swap. □

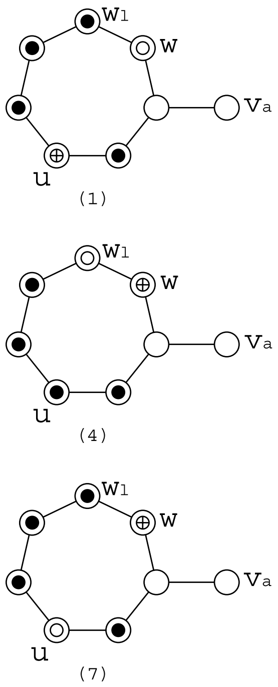

Fig. 14 Swapping pebbles w and u on a cycle with an attached edge. To help follow the steps, the swapped pebbles are marked with different texture

Lemma 8.5 *If $p < n - 1$ and G is a graph which contains a cycle with an attached edge, then all the vertices of $V(G)$ are in the same equivalence class.*

Proof The proof is very similar to the proof of Lemma 8.4. In this case the first step is to move one of the pebbles which do not take part in the swap to the vertex not on the cycle, then follow the steps described in Lemma 8.4 on the cycle, and finally move the pebble back to its initial location. □

Lemma 8.6 *Consider a color arrangement A on G, a vertex $a \in V(G) \setminus S_0$ and a vertex $b \in S_0$. Let $a' \in V(G_T) \setminus S_0$ and $b' \in S_0$ be the corresponding vertices in G_T. There is a plan f_{ab} that moves the pebble from a to b on G, without changing the final locations of the other pebbles, if and only if there is a plan $f'_{a'b'}$ that moves the pebble form a' to b' on G_T, without changing the final locations of the other pebbles.*

Proof The first direction of the proof follows from the nature of the "stars" in G_T. In a star graph every vertex is connected to the other vertices of the star, this allows the same pebble movement as if it was a clique. In a clique there is a direct connection between the vertices, while in a star the connection is through the transshipment vertex. However, it does not affect the possible pebble moves. If there is a plan f_{ab} that moves the pebble from a to b on G, it will remain feasible if we add edges to G

and convert each 2-vertex-connected component into a clique. Hence, there is also a plan $f'_{a'b'}$ that moves the pebble from a' to b' on G_T.

The other direction follows from the fact that there are no more movement possibilities in G_T than in G. If a' and b' belong to the same star, then a and b belong to the same 2-vertex component and by Observation 8.1 f_{ab} exists.

If a' and b' are not in the same star, $f'_{a'b'}$ might include moves from vertices included and not included in stars. Since $\text{holes}(G) = \text{holes}(G_T)$ the moves that do not use vertices included in a star are feasible also on G. When $f'_{a'b'}$ includes moves from/to vertices within stars, the feasibility of the moves in G follows from Lemma 8.2 (if the 2-vertex-connected component is part of a 2-edge-connected component) and from Lemma 8.5 (if the 2-vertex-connected component contains a cycle). □

Corollary 8.7 *I is feasible if and only if I_T is feasible.*

Proof I is feasible if and only if there is a plan f that solves it, and by Lemma 8.6 such a plan exists if and only is there is a plan f' that solves I_T, which means I_T is feasible. □

Remark 8.8 When G is biconnected and $p = n - 1$ the feasibility of m-PM is decidable in linear time. The proof is based on an analysis of the graph structure which implies the possible feasible plans.

Theorem 8.9 *Feasibility of m-PM is decidable in linear time.*

Proof If $p < n - 1$ let $I = (G, A^{(0)}, A^{(*)})$ be an m-PM instance where G is a connected graph. Construct I_T in linear time. By Theorem 7.9, the feasibility of I_T can be decided in linear time. By Corollary 8.7 I_T is feasible if and only if I is feasible.

If $p \geq n - 1$ and G is not biconnected or if $p > n - 1$ then the pebble movement is very limited and the feasibility is easily decidable. If $p = n - 1$ and G is biconnected then by remark 8.8 the feasibility is decidable in linear time. □

9 Final Remarks

We studied the feasibility of m-PM and obtained the following results. When G is a tree, Theorem 7.9 gives a necessary and sufficient condition for feasibility. Given a specific instance of the problem on a tree, there is an algorithm which divides $V(G) \setminus S_0$ into equivalence classes. The instance is feasible if and only if for every equivalence class the number of vertices of each color equals the number pebbles of the same color.

When G is a general graph, $p < n - 1$, Theorem 8.9 transforms the feasibility question to a similar question on a tree. The transformation is done in linear time.

It would have been interesting to estimate the number of moves as a function of the input parameters (m, p, n), but it is not part of this study.

Appendix 1: An Illustrative Example of Algorithm Mark

The following example demonstrates how Algorithm Mark works. Consider the instance of 2-PM on a tree given in Fig. 15.

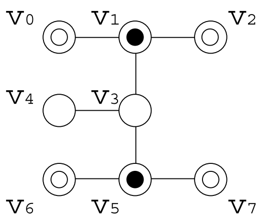

Fig. 15 An instance of 2-PM

We set v_0 as the root. Each row in Table 2 represents the classes each time the algorithm completes the loop that handles a vertex, v_i, $i = 0, 1, 2, 3, 4, 5$. We omit the steps of the unoccupied vertices since they have no effect.

Table 2

Loops	Class C_1	Class C_2	Class C_3
Loop 0	$\{v_0\}$	$\{v_1\}$	
Loop 1	$\{v_0, v_2\}$	$\{v_1, v_5\}$	
Loop 2	$\{v_0, v_2\}$	$\{v_1, v_5\}$	
Loop 5	$\{v_0, v_2\}$	$\{v_1, v_5\}$	$\{v_6, v_7\}$
Loop 6	$\{v_0, v_2\}$	$\{v_1, v_5\}$	$\{v_6, v_7\}$
Loop 7	$\{v_0, v_2\}$	$\{v_1, v_5\}$	$\{v_6, v_7\}$

Appendix 2: Table of Notation

Notation	Meaning	Reference
A	The vector $(S_0(A), \ldots, S_m(A))$, denotes the color arrangement of pebbles on the vertices of G	
$A^{(0)}$	An initial color arrangement	Definition 3.1
$A^{(*)}$	A final color arrangement	Definition 3.3
$\deg(v)$	The number of edges connected to $v \in V$	Definition 6.5
$\mathrm{dist}(u, v)$	Distance between two adjacent vertices u, v. For $u \in V_R$ $\mathrm{dist}(u, v) = 0.5$ if $v \in V_T$ and $\mathrm{dist}(u, v) = 1$ if $v \in V_R$	Definition 6.7
$\mathrm{dist}(a, b)$	The distance $\mathrm{dist}(a, b)$ as the sum of	Definition 6.7

Notation	Meaning	Reference
	$\text{dist}(u, v)$ over the inspired path P_{ab} in G.	
f	A plan—a sequence of moves	Definition 3.2
$F(u)$	The forest obtained from G by removing u and the edges connected to it	Definition 6.6
g	A plan—sequence of moves	Definition 3.2
$\text{holes}(Q)$	The number of holes in $Q \subseteq G$	Definition 6.8
$I = (G, A^{(0)}, A^{(*)})$	An instance of a pebble motion problem	Definition 3.3
P_{uv}	The unique path between u and v excluding u, v	Definition 6.2
$R(F, v)$	$F \backslash \text{tree}(F, v)$, where F is a forest	Definition 6.6
s	The vector $(s_0, \dots, s_m)$, indicates the number of pebbles of each color	Definition 3.1
s_c	The number of pebbles of color c s_0 is the number of holes	Definition 3.1
S_c	The set of vertices hosts pebbles of color c S_0 is the set of holes	Definition 3.1
$\text{seen}(v, Q)$	$\{u \in V_R(Q) \setminus S_0 \mid Q \in F(v), P_{vu} \subseteq S_0\}$	Definition 6.8
$T_i(v)$	The tree rooted at the i^{th} child of v	
$T(f, A)$	The color arrangement result of executing f on A	Definition 3.2
$\text{tree}(F, v)$	The tree of F containing v, where F is a forest	Definition 6.6
$V_T(G)$	The set of transshipment vertices of G	
$V_R(G)$	The set of regular vertices of G	

Appendix 3: Table of Problems

Notation	Meaning	Reference
m-PM	Multi color pebble motion problem	
PPM	Permutation pebble motion problem	
T-m-PM	Multi color pebble motion problem on a graph with transshipment vertices	Definition 6.3
T-PPM	Permutation pebble motion problem on a graph with transshipment vertices	Definition 6.3

References

1. Archer, A.F.: Modern treatment of the 15 puzzle. Am. Math. Mon. **106**, 793–799 (1999)
2. Auletta, V., Persiano, P.: Optimal pebble motion on a tree. Inf. Comput. **165**, 42–68 (2001)
3. Auletta, V., Monti, A., Parente, D., Persiano, G.: A linear time algorithm for the feasibility of pebble motion on trees. Algorithmica **23**, 223–245 (1999)
4. Christofides, N., Colloff, I.: The rearrangement of items in a warehouse. Oper. Res. **21**, 577–589 (1973)
5. Călinescu, G., Dumitrescu, A., Pach, J.: Reconfigurations in graphs and grids. SIAM J. Discrete Math. **22**, 124–138 (2008)
6. Dumitrescu, A., Pach, J.: Pushing squares around. In: Graphs and Combinatorics, pp. 37–50 (2006)
7. Goldreich, O.: Shortest move-sequence in the generalized 15-Puzzle is NP-hard. Technical Report no. 792, Computer Science Department, Technion (1993)
8. Hayes, R.: The Sam Loyd 15-puzzle. Technical Report no. TCD-CS-2001-24, Computer Science Department, The University of Dublin, Trinity College (2001)
9. Jacobs, D.J., Hendrickson, B.: An algorithm for two-dimensional rigidity percolation: the pebble game. J. Comput. Phys. **137**, 346–365 (1997)
10. Jacobs, D.J., Rader, A.J., Kuhn, L.A., Thorpe, M.F.: Protein flexibility predictions using graph theory. Proteins Struct. Funct. Genet. **44**, 150–165 (2001)
11. Kornhauser, D., Miller, G., Spirakis, P.: Coordinating pebble motion on graph, the diameter of permutation groups, and applications. In: Proceedings of the 25-th Symposium on the Foundations of Computer Science (FOCS), pp. 241–250 (1984)
12. Milans, K., Clark, B.: The complexity of graph pebbling. SIDMA **20**(3) (2006) (archived in 2005 at http://arxiv.org/abs/math/0503698)
13. Loyd, S.: Mathematical Puzzles of Sam Loyd. Dover, New York (1959)
14. Parberry, I.: A real-time algorithm for the $(n^2 - 1)$-puzzle. Inf. Process. Lett. **56**, 23–28 (1995)
15. Papadimitriou, D., Raghavan, P., Sudan, M., Tamaki, H.: Motion planning on a graph. In: Proceedings of the 35-th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 511–520 (1994)
16. Stagemann, R.: Rob's puzzle page. http://home.comcast.net/~stegmann/sliding.htm
17. Ratner, D., Warmuth, M.: Finding a shortest solution for the $(N \times N)$-extension of the 15 puzzle is NP-hard. J. Symb. Comput. **10**, 111–137 (1990)
18. Reinefeld, A.: Complete solution of the eight-puzzle and the benefit of node ordering in IDA. In: Proceeding of the International Joint Conference on Artificial Intelligence, pp. 248–253 (1993)
19. Wilson, R.M.: Graph puzzles, homotopy, and the alternating group. J. Comb. Theory Ser. B **16**, 86–94 (1974)