Journal of Artificial Intelligence Research 79 (2024) 483-514 Submitted 07/2023; published 02/2024

An Algorithm with Improved Complexity for Pebble Motion/Multi-Agent Path Finding on Trees

Stefano Ardizzoni STEFANO.ARDIZZONI@UNIPR.IT
Irene Saccani IRENE.SACCANI@UNIPR.IT
Luca Consolini LUCA.CONSOLINI@UNIPR.IT
Marco Locatelli MARCO.LOCATELLI@UNIPR.IT
Dipartimento di Ingegneria e Architettura, Università di Parma, Parco Area delle Scienze, 181/A, Parma, Italy

Bernhard Nebel NEBEL@INFORMATIK.UNI-FREIBURG.DE
Institut für Informatik, Albert-Ludwigs-Universität Georges-Köhler-Allee 52, Freiburg, Germany

Abstract

The *pebble motion on trees* (PMT) problem consists in finding a feasible sequence of moves that repositions a set of pebbles to assigned target vertices. This problem has been widely studied because, in many cases, the more general Multi-Agent path finding (MAPF) problem on graphs can be reduced to PMT. We propose a simple and easy to implement procedure, which finds solutions of length $O(|P|nc + n^2)$, where n is the number of nodes, P is the set of pebbles, and c the maximum length of corridors in the tree. This complexity result is more detailed than the current best known result $O(n^3)$, which is equal to our result in the worst case, but does not capture the dependency on c and $|P|$.

1. Introduction

Multi-Agent Path-Finding (MAPF), also called pebble motion on graphs, or cooperative path-finding, is the problem of finding a collision-free movement plan for a set of agents (or pebbles) moving on a graph. This problem has been widely discussed, together with its many variants (Stern et al., 2019), on various types of graphs. For most graph classes, finding an optimal solution of MAPF (that is, a solution with a minimum number of moves) is NP-hard (Yu & LaValle, 2013). Instead, the complexity of checking MAPF feasibility depends on the specific graph class. For instance, it is polynomial on undirected graphs (Kornhauser et al., 1984), on strongly biconnected directed graphs (Botea & Surynek, 2015), and on strongly connected directed graphs (Ardizzoni et al., 2022). Instead, it is NP-hard in the general case of directed graphs (Nebel, 2020). Optimal and suboptimal algorithms have been proposed in the last forty years (Achá et al., 2022; Alotaibi & Al-Rawi, 2016; Ardizzoni et al., 2022; Auletta et al., 1999; Botea & Surynek, 2015; Botea et al., 2018; De Wilde et al., 2014; Kornhauser et al., 1984; Sharon et al., 2015).

We focus on one of the simplest versions of this problem, the *pebble motion on a tree* (PMT) (Auletta & Persiano, 2001; Auletta et al., 1999; Goraly & Hassin, 2010; Khorshid et al., 2011; Krontiris et al., 2013), which is defined as follows. Let $T = (V, E)$

be a tree with n vertices and P a set of distinct pebbles such that $|P| < n$, numbered $1, \ldots, |P|$, placed on distinct vertices. A *move* consists in transferring a pebble from its current position to an adjacent unoccupied vertex. The PMT consists in finding a sequence of moves that repositions all pebbles to assigned target vertices. In particular, we focus on the problem of finding a feasible solution, not necessarily optimal.

Although PMT is one of the simplest versions of MAPF, it is quite relevant. Indeed, various algorithms that solve MAPF on more general graphs are based on a reformulation of MAPF as a PMT, over a suitably defined tree (Ardizzoni et al., 2022; Goraly & Hassin, 2010; Krontiris et al., 2013). We will further discuss this in Section 6. In literature, there exist many complete sub-optimal algorithms for solving PMT. In particular, Kornhauser and coauthors (Kornhauser et al., 1984) present a procedure which solves it in $O(n^3)$ moves. However, the approach is not described algorithmically, but must be derived from a number of proofs in the paper (Röger & Helmert, 2012). This requires significant effort, and, to the best of our knowledge, Kornhauser's results have never been fully implemented. Auletta and coauthors (Auletta et al., 1999) present an algorithm for deciding the feasibility of PMT, from which a solution can be derived requiring $O(|P|^2(n-|P|))$ moves. However, the paper does not explicitly explain how to find such a solution. Korshid and coauthors (Khorshid et al., 2011) present an algorithm for PMT (called TASS) that is easy to understand and implement. However, solutions provided by TASS require $O(n^4)$ moves.

Therefore, after Kornhauser and coauthors in 1984, no one has proposed a clear and detailed algorithm with length-complexity $O(n^3)$. The result of Kornhauser is a fundamental step in the study of PMT complexity. However, under a more practical point of view, the implementation of a simple and efficient algorithm for solving PMT with length-complexity at least $O(n^3)$ remains an open problem. The aim of this paper is to address this problem by proposing an efficient, clear, and simple PMT solver, with a more detailed complexity result with respet to Kornhauser's.

In this work, we also deal with two variants of PMT: the *motion planning* problem and the *unlabeled* PMT. In the first one, a single marked pebble has to reach a desired target vertex, while non-marked pebbles are obstacles that need be moved out of the way to re-position the marked one (Auletta & Persiano, 2001; Papadimitriou et al., 1994; Wu & Grumbach, 2009, 2010). In the second one (also known as U-GRAPH-RP (Călinescu et al., 2008; Dumitrescu, 2013) or *anonymous MAPF* (Ma & Koenig, 2016)), pebbles are not labeled (i.e., there is a set of target positions D, and each pebble has to reach a vertex in D, not specified in advance).

Statement of contribution. We present three main contributions:

1. In Section 4, we propose the sub-optimal CATERPILLAR algorithm to solve the motion planning problem on a tree. It provides solutions with $O(nc)$ moves, where c is the maximum length of the corridors (i.e., paths whose internal nodes all have degree two, and whose end nodes have degree different from two). We are able to guarantee this complexity since, when we move the marked pebble p to its target, we only move the obstacles that are along the path of p, sliding them section by section from one subset of vertices to another, avoiding unnecessary motions.

2. In Section 5, we propose a sub-optimal algorithm for PMT (called *Leaves procedure for* PMT). The idea of the *Leaves procedure for* PMT is to use the CATERPILLAR

algorithm to move the pebbles to the leaves of the tree, used as intermediate targets. At the end, we solve an unlabeled PMT instance, which brings the pebbles to the original targets. This procedure is simpler and easier to implement than the one proposed by Kornhauser and coauthors (Kornhauser et al., 1984). In addition, we prove a more detailed complexity result than the one provided by Kornhauser and coauthors. Indeed, our algorithm finds solutions with a number of moves $O(|P|nc + n^2)$, which in the worst case is $O(n^3)$. Therefore, the number of moves of these solutions depends on the tree structure and the number of pebbles.

3. In Section 6, we discuss a variant of the PMT problem, called PMT *with transshipment vertices* (*ts*-PMT). This variant is relevant since a MAPF instance on a generic graph can be reduced to an instance of this problem. *ts*-PMT can be solved with the *Leaves procedure for* PMT with some minor modifications. This permits us to provide an upper bound for the solution length of MAPF on graphs.

2. Problem Definition

Let $T = (V, E)$ be a tree, with vertex set V and edge set E. We are also given a set P of pebbles and a set H of holes, and each vertex of T is occupied either by a pebble or by a hole, so that $|V| = |P| + |H|$. A *configuration* is a function $\mathcal{A} : P \cup H \to V$ that assigns to each pebble or hole the vertex occupied by it. A configuration is *valid* if it is one-to-one (i.e., each vertex is occupied by one and only one pebble or hole). The collection $\mathcal{C} \subset \{P \cup H \to V\}$ contains all valid configurations.

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

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

As mentioned in the Introduction, a *move* is the movement of a pebble along an edge. For each edge $(u, v) \in E$ we can define two possible moves, that are the two ordered pairs $u \to v$ and $v \to u$. We indicate with $\hat{E}$ the set of all the moves on tree T. Function $\rho : \mathcal{C} \times \hat{E} \to \mathcal{C}$ is a partially defined transition function such that $\rho(\mathcal{A}, u \to v)$ is defined if and only if v is empty (i.e., occupied by a hole). In this case $\rho(\mathcal{A}, u \to v)$ is the configuration obtained by exchanging the pebble or the hole in u with the hole in v. Notation $\rho(\mathcal{A}, u \to v)!$ means that the function is defined. In other words $\rho(\mathcal{A}, u \to v)!$ if and only if $(u, v) \in E$ and $\mathcal{A}^{-1}(v) \in H$. If $\rho(\mathcal{A}, u \to v)!$, then $\rho(\mathcal{A}, u \to v) = \mathcal{A}[u, v]$. Note that the hole in v moves along $v \to u$, while the pebble or hole on u moves on v.

We represent a *plan* as an ordered sequence of moves. It is convenient to view the elements of $\hat{E}$ as the symbols of a language. We denote by E^* the Kleene star of $\hat{E}$, that is the set of ordered sequences of elements of $\hat{E}$ with arbitrary length, together with the empty string ϵ:

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

E^* represents the set of all the plans. We extend the function $\rho : \mathcal{C} \times \hat{E} \to \mathcal{C}$ to $\rho : \mathcal{C} \times E^* \to \mathcal{C}$, by setting $(\forall \mathcal{A} \in \mathcal{C})\rho(A, \epsilon)!$ and $\rho(\mathcal{A}, \epsilon) = \mathcal{A}$. Note that ϵ is the trivial plan that keeps all pebbles and holes on their positions. We denote by $|f|$ the length of a plan f (i.e., the number of moves of f). Moreover, $(\forall s \in E^*, e \in \hat{E}, \mathcal{A} \in \mathcal{C})\ \rho(\mathcal{A}, se)!$ if and only if $\rho(\mathcal{A}, s)!$ and $\rho(\rho(\mathcal{A}, s), e)!$ and, if $\rho(\mathcal{A}se)!$, then $\rho(\mathcal{A}se) = \rho(\rho(\mathcal{A}s), e)$. We define an equivalence relation $\sim$ on E^*, by setting, for $s, t \in E^*$, $s \sim t \leftrightarrow (\forall \mathcal{A} \in \mathcal{C})\, \rho(\mathcal{A}, s) = \rho(\mathcal{A}, t)$. In other words, two plans are equivalent if they reconfigure pebbles and holes in the same way. Given a configuration $\mathcal{A}$ and a plan f such that $\rho(\mathcal{A}, f)!$, a plan f^{-1} is a *reverse* of f if $\mathcal{A} = \rho(\rho(\mathcal{A}, f), f^{-1})$ (i.e., f^{-1} moves each pebble and hole back to their initial positions). We can also write $ff^{-1} \sim \epsilon$, so that f^{-1} behaves like a right-inverse.

Proposition 2.1. *For any configuration $\mathcal{A} \in \mathcal{C}$ and any plan $f \in E^*$, such that $\rho(\mathcal{A}, f)!$, there exists a reverse plan f^{-1} such that $|f| = |f^{-1}|$.*

Proof. The thesis can be proved by induction as follows. If $f = \epsilon$ the thesis is trivial, since $f^{-1} = \epsilon$. If $f \in \hat{E}$, then there exist $u, v \in V$ such that $f = u \to v$ and $f^{-1} = v \to u$. Now, suppose that for any $f \in E^*$ such that $|f| = n > 1$ there exists a reverse f^{-1} such that $|f| = |f^{-1}|$. We prove that the thesis is verified also for each plan of length $n + 1$. Indeed, if $|f| = n + 1$ then $f = ge$ with $|g| = n$ and $e \in \hat{E}$. Therefore, there exist g^{-1} and e^{-1} the corresponding reverse plans such that $|g^{-1}| = |g|$ and $|e^{-1}| = |e|$, and we can define $f^{-1} := e^{-1}g^{-1}$ which is a reverse plan of f:

$$ff^{-1} = gee^{-1}g^{-1} \sim gg^{-1} \sim \epsilon.$$

Moreover, $|f^{-1}| = |g^{-1}| + |e^{-1}| = n + 1$. □

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

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

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

Definition 2.2. ***(PMT problem)****. Given a tree T, a pebble set P, an initial valid configuration $\tilde{\mathcal{A}}^s$, and a final valid configuration $\tilde{\mathcal{A}}^t$, find a plan f such that $\tilde{\mathcal{A}}^t = \rho(\tilde{\mathcal{A}}^s, f)$.*

We also focus on two relaxations of the PMT problem: the *motion planning problem* and the *unlabeled PMT problem.* The former consists in finding a plan such that a single marked pebble reaches a desired target vertex, avoiding collisions with other pebbles, which are movable *obstacles.* The latter consists in finding a plan such that each pebble reaches any vertex belonging to the set of targets.

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

Definition 2.4. ***(Unlabeled PMT problem)****. Let $T = (V, E)$ be a tree, P a set of pebbles, and $\tilde{\mathcal{A}}^s$ an initial valid configuration. Given $D \subset V$, a set of destinations such that $|D| = |P|$, find a plan f such that $D = \rho(\tilde{\mathcal{A}}^s, f)(P)$.*

2.1 Notation

Let $T = (V, E)$ be a tree with n nodes and let $u \in V$. We denote by $F(u)$ the connected components of the *forest* obtained from T by deleting u. For some $v \in V \setminus \{u\}$, $T(F(u), v)$ is the connected component containing v, while $\mathcal{C}(F(u), v)$ is the set of remaining connected components of $F(u)$ excluding $T(F(u), v)$.

Given two nodes $a, b \in V$, we denote by π_{ab} the set of vertices of the unique path in T from a to b, and with $d(a, b)$ the length of this path. In particular, π_{ab} is a *corridor* if a and b have degree different from two, while the internal nodes of the path all have exactly degree two. Moreover, $\pi_{ab} \setminus \{a\}$ is the set of all vertices of π_{ab}, except for a. We denote by $C(T)$ the set of all corridors of T, and by c_1 the maximum corridor length:

$$c_1(T) = \max\{d(a, b) : \pi_{ab} \in C(T)\}.$$

Let $J \subset V$ be the set of *junctions* (i.e., nodes with degree greater than two). We define the subclass of corridors $\bar{C}(T) \subset C(T)$ that have only junctions as endpoints, and we denote by $c_2(T)$ the maximum length of this subclass of corridors. Obviously, $c_2(T) \leq c_1(T)$. Note that corridors in $C(T) \setminus \bar{C}(T)$ are those for which at least one endpoint is a leaf of the tree.

Let us define the distance between a node s and a subset of nodes $W \subset V$ as

$$d(s, W) = \min_{a \in W} d(s, a).$$

Furthermore, we define the distance of W_1 from W_2 as

$$d(W_1, W_2) = \sum_{a \in W_1} d(a, W_2).$$

Then, a *subset of V of cardinality q closest to* W_1 is a set W such that

$$W \in \arg \min_{\substack{W \in \mathcal{P}(V) \\ |W| = |q|}} d(W, W_1),$$

where $\mathcal{P}(V)$ denotes the power set of V (i.e., the set of all its subsets).

2.2 Assumptions

In all the problems we focus on, we assume that the following assumption holds:

$$|H| \geq c(T), \tag{2}$$

where $c(T) \in \mathbb{N}$ is a constant depending on the structure of the tree T. In particular, if T is a path graph (i.e., a tree with two nodes of degree 1, and the remaining $n-2$ nodes of degree 2) $c(T) := c_1(T)$. Otherwise, in all other cases, $c(T)$ is defined as follows:

$$c(T) := \max\{c_1(T) + 1,\ c_2(T) + 2\}. \tag{3}$$

From now on, when we write c_1, c_2 and c without the indication of a tree within the parenthesis, it is given as understood that the three parameters refer to the tree T on which we are solving the PMT problem, while for other trees we will explicitly indicate them within parenthesis.

Kornhauser and coauthors (Kornhauser et al., 1984) showed that Assumption (2) is a necessary and sufficient condition for the feasibility of PMT on trees. Obviously, it follows that this condition is also sufficient for the feasibility of any instance of the unlabeled PMT problem, and of the motion planning problem.

2.3 Basic Plans

Let T be a tree and $\mathcal{A}$ a configuration on it. Given a path $\pi_{vw} = v\, u_2 \cdots u_{n-1}\, w$, we define the following plans:

1. If $w \in \mathcal{A}(H)$, Bring hole from w to v is defined as

$$\alpha_{vw} = (u_{n-1} \to w, \ldots, v \to u_2). \tag{4}$$

   In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. For instance, see the example of Figure 1.

2. If $v \in \mathcal{A}(P)$ and $\pi_{vw} \setminus \{v\} \subset \mathcal{A}(H)$, Move Pebble from v to w is defined as

$$\beta_{vw} = (v \to u_2, \ldots, u_{n-1} \to w). \tag{5}$$

   For instance, see the example of Figure 2.

3. Unlabeled PMT Problem

Let P be a set of unlabeled pebbles on a tree $T = (V, E)$. Let $\mathcal{A}^s$ be the initial configuration and D be the set of destinations. We denote by $S = \mathcal{A}^s(P)$ the set of pebbles initial positions. The goal of the *unlabeled pebble motion on trees* is to move each pebble from its initial position to any position of D. In the following, we introduce an algorithm presented by Kornhauser and coauthors (Kornhauser et al., 1984).

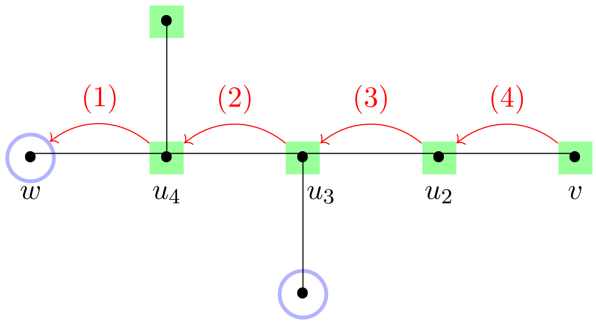
1. If V is empty: terminate the procedure.

2. Select any leaf v of T.

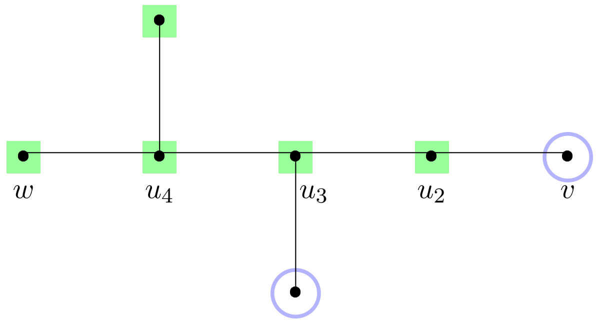
   - If $v \in S \cap D$ or $v \notin S \cup D$, then "prune" v from T, i.e., $V = V \setminus \{v\}$, and set $S = S \setminus \{v\}$, $D = D \setminus \{v\}$. Go to Step 1.
   - If $v \in D \setminus S$, select a pebble p on a vertex w such that

$$w \in \arg\min_{v' \in S} d(v', v).$$

     Let p be the pebble on w. By definition of w, path π_{wv} contains only pebble p. Therefore, move p to v and update $S = S \setminus \{w\}$. Then, "prune" v from T, i.e., $V = V \setminus \{v\}$, and set $D = D \setminus \{v\}$. Go to Step 1.



(a) Initial configuration. The red edges represent plan $\alpha_{vw} = (u_4 \to w, u_3 \to u_4, u_2 \to u_3, v \to u_2)$.



(b) Final configuration after bringing the hole from w to v.

Figure 1: Example of Bring hole from w to v. Green squares represent pebbles, blue circles represent holes.

- if $v \in S \setminus D$. Find an unoccupied vertex u which is at minimum distance from v on T:

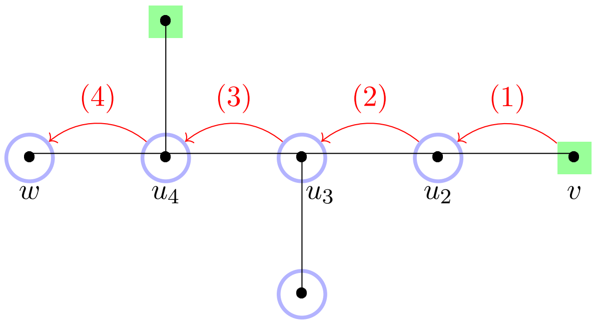
$$u \in \arg\min_{v' \in V \setminus S} d(v', v).$$

  Then, path π_{vu} has pebbles on each vertex except u. Move each pebble on the path π_{vu} towards u with plan α_{vu} as defined in (4). This makes v unoccupied and u occupied. Then, set $S = (S \setminus \{v\}) \cup \{u\}$, "prune" v from T, i.e., $V = V \setminus \{v\}$, and go to Step 1.

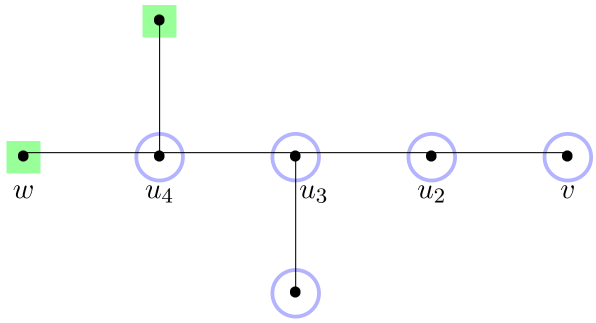
Since at most n moves are made at each execution of Step 2, and Step 2 is executed n times (at each iteration the cardinality of V is decreased by one), the total number of required moves is at most n^2. Therefore, the complexity of this algorithm is $O(n^2)$.

3.1 Gather Holes Problem

In this subsection, we focus on a particular case of the *unlabeled* PMT problem: the *gather holes problem*. Let $T = (V, E)$ be a tree with n nodes, P a set of pebbles, and H



(a) Initial configuration. The red edges represent plan $\beta_{vw} = (v \to u_2, u_2 \to u_3, u_3 \to u_4, u_4 \to w)$.



(b) Final configuration after moving the pebble from v to w.

Figure 2: Example of MOVE PEBBLE FROM v TO w. Green squares represent pebbles, blue circles represent holes.

the set of holes. Let $\bar{T} = (\bar{V}, \bar{E})$ be a subtree with $q = |\bar{V}| \leq |H|$. Then, *gather holes in* $\bar{T}$ consists in bringing q holes of the tree to the nodes of $\bar{T}$.

Definition 3.1. ***(Gather holes problem).*** *Let T be a tree and $\bar{T} = (\bar{V}, \bar{E})$ be a subtree. Let P be a set of pebbles, and $\tilde{\mathcal{A}}^s$ an initial valid configuration. Find a plan f such that $\bar{V} \cap \rho(\tilde{\mathcal{A}}^s, f)(P) = \emptyset$.*

Even if *gather-holes* can be solved by the previous algorithm, we present a specific procedure that allows finding a feasible solution with a lower time-complexity. The solution plan removes pebbles from vertices of $\bar{T}$, and replaces them with holes. To search for a short plan, it is convenient to bring holes that are already close to $\bar{V}$. Therefore, we choose the holes in a set M such that

$$M \in \arg \min_{W \subset \mathcal{A}^s(H):\ |W|=q} d(W, \bar{V}), \tag{6}$$

i.e., M is a subset of vertices with cardinality q closest to $\bar{V}$ and containing only holes of the initial configuration. Denote by $\tilde{H}$ the set of holes on M ($\mathcal{A}^s(\tilde{H}) = M$). Then, we want to find a plan f such that $\bar{V} = \rho(\mathcal{A}^s, f)(\tilde{H})$. Moreover, let $\bar{H} = \{h \in \tilde{H} : \mathcal{A}^s(h) \notin \bar{V}\}$.

Denoting by $\bar{V}_P = \mathcal{A}^s(P) \cap \bar{V}$ the initial set of occupied vertices of $\bar{T}$, we can proceed as follows:

1. If $\bar{V}_P$ is empty: terminate the procedure;

2. Select $h \in \bar{H}$, let $v = \mathcal{A}^s(h)$ and $u \in \bar{V}_P$ be a closest node of $\bar{V}_P$ to v:

$$u \in \arg\min_{v' \in \bar{V}_P} d(v', v). \tag{7}$$

   Denote by p the pebble on u. If $\pi_{uv} \cap \bar{V} = \{u\}$, then move each pebble on the path π_{uv} towards v with plan α_{uv} defined in (4), and update $\mathcal{A}^s = \rho(\mathcal{A}^s, \alpha_{uv})$. Otherwise, let w be the closest node to v of $\pi_{uv} \cap \bar{V}$: move p from u to w with plan β_{uw} defined in (5) (note that $\pi_{uw} \setminus \{u\}$ contains only unoccupied vertices); then, move each pebble on path π_{wv} towards v with plan α_{wv}. Finally, update $\mathcal{A}^s = \rho(\mathcal{A}^s, \beta_{uw}\alpha_{wv})$. This makes u unoccupied.

3. Update $\bar{H} = \bar{H} \setminus \{h\}$ and $\bar{V}_P = \bar{V}_P \setminus \{u\}$. Go to Step 1.

Since at most n moves (with $n = |V|$) are made at each execution of Step 2, and Step 2 is executed at most q times (with $q = |\bar{V}|$), (since the cardinality of $\bar{V}_P$ is reduced by one at each iteration), we have the following complexity result.

Proposition 3.2. *The length complexity of the solution provided by the gather holes procedure is $O(nq)$.*

Figure 3 presents an example of the execution of the procedure just described. The blue circles represent the holes, and $\bar{H} = \{c, d, g\}$ is a subset of holes closest to subtree $\bar{T}$ (Figure 3a). Another possible choice for $\bar{H}$ is, for example, $\bar{H} = \{c, e, g\}$. Figure 3b shows the final configuration, in which the holes of $\bar{H}$ have been moved to the subtree.

4. Motion Planning on Trees

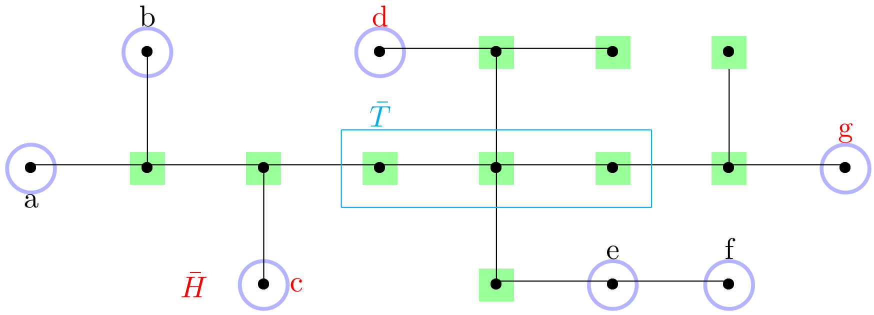
Let $T = (V, E)$ be a tree, P a set of pebbles and H a set of holes. We assume that condition (2) of Section 2.2 holds. Recall that

$$c := \begin{cases} c_1, & \text{if } T \text{ is a path graph,} \\ \max\{c_1 + 1,\, c_2 + 2\}, & \text{otherwise,} \end{cases} \tag{8}$$

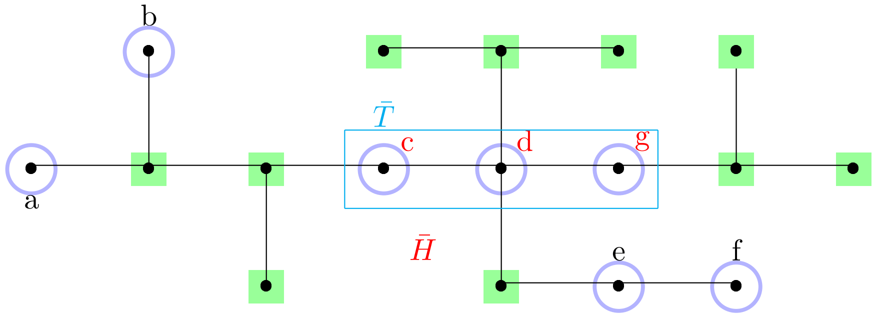
where c_1 is the maximum length of all the corridors and c_2 is the maximum length of the corridors with endpoints that are junctions. Let $\mathcal{A}^s$ be an initial valid configuration. Given a pebble $\bar{p}$ on $r = \mathcal{A}^s(\bar{p})$, and a target node $t \in V$, we show how to find a plan f such that $t = \rho(\mathcal{A}^s, f)(\bar{p})$. To do that, we analyze two cases:

A. $|\mathcal{A}^s(H) \cap T(F(r), t)| \geq c$, i.e., $T(F(r), t)$ contains at least c holes.

B. $|\mathcal{A}^s(H) \cap T(F(r), t)| < c$, i.e., $T(F(r), t)$ contains less than c holes.



(a) Initial configuration. We choose $\bar{H} = \{c, d, g\}$ the subset of holes closest to subtree $\bar{T}$.



(b) Final configuration after moving holes c, d and g to $\bar{T}$.

Figure 3: Example of *Gather hole* problem. We want to move three closest holes to the subtree $\bar{T}$.

For each of the two cases we present a solution procedure (*Procedure A* and *Procedure B*). The union of these procedures constitutes an algorithm to solve any instance of motion planning on trees, called the CATERPILLAR algorithm.

Case A. $T(F(r), t)$ contains at least c holes.

The main idea of the algorithm is to clear, piece by piece, the path that goes from r to t, allowing the pebble to reach the target. We identify intermediate junctions on π_{rt} (denoted by i_k), and "parking" positions (denoted by ℓ_k) which are neighbor nodes of i_k, but do not belong to π_{rt}. The pebble moves from one parking position to the next one, until it reaches the target. When the pebble is on ℓ_k, we move out of the way all the obstacles that are on $\pi_{i_k i_{k+1}}$, so that we can freely move the pebble from ℓ_k to ℓ_{k+1}. We identify a sequence of subsets of vertices on which the movement of the pebble from ℓ_k to ℓ_{k+1} will be defined: each of them will contain the path from i_k to i_{k+1} combined with the parking positions ℓ_k and ℓ_{k+1}. These subsets $(S_0, \ldots, S_m)$, called *caterpillar sets*, are delimited by the nodes (i_k, j_k, ℓ_k) (see Figure 4). In particular, they are defined as follows:

$$S_k = \pi_{i_k j_k} \cup \{\ell_k\} \cup \{\ell_{k+1}\}, \quad \forall k = 0, \ldots, m-1,$$

$$S_m = \pi_{i_m j_m} \cup \{\ell_m\}.$$

We can easily note that the union of all caterpillar sets is a caterpillar tree (i.e., a tree in which all the vertices are within distance 1 from a central path). Moreover, the following properties hold:

- the restriction of T to S_k is a connected subtree for all $k = 0, \ldots, m$;
- $|S_k| = c + 1$ for all $k = 0, \ldots, m-1$, and $|S_m| \leq c + 1$;
- $s \in S_0$ and $t \in S_m$;
- $|S_k \cap S_{k+1}| \geq 2$ for all $k = 0, \ldots, m-1$, i.e., two consecutive sets have at least two nodes in common.
- $S_k \cap S_{k+1} \cap J \neq \emptyset$ for all $k = 0, \ldots, m-1$, i.e., two consecutive sets have at least one junction in common.

These properties guarantee that there are enough holes to clear the path and move the pebble from one parking position to the next one.

4.0.1 Construction of caterpillar sets

Along path π_{rt} we select the node triple (i_k, j_k, ℓ_k): i_k and j_k represent the ends of a caterpillar set, while ℓ_k is a parking position. We proceed as follows:

1. Let $\ell_0 = r$, i_0 be the neighbor of r that belongs to π_{rt}, and:
   - if $d(i_0, t) \leq c - 1$, set $j_0 = t$, $m = 0$ and stop;
   - otherwise, let j_0 be the node on π_{rt} such that $d(i_0, j_0) = c - 2$. Set $k = 0$, $j_{-1} = i_0$ and go to Step 2.
2. let i_{k+1} be the closest junction to j_k on $\pi_{j_{k-1} j_k} \setminus \{j_{k-1}\}$, and ℓ_{k+1} be one of the neighbors of i_{k+1} not belonging to π_{rt}:
   - if $d(i_{k+1}, t) \leq c - 1$, set $j_{k+1} = t$, $m = k + 1$ and stop;
   - otherwise, let j_{k+1} be the node on $\pi_{i_{k+1} t}$ such that $d(i_{k+1}, j_{k+1}) = c - 2$. Set $k = k + 1$ and repeat Step 2.

Remark 4.1. *Note that i_{k+1} is always well-defined. Indeed, $d(i_k, j_k) = c - 2 \geq c_2$ guarantees that there exists a junction on $\pi_{i_k j_k} \setminus \{i_k\}$ ($i_{k+1} \in \pi_{i_k j_k} \setminus \{i_k\}$). Since, by definition, i_k is the closest junction to j_{k-1} on $\pi_{j_{k-2} j_{k-1}} \setminus \{j_{k-2}\}$, then $i_{k+1} \notin \pi_{i_k j_{k-1}}$. Therefore, there exists $i_{k+1} \in \pi_{j_{k-1} j_k} \setminus \{j_{k-1}\}$ which is the closest junction to j_k.*

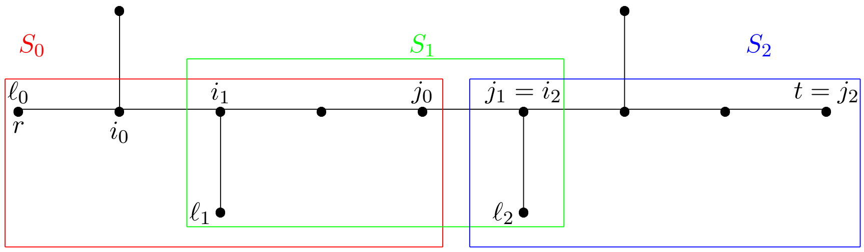


Figure 4: We consider the motion planning problem with source vertex r and target vertex t on a tree with $c = 5$. S_0, S_1 and S_2 are the *caterpillar sets* along path π_{rt}.

Observation 4.2. *Since $d(i_k, j_k) = c - 2$, it follows that*

$$d(i_k, i_{k+2}) = d(i_k, j_k) + d(j_k, i_{k+2}) \geq c - 1.$$

Now, let us find a lower bound for

$$d(i_0, i_m) = \sum_{k=0}^{m-1} d(i_k, i_{k+1}).$$

If m is even,

$$\sum_{k=0}^{m-1} d(i_k, i_{k+1}) = \sum_{k=0}^{\frac{m}{2}-1} [d(i_{2k}, i_{2k+1}) + d(i_{2k+1}, i_{2k+2})] =$$

$$= \sum_{k=0}^{\frac{m}{2}-1} d(i_{2k}, i_{2k+2}) \geq \frac{m}{2} \cdot (c-1).$$

Otherwise, if m is odd,

$$\sum_{k=0}^{m-1} d(i_k, i_{k+1}) = \left(\sum_{k=0}^{m-2} d(i_k, i_{k+1}) \right) + d(i_{m-1}, i_m)$$

$$\geq \frac{m-1}{2} \cdot (c-1).$$

Therefore,

$$m \leq 2 \cdot \frac{d(i_0, i_m)}{c-1} + 1 \leq 2 \cdot \frac{\delta}{c-1} + 1,$$

where δ is the diameter of the tree. If $c \geq 2$, it follows that $m = O(\frac{\delta}{c})$. Note that $c = 1$ only holds for the trivial case of the tree with 2 edges.

We can easily verify that with the construction just presented, all the properties of the caterpillar sets are verified:

- S_k is a connected component of T, and so it is a subtree;

- $|S_k| = |\pi_{i_k j_k}| + 2 = c + 1$ for all $k = 0, \ldots, m-1$ and $|S_m| = |\pi_{i_m j_m}| + 2 \leq c + 1$;
- $\{i_{k+1}, l_{k+1}\} \subset S_k \cap S_{k+1}$ and $i_{k+1} \in J$, so $|S_k \cap S_{k+1}| \geq 2$ and $S_k \cap S_{k+1} \cap J \neq \emptyset$ for $k = 0, \ldots, m-1$.

Moreover, it holds that

$$\begin{aligned} |S_{k+1} \cup S_k| = |S_{k+1}| + |S_k| - |S_{k+1} \cap S_k| \leq \\ \leq (2c+2) - 2 = 2c. \end{aligned}$$

We are now ready to describe the procedure for solving the motion planning problem in case A.

Procedure A

1. Gather holes in $S_0 \setminus \{\ell_0\}$: $O(nc)$ moves in view of Proposition 3.2.
2. Move pebble from ℓ_0 to ℓ_1: c moves.
3. For all $k = 0, .., m-1$ moves the holes from $S_k \setminus \{\ell_{k+1}\}$ to $S_{k+1} \setminus \{\ell_{k+1}\}$ and move pebble p from ℓ_{k+1} to t or ℓ_{k+2} (see Figure 5). The former operation consists in sliding the obstacles from $S_{k+1} \setminus S_k$ to $S_k \setminus S_{k+1}$: this is equivalent to *gather holes in* S_{k+1} in the subtree obtained by the restriction of T to $S_{k+1} \cup S_k$, and so it requires at most $(|S_{k+1} \cup S_k| \cdot c) = O(c^2)$ moves. The last operation requires at most c moves. Overall the `for` cycle has length complexity $O(mc^2)$.

Note that the operations just described in *Procedure A* are all feasible because the properties of caterpillar sets hold.

Then, we proved the following result.

Proposition 4.3. *The length complexity of the solution provided by Procedure A is* $O(nc)$*.*

Proof. The solution provided by Procedure A requires

$$O(nc) + O(mc^2)$$

moves. Equivalently, the length complexity of the solution is

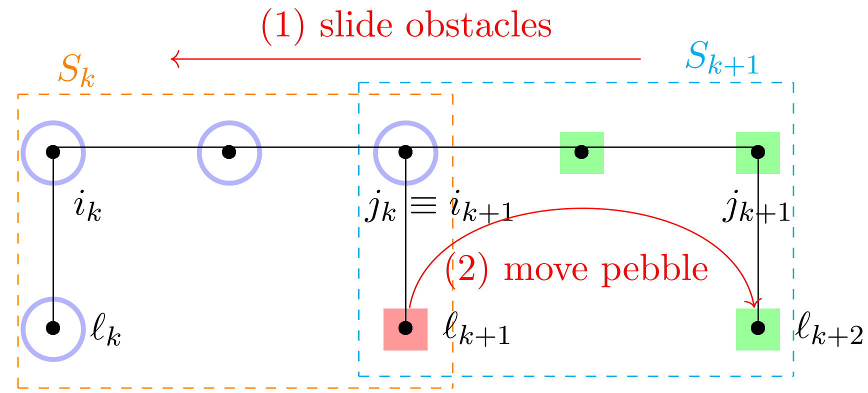
$$O(nc) + O(\delta c) = O(nc),$$

recalling that $m = O\left(\frac{\delta}{c}\right)$.

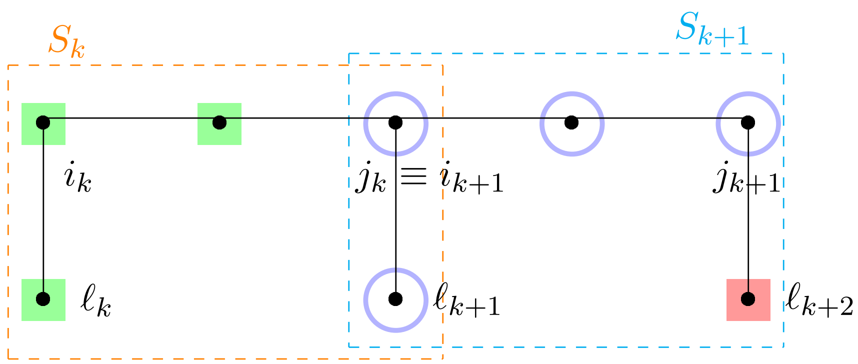
□

Case B. $T(F(s), t)$ contains less than c holes.

Suppose that q holes are missing to get to c. In this case, we create a neighborhood of s made up of q holes in $\mathcal{C}(F(s), t)$, and then we move the pebble p to a node v such that $T(F(v), t)$ contains at least c holes. Let $z_1, \ldots, z_k$ the neighbors of s which are not in $T(F(s), t)$. For all $j = 1, \ldots, k$, we consider $T_j := T(F(s), z_j)$, the subtree containing z_j, and V_j the corresponding set of nodes. See Figure 6, which illustrates this situation. In this case the motion planning problem is solved through the following procedure.



(a) Initial configuration. First we slide the obstacles from $S_{k+1} \setminus S_k$ to $S_k \setminus S_{k+1}$, then we move the pebble from ℓ_{k+1} to ℓ_{k+2}.



(b) Final configuration.

Figure 5: Example of execution of an iteration of the `for` cycle of Step 3 of *Procedure A*. Blue circles are holes, green squares are obstacles and the red square is the marked pebble.

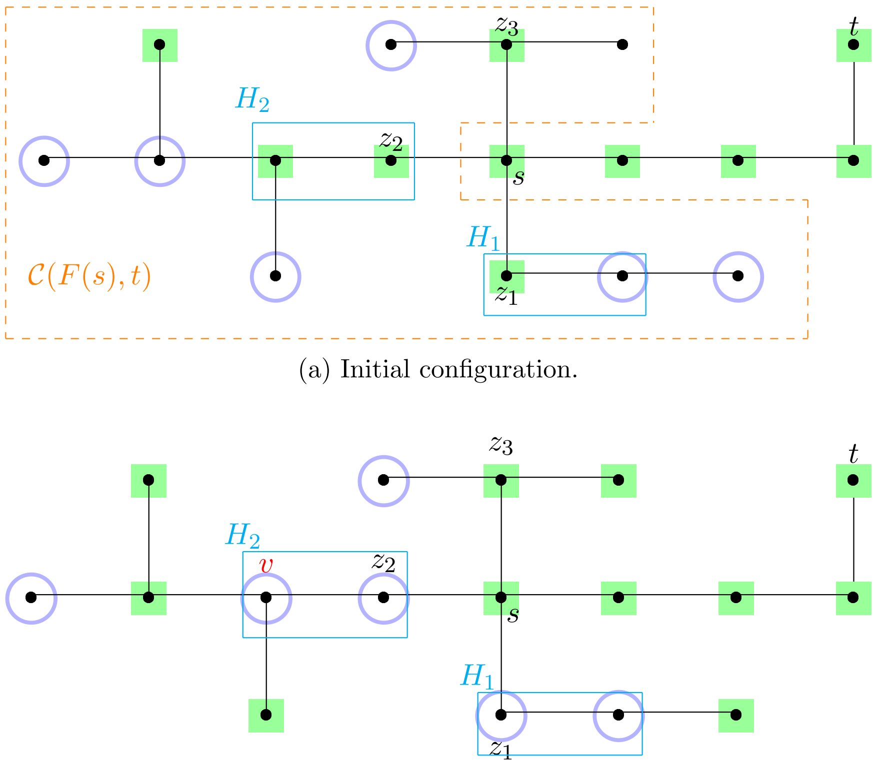
Procedure B

1. Set $j = 1$;
2. Let $q_j = |\mathcal{A}^s(H) \cap V_j|$ be the number of holes in T_j:
   - if $q_j \leq q$, gather all the holes of T_j in H_j, which is a subset of V_j of cardinality q_j closest to the subset $\{s\}$:
   $$H_j \in \arg \min_{\substack{W \in \mathcal{P}(V_j) \\ |W| = q_j}} d(W, \{s\});$$
   Set $j = j + 1$, $q = q - q_j$ and go back to Step 2;
   - otherwise, gather q holes of T_j in H_j which, in this case, is defined as a subset of V_j of cardinality q closest to s.

3. choose $v \in H_j$ which has the maximum distance from s and move pebble p on v;

4. reinitialize s with node v and apply *Procedure A*.

Note that Step 4 is feasible because in $T(F(v), t)$ there are certainly at least c holes.



(a) Initial configuration.

(b) Final configuration after *Procedure B*.

Figure 6: Example of situation of Case B with $q = 4$ missing holes. $H_1 \cup H_2$ is the neighborhood of s contained in $\mathcal{C}(F(s), t)$ where we group the q missing holes.

Let n_j be the number of nodes of T_j. Then, recalling the length complexity of the gather holes procedure, *Procedure B* (without the final application of *Procedure A*) requires at most $\left(\sum_{j=1}^{k} n_j q_j\right)$ moves to bring the holes in the neighborhood of s, and at most n moves to bring pebble p on v. Therefore, the length complexity of this procedure is $O(nq)$ which, in the worst case, is $O(nc)$. The final application of *Procedure A* does not modify the complexity result. Thus, we proved the following result.

Proposition 4.4. *The complexity of Procedure B is $O(nc)$.*

5. Pebble Motion on Trees

Now we are ready to provide a procedure for the solution of PMT. We are given a tree $T = (V, E)$, a pebble set P, an initial valid configuration $\mathcal{A}^s$, and a final valid configuration $\mathcal{A}^t$. As already mentioned, the PMT problem consists in generating a plan f such that $\mathcal{A}^t(p) = \rho(\mathcal{A}^s, f)(p)\ \forall\, p \in P$.

We present the main idea for finding plan f. First, we move the pebbles to an ordered set of *intermediate targets* $\bar{t}_1, \ldots, \bar{t}_{|P|}$. Then, we move them to the final targets. We focus on the first part of the plan, that is the motion to the intermediate targets. In this stage, we use a simple recursive strategy, based on the solution of a sequence of motion planning problems. Namely,

- We choose a pebble and move it to a target leaf, solving a motion planning problem.
- We remove the pebble and the intermediate node from the tree.
- We repeat the procedure with another pebble.

In this way, after a pebble has reached the intermediate target (that is always a leaf of the current tree), the pebble is no longer considered. Furthermore, we choose the intermediate targets so that, after each pebble has reached its intermediate target, it is sufficient to solve an unlabeled PMT to move all pebbles to the final targets.

In more detail, we will use the following strategy.

- First, we solve an unlabeled PMT on T that would bring the pebbles from the targets to the intermediate targets. In this way, we associate each intermediate target $\bar{t}_k$ to a pebble $p_{i_k} \in P$. Later, we will discuss the choice of the intermediate targets. We call g the corresponding plan. Note that *we do not apply this plan.* Instead, as the last part of the plan, we will apply the *inverse plan* g^{-1} to move the pebbles from the intermediate targets to the final targets.
- Then, we solve a set of $|P|$ motion planning problems on a nested sequence T_k of subtrees of T. The sequence is such that $T_1 = T$ and, for $k = 1, \ldots, |P| - 1$, T_{k+1} is obtained by removing from T_k the intermediate target $\bar{t}_k$ (that occupies a leaf of T_k). Over each subtree, we use the CATERPILLAR algorithm to move each pebble $p_{i_k} \in P$ from source $\mathcal{A}^s(p_{i_k})$ to intermediate target $\bar{t}_k$. After the execution of the corresponding plan, we remove target $\bar{t}_k$ from tree T_k, obtaining tree T_{k+1}, and remove p_{i_k} from the set of pebbles. This can be done because, as previously mentioned, when a pebble lands on a leaf, it no longer needs to be moved to let the other pebbles pass.
- Finally, we apply the inverse plan g^{-1} on T and each pebble p_{i_k} moves to the corresponding final target t_{i_k}.

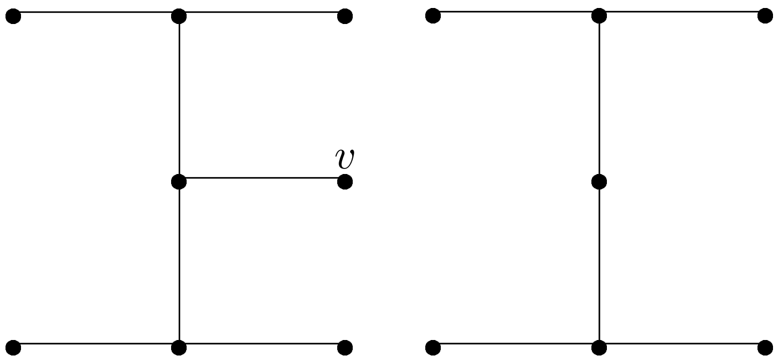
We choose the *intermediate targets* such that trees T_k satisfy the following properties, for $k = 1, \ldots, |P| - 1$:

- T_k contains $\bar{t}_k, \bar{t}_{k+1}, \ldots, \bar{t}_{|P|}$, but does not contain $\bar{t}_1, \ldots, \bar{t}_{k-1}$;
- $c(T_k) \geq c(T_{k+1})$.

Let $L(T_k)$ be the set of all the leaves of the tree T_k. We denote the set of intermediate targets by $V_1 = \{\bar{t}_k : k \in \{1, \ldots, |P|\}\}$. We define them by the following procedure:

1. Set $k = 1$ and $T_1 = T$.
2. If $k > |P|$, stop.
3. Select $v \in L(T_k)$ such that, given the tree T_k^v obtained by removing v from T_k, it holds that $c(T_k^v) \leq c(T_k)$.
4. Define $\bar{t}_k = v$ and $T_{k+1} = T_k^v$. Set $k = k + 1$ and to go Step 2.

In Step 3, v must be chosen appropriately, because in some cases it may happen that $c(T_k^v) > c(T_k)$ (see for example Figure 7).



(a) T_k: $c_1 = 1$, $c_2 = 1 \Rightarrow c(T_k) = 3$. (b) T_k^v: $c_1 = 2, c_2 = 2 \Rightarrow c(T_k^v) = 4$.

Figure 7: A case in which $c(T_k^v) > c(T_k)$.

However, the following proposition shows that Step 3 of the above procedure is well defined (i.e., it is always possible to find a leaf v of T_k such that $c(T_k^v) \leq c(T_k)$). Note that the result is valid for general trees and not only for the subtrees T_k generated by the above procedure.

Proposition 5.1. *For all $k \in \{1, \ldots, |P| - 1\}$, there exists $v \in L(T_k)$ such that $c(T_k^v) \leq c(T_k)$, where T_k^v is the tree obtained by removing v from T_k.*

Proof. For each $w \in L(T_k)$ we denote by n_w the unique neighbor of w, and by $deg(n_w)$ its degree. We define three subsets of leaves: $L_1(T_k) := \{w \in L(T_k) : deg(n_w) = 1\}$, $L_2(T_k) := \{w \in L(T_k) : deg(n_w) = 2\}$, $L_3(T_k) := \{w \in L(T_k) : deg(n_w) = 3\}$, $L_4(T_k) = \{w \in L(T_k) : deg(n_w) \geq 4\}$. Note that $L_1(T_k) \cup L_2(T_k) \cup L_3(T_k) \cup L_4(T_k) = L(T_k)$. We choose v as follows (see also Figure 8):

- if $L_1(T_k) \neq \emptyset$, then T_k contains only the edge (w, n_w): in this case we choose $v = n_w$ and it follows that $c(T_k^v) = 0 < c(T_k) = 1$;
- else, if $L_4(T_k) \neq \emptyset$, we choose $v \in L_4(T_k)$: in this case $c(T_k^v) = c(T_k)$, since by removing v, we just remove a corridor of length 1 (see Figure 8a);

- else, if $L_2(T_k) \neq \emptyset$, we choose $v \in L_2(T_k)$: in this case $c_1(T_k^v) \leq c_1(T_k)$ and then $c(T_k^v) \leq c(T_k)$, since by removing v, we are reducing by one the length of one corridor in $C(T_k) \setminus \bar{C}(T_k)$ (see Figure 8b);

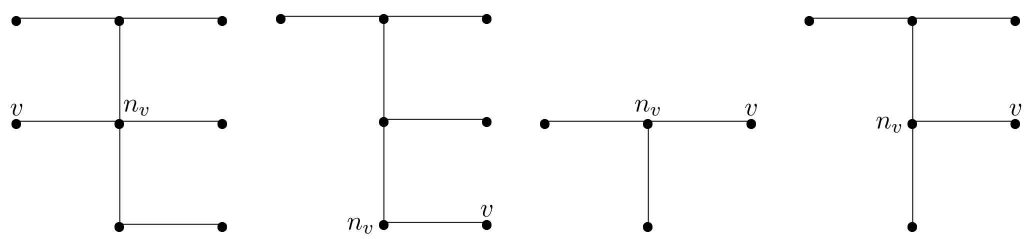
- otherwise, all corridors in $C(T_k) \setminus \bar{C}(T_k)$ have length one. We choose $v \in L_3(T_k)$. Two cases are possible:

  1. $\bar{C}(T_k) = \emptyset$, so that T_k is a star graph with a central node and three neighboring leaves. It holds that $c(T_k) = 2$. Removing one leaf v, T_k^v is a path graph of length 2, therefore $c(T_k^v) = 2 = c(T_k)$ (see Figure 8c).
  2. $\bar{C}(T_k) \neq \emptyset$: then, it holds that $c_1(T_k) = c_2(T_k)$ and, consequently, $c(T_k) = c_2(T_k) + 2$. Moreover, there exists at least one corridor in $\bar{C}(T_k)$ such that one of its endpoints is a junction connected to exactly two leaves, since $L_2(T_k) = L_4(T_k) = \emptyset$. We choose v between one of the two leaves (see Figure 8d). In this case, $c_1(T_k^v) \leq c_2(T_k) + 1$ and $c_2(T_k^v) \leq c_2(T_k)$, therefore

$$c(T_k^v) = \max\{c_1(T_k^v) + 1, c_2(T_k^v) + 2\} \leq$$

$$\leq \max\{(c_2(T_k) + 1) + 1, c_2(T_k) + 2\} = c(T_k).$$

$\square$



(a) Case $v \in L_4(T_k)$. (b) Case $v \in L_2(T_k)$. (c) Case $v \in L_3(T_k)$, $\bar{C}(T_k) = \emptyset$. (d) Case $v \in L_3(T_k)$, $\bar{C}(T_k) \neq \emptyset$.

Figure 8: The four cases of Proposition 5.1.

We propose the following *Leaves procedure for PMT*, which breaks down the PMT problem into an unlabeled problem and a series of motion planning problems, and find plan f for any PMT instance.

Leaves procedure for PMT

1. Let V_1 be the set of intermediate targets found with the previous procedure. Find a plan g which solves the unlabeled PMT problem from the final configuration $\mathcal{A}^t$ to V_1, i.e., such that
$$V_1 = \rho(\mathcal{A}^t, g)(P),$$
and let $\mathcal{A}^{\bar{t}} := \rho(\mathcal{A}^t, g)$ be the *intermediate configuration.* Note that for all $k \in \{1, \ldots, |P|\}$, g would move a pebble from t_{i_k} to the intermediate target $\bar{t}_k$.

2. Set $k = 1$ and perform the following procedure:

(a) if $k > |P|$, stop;

(b) using CATERPILLAR algorithm, solve the motion planning for pebble p_{i_k} from $\mathcal{A}^s(p_{i_k})$ to $\bar{t}_k$, i.e., find a plan f_k, over the tree T_k obtained from T by removing nodes $\bar{t}_1, \dots, \bar{t}_{k-1}$, such that $\bar{t}_k = \rho(\mathcal{A}^s, f_k)(p_{i_k})$;

(c) update $\mathcal{A}^s = \rho(\mathcal{A}^s, f_k)$;

(d) set $k = k + 1$ and go back to Step a).

3. Apply g^{-1}, the inverse plan of g, which, for all $k \in \{1, \dots, |P|\}$, moves pebble p_{i_k} from $\bar{t}_k$ to t_{i_k}.

Therefore, plan f which solves a given PMT instance is

$$f = f_1 f_2 \cdots f_{|P|}\, g^{-1}. \tag{9}$$

The complexity of the proposed procedure is stated in the following theorem.

Theorem 5.2. *The length complexity of the proposed procedure for the solution of the PMT problem is* $O(|P|nc + n^2)$.

Proof. Note that f_k requires at most $O(nc)$ moves for all $k = 1, \cdots, |P|$. Indeed, each f_k is the solution of a motion planning problem, which, in view of Propositions 4.3 and 4.4, is solved in $O(nc)$ moves. Moreover, g^{-1} requires at most n^2 moves. Indeed, g is the solution of an unlabeled PMT, which requires $O(n^2)$ moves, as seen in Section 3. Since g^{-1} is obtained from g by reversing its moves, it has the same length (see Observation 2.1). Then, the total number of moves is

$$O(|P|nc) + O(n^2).$$

□

Figure 16 provides an example of application of the *Leaves procedure* for a PMT instance with three pebbles.

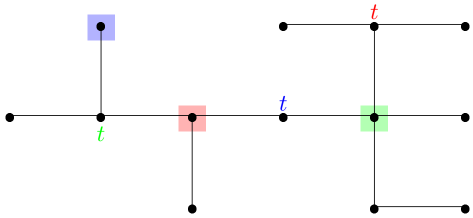
An interesting property of the *Leaves procedure for* PMT concerns the number of times each vertex is traversed by pebbles. As we further discuss in Section 6, the pebble motion problem on general graphs can be solved after converting it on a variant of PMT over trees. The bound on the number of times each vertex is crossed by the pebbles in the PMT problem over trees provided by the following proposition, allows deriving a complexity result also for the pebble motion problem over general graphs. Such complexity result will be the topic of a forthcoming paper.

Proposition 5.3. *In any solution provided by the proposed procedure, each vertex is crossed* $O(|P|c)$ *times by the pebbles.*

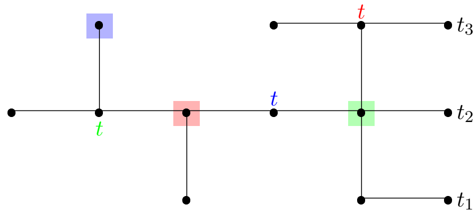
Proof. Let us count how many times each vertex is crossed in the solutions of each problem described in the paper:

1. *Basic plans.*

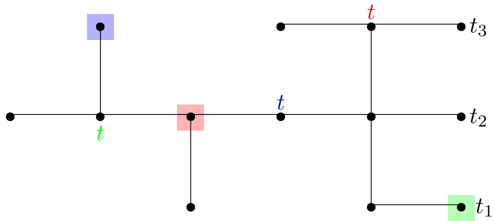
    - Bring hole from w to v (α_{vw}): each pebble along path π_{vw} moves forward one position, therefore each vertex is crossed at most once.



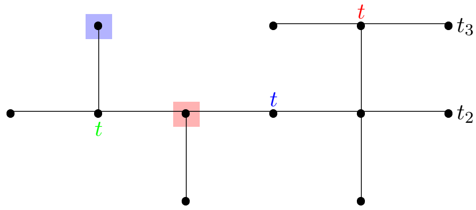
(a) Initial configuration.



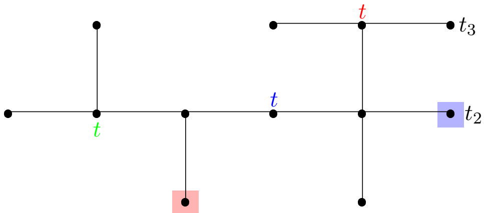
(b) Define intermediate targets t_1, t_2, t_3.



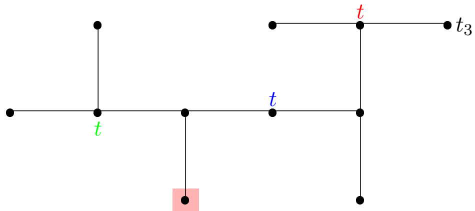
(c) Move green pebble to intermediate target.



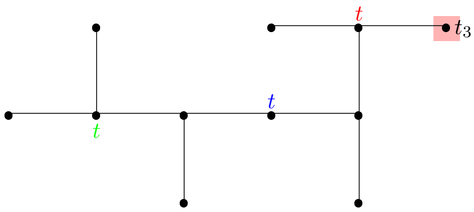
(d) Remove t_1.



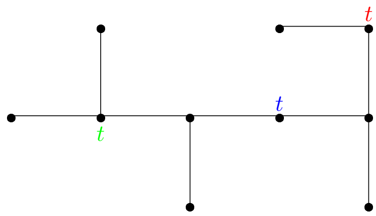
(e) Move blue pebble to intermediate target.



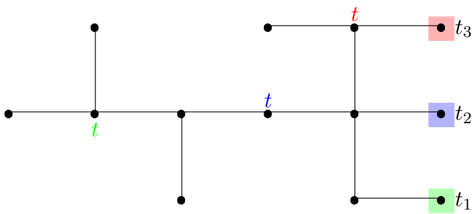
(f) Remove t_2.



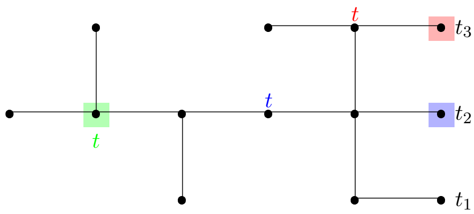
(g) Move red pebble to intermediate target.



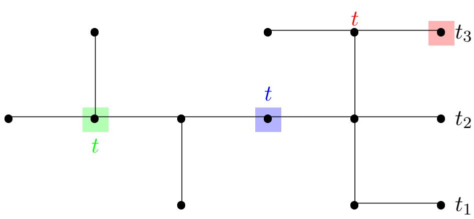
(h) Remove t_3



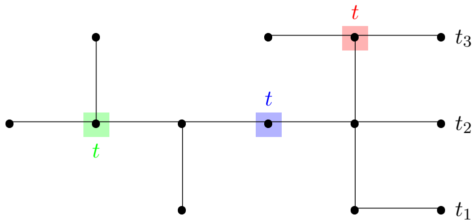
(i) Intermediate configuration.



(j) Move green pebble to final target.



(k) Move blue pebble to final target.



(l) Move red pebble to final target.

Figure 9: Example of application of the *Leaves procedure*. Green, red and blue squares represent the pebbles, while the other nodes host holes.

- Move Pebble from v to w (β_{vw}): one pebble moves on the path π_{vw}, therefore each vertex is crossed at most once.

2. *Unlabeled* PMT *problem.* In the solution of the *unlabeled* problem proposed in Section 3, each vertex is crossed at most once by each pebble for a total of $O(|P|)$ times in the overall procedure.

3. *Gather c holes.* In each iteration of Step 2 of the procedure described in Section 3.1, each vertex is crossed at most once because it performs α_{uv} or $\beta_{uw}\alpha_{wv}$. Since Step 2 is executed at most c times, each vertex is crossed $O(c)$ times.

4. CATERPILLAR algorithm. Each vertex is crossed $O(c)$ times, indeed:

   (a) *Procedure A*. In Step 1 we solve a *gather hole problem*, therefore by point (2) each vertex is crossed $O(c)$ times. In Step 3 each vertex that belongs to a caterpillar set S_k is traversed once time by the pebble and once time by at most c obstacles that arrived from S_{k+1} and then moved to S_{k-1}. Therefore, in *Procedure A* each vertex is crossed $O(c)$ times.

   (b) *Procedure B*. In Step 2 we solve a *gather hole problem* with q_j holes, therefore each vertex is crossed $O(q_j)$ times. Since $\sum_{j=1}^{k} q_j = q \leq c$, each vertex is crossed at most $O(c)$ times.

5. *Leaves Procedure.* A solution of PMT given by this procedure is $f = f_1 f_2 \cdots f_{|P|}\, g^{-1}$ (see (9)). For each $k \in \{1, \ldots, |P|\}$, f_k is the solution of a motion planning problem provided by the CATERPILLAR algorithm, therefore each vertex is crossed $O(c)$ times. Moreover, g^{-1} is the inverse of the solution of an unlabeled problem: therefore each vertex is crossed $O(|P|)$ times. We can conclude that in the solution plan of PMT each vertex is crossed

$$|P|O(c) + O(|P|) = O(|P|c)$$

   times.

□

6. PMT with Trans-shipment Vertices

The more general MAPF problem can always be reduced to a variant of the PMT (called *ts*-PMT) only under appropriate assumptions. In particular, it is possible if the following conditions hold:

- the graph is not a cycle;
- there are at least two holes;
- the graph is undirected or strongly connected directed (i.e., for each couple of nodes there exists a path that connects them).

These conditions are discussed in (Goraly & Hassin, 2010; Krontiris et al., 2013) for the case of undirected graphs and in (Ardizzoni et al., 2022; Botea & Surynek, 2015) for the case of directed graphs. Given a graph G, we can convert it into a tree (called *biconnected component tree*), adding a new type of vertex called *trans-shipment* (Ardizzoni et al., 2022; Goraly & Hassin, 2010; Krontiris et al., 2013). In particular, each biconnected component of the graph is converted into a star subgraph, whose internal vertex is a trans-shipment (see Figure 10).

(a) Biconnected component.

(b) Star subgraph with a trans-shipment internal vertex (diamond shape).

Figure 10: Conversion of a biconnected component of a graph into a star subgraph.

This way, the original problem on graph G is converted into a problem over a tree with trans-shipment vertices. Once a solution of the problem over the tree is obtained, this can be converted back into a solution for the original problem. In particular, each passage through a trans-shipment vertex on the star graph corresponds to a sequence of movements of the pebbles on the biconnected component. Figures 11 and 12 show an example of this conversion. For more details on the conversion algorithm see (Ardizzoni et al., 2022; Goraly & Hassin, 2010; Krontiris et al., 2013).

(a) Initial configuration.

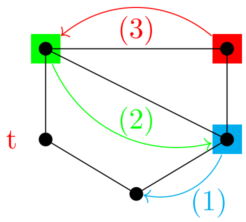
(b) Final configuration.

Figure 11: Example of plan that brings the red pebble to target t on the star graph.

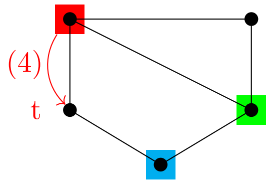
For this reason, we need to study a variant of the PMT problem, the *pebble motion on trees with trans-shipment vertices* (*ts*-PMT), which is a PMT problem on a tree such that the vertex set is partitioned in trans-shipment and regular vertices.

Definition 6.1. *A trans-shipment vertex is a vertex with degree greater than one that cannot host a pebble: pebbles can cross this node, but cannot stop there. More formally, given a trans-shipment vertex s,*

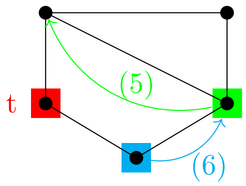
1. *deg*$(s) \geq 2$*;*
2. $\rho(\mathcal{A}, (u \to s)(w \to v))!$ *if and only if* $w = s$, $(u, s), (s, v) \in E$, *and* $\mathcal{A}^{-1}(v) \in H$. *If* $\rho(\mathcal{A}, (u \to s)(s \to v))!$, *then* $\rho(\mathcal{A}, (u \to s)(s \to v)) = \mathcal{A}[u, v]$.



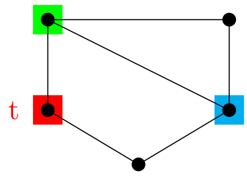
(a) Initial configuration, which corresponds to Figure 11a. First, we move pebbles in the order: blue, green and red.



(b) New configuration after moving pebbles as outlined in Figure 12a. Then, we will move red pebble to the target.



(c) New configuration after red pebble arrived at the target position. Then, we will move back green and blue pebbles to their initial positions.



(d) New configuration after moving back green and blue pebbles as explained in Figure 12c. This configuration corresponds to Figure 11b.

Figure 12: Conversion of the solution on the star graph of Figure 11 into a solution on the corresponding graph.

The second property means that, if a pebble is moved to a trans-shipment vertex, then it must be immediately moved to another node.

We denote by V_T the set of all the trans-shipment vertices and $V_R = V \setminus V_T$ the set of regular vertices. We require that V_T satisfies the following property

$$\forall\, v, w \in V_T \qquad d(v, w) > 1. \tag{10}$$

This assumption is motivated by the fact that trans-shipment vertices are the internal vertices of the stars, so they cannot be adjacent to each other.

Now we can formally define the PMT problem with trans-shipment vertices as follows.

Definition 6.2. ***(PMT problem with trans-shipment vertices)****. Let $T = (V, E)$ be a tree with $V = V_R \cup V_T$, where the set of trans-shipment vertices V_T is such that (10) holds. Given a pebble set P, initial and final valid configurations $\tilde{\mathcal{A}}^s$, $\tilde{\mathcal{A}}^t$ such that $\tilde{\mathcal{A}}^s(P), \tilde{\mathcal{A}}^t(P) \subset V_R$, find a plan f such that $\tilde{\mathcal{A}}^t = \rho(\tilde{\mathcal{A}}^s, f)$.*

This problem can be solved with the same procedure described in Section 5. However, some changes need to be made to ensure that the second property of Definition 6.1 is fulfilled. In the next subsections we show the changes we need to introduce into the previous procedures to address the presence of trans-shipment vertices.

6.1 Basic Plans

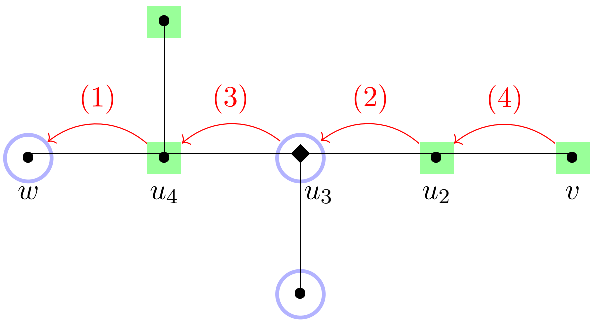
We generalize the definition of the plan BRING HOLE FROM w TO v to the case in which there is a trans-shipment vertex on the path π_{vw}.

For instance, if $\pi_{vw} = v\, u_2 \cdots u_i \cdots u_{n-1}\, w$ such that $u_i \in V_T$, then α_{vw} is defined as follows

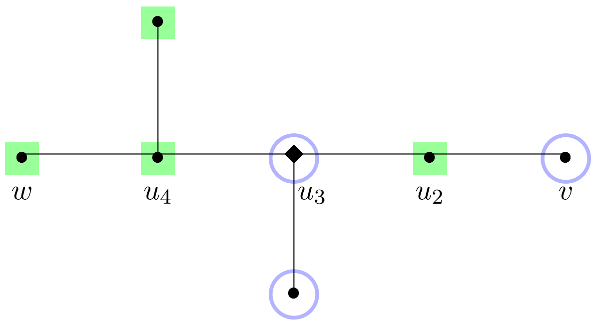
$$(u_{n-1} \to w, \ldots, u_{i-1} \to u_i, u_i \to u_{i+1}, \ldots, v \to u_2). \tag{11}$$

In other words, the only difference from the previous definition is that if a pebble move on u_i, then it immediately moves to u_{i+1}. For instance, see the example of Figure 13, where node u_3 is a trans-shipment vertex.

Observation 6.3. *Note that α_{vw} can be defined only if $w \in \mathcal{A}(H) \cap V_R$, which means that it is not allowed to bring hole from a trans-shipment vertex. Indeed, this could imply that in the final configuration a pebble lands on w. For the same reason plan* MOVE PEBBLE FROM v TO w *(i.e., β_{vw}), which in this case does not change, can be defined only if $w \in \mathcal{A}(H) \cap V_R$.*



(a) Initial configuration. The red edges represent plan $\alpha_{vw} = (u_4 \to w, u_2 \to u_3, u_3 \to u_4, v \to u_2)$. Node u_3 (identified by the diamond shape) is a trans-shipment vertex



(b) Final configuration after bringing the hole from w to v.

Figure 13: Example of BRING HOLE FROM w TO v. Vertex u_3 is a trans-shipment. Green squares represent pebbles, blue circles represent holes.

6.2 Assumption

Observation 6.3 implies that the main difference of the new algorithm is that the holes on the trans-shipment vertices cannot be used in bring hole and gather hole operations, which are the basis for all the procedures that constitute the algorithm to solve PMT. For this reason, we define a new distance $\tilde{d}$ which does not take into account trans-shipment vertices. Given a path π_{uv}, $\tilde{d}(u,v)$ counts how many regular vertices belong to the path:

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

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

$$\tilde{c}_1 := \max\{\tilde{d}(a,b) : \pi_{ab} \in C(T)\},$$
$$\tilde{c}_2 := \max\{\tilde{d}(a,b) : \pi_{ab} \in \bar{C}(T)\}.$$

Moreover, we define $\tilde{c} := \tilde{c}_1$ in the case of a path graph, $\tilde{c} := \max\{\tilde{c}_1 + 1, \tilde{c}_2 + 2\}$ otherwise. We note that on a tree with $V_T = \emptyset$, it holds that $d(u,v) = \tilde{d}(u,v) - 1$ and $c = \tilde{c} - 1$. Thus, to ensure the feasibility of any ts-PMT instance, at least $\tilde{c} - 1$ holes on regular vertices and $|V_T|$ holes for all trans-shipment vertices are needed. Therefore, Assumption (2) becomes

$$|H| \geq |V_T| + \tilde{c} - 1. \tag{12}$$

6.3 Unlabeled PMT with Trans-shipment Vertices

To solve this problem, we use the same procedure described in Section 3 to solve the classical *Unlabeled* PMT. The only difference is in Step 2 in the case vertex v is a source but not a target ($v \in S \setminus D$). Here, we need an unoccupied vertex u in order to move each pebble on the path π_{vu} towards it with plan α_{vu}. In this case u must be a regular vertex, so that we need to replace (7) with:

$$u \in \arg \min_{v' \in V_R \setminus S} d(v', v).$$

6.4 Gather Holes Problem with Trans-shipment Vertices

We use the same procedure described in Section 3.1. However, the choice of set M defined in (6) needs to be replaced by:

$$M \in \arg \min_{W \subset \mathcal{A}^s(H) \cap V_R:\ |W| = q} d(W, \bar{V}),$$

to guarantee that the holes in M are at regular vertices.

6.5 Motion Planning Problem with Trans-shipment Vertices

We must take into account the fact that trans-shipment vertices cannot host the marked pebble or the obstacles. Therefore, to ensure that the obstacle moves are feasible, we cannot only consider the cardinality of caterpillar sets, but the number of regular vertices they contain. To ensure this, we have to introduce the following modifications in the construction of the caterpillar sets:

1. replace d and c with $\tilde{d}$ and $\tilde{c}$;

2. the request on the size of the caterpillar sets concerns only the regular nodes: $|S_k \cap V_R| = \tilde{c}$ for all $k = 0, \ldots, m-1$, and $|S_m \cap V_R| \leq \tilde{c}$;

3. parking positions ℓ_k cannot be trans-shipment vertices. At each step k, if the neighbors of i_k not belonging to π_{rt} are all trans-shipment vertices, then let ℓ_k be one of the 2-hop neighbors of i_k, which certainly exist in view of the first property of Definition 6.1 and are regular vertices because of assumption (10). Therefore, we can generalize the definition of *caterpillar sets* as follows:

$$S_k = \pi_{i_k j_k} \cup \pi_{i_k \ell_k} \cup \pi_{i_{k+1} \ell_{k+1}}, \quad \forall k = 0, \ldots, m-1,$$

$$S_m = \pi_{i_m j_m} \cup \pi_{i_m \ell_m}.$$

   For instance, see Figure 14.

To solve the motion planning problem, we use *Procedure A* and *Procedure B* with some small tweaks:

1. In *Procedure A*: when we slide the obstacles, we move them from $(S_{k+1} \setminus S_k) \cap V_R$ to $(S_k \setminus S_{k+1}) \cap V_R$. Indeed, we cannot bring holes from trans-shipment vertices.

2. In *Procedure B*: at each iteration we gather the holes that are on $V_j \cap V_R$ in H_j, which is a subset of $V_j \cap V_R$ of cardinality q_j closest to s:

$$H_j \in \arg \min_{\substack{W \in \mathcal{P}(V_j \cap V_R) \\ |W| = q_j}} d(W, \{s\}),$$

   where $q_j = |\mathcal{A}^s(H) \cap V_j \cap V_R|$.

7. Experimental Results

We performed two distinct set of experiments, one regarding only the motion planning algorithm, one for the whole PMT algorithm. The algorithms have been implemented in `Matlab`. They can be downloaded at `https://github.com/auroralab-unipr/PMT`.

7.1 Motion Planning

In the first set of experiments, we generated random trees with a number of nodes $|V|$ ranging from 20 to 200 by 20 using the *NetworkX* (Hagberg et al., 2008) graph generator for random trees (function `random_tree()`). The number of pebbles $|P|$ ranges from 2 to $|V|-2$, while $\mathcal{A}^s$ and $\mathcal{A}^t$ are randomly generated. Only instances that fulfill Assumption (2) are taken into account. For every combination of number of nodes and number of pebbles, we generated 100 instances, each instance refers to a different graph. In Figure 15 we display the average number of moves of the solutions found on *nc*.

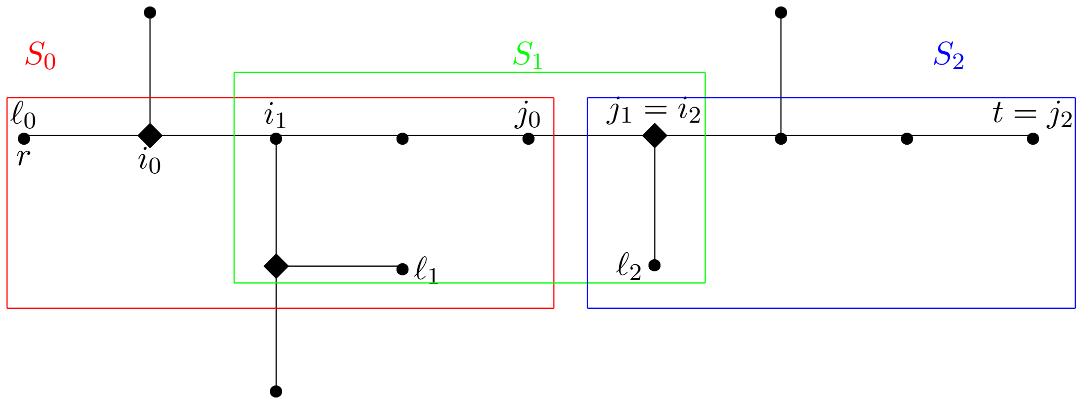


Figure 14: We consider the motion planning problem with source vertex r and target vertex t on a tree with $\tilde{c} = 5$. Diamond shapes represent trans-shipment vertices. S_0, S_1 and S_2 are the *caterpillar sets* along path π_{rt}.

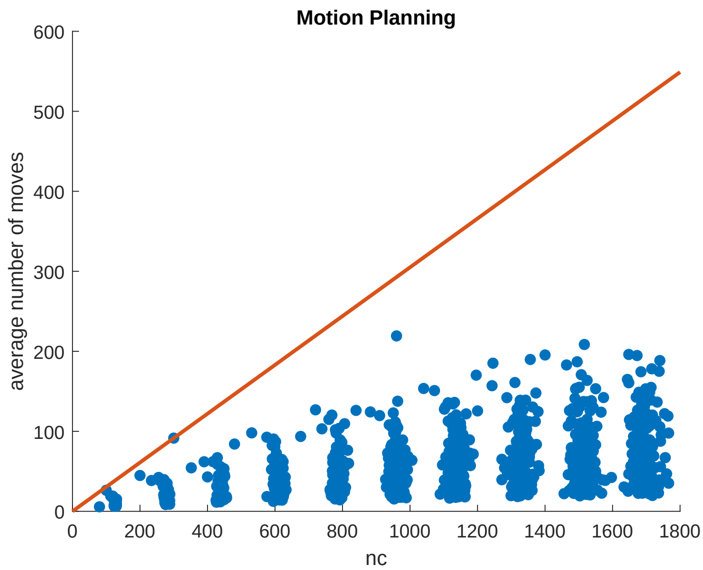


Figure 15: Average number of moves for the algorithm motion planning on nc.

According to Propositions 4.3 and 4.4, the motion planning algorithm returns solutions with length complexity $O(nc)$. In Figure 15 we can see a linear upper bound for the average number of moves, that supports the complexity result. We can also see how the number of moves is often much lower than the upper bound found. We remark that instances with number of moves closer to the upper bound line are those for which the number of pebbles is very high (as expected, these are more tricky instances).

7.2 PMT

In the second set of experiments, we generated random trees with a number of nodes n ranging from 20 to 200 by 20 using the same procedure used for the first set of experiments. The number of pebbles $|P|$ ranges from 5 to $(3/4)n$ by 5, while $\mathcal{A}^s$ and $\mathcal{A}^t$

are randomly generated. As for the first set of experiments, only instances that fulfill Assumption (2) are taken into account. For every combination of number of nodes and number of pebbles, we generated 20 instances of PMT problem. In Figure 16 we display the average number of moves of the solutions on $n|P|c + n^2$, found with the *Leaves Procedure*.

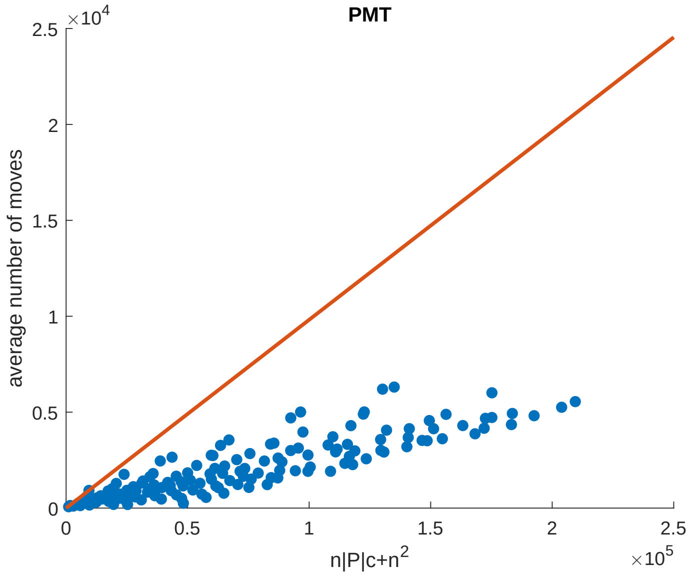


Figure 16: Average number of moves for the algorithm for PMT on $n|P|c + n^2$.

As stated in Theorem 5.2, the length complexity of the*Leaves Procedure* is $O(n|P|c+n^2)$.

Figure 16 displays a linear upper bound on the average number of moves, therefore confirming the complexity result.

Finally, in Figure 17 we show how the numbers of moves varies with the density $\frac{|P|}{n}$ of the tree. To do that, we generated random trees with a number of nodes ranging from 20 to 100, with unitary increments, and the number of pebbles $|P|$ ranges from 2 to $n/2$, with unitary increments. Note that we used unitary increments to consider a large number of density values. Figure 17 suggests that the number of moves increases polynomially with respect to the pebble density. Indeed, for a fixed number of nodes, if the density is higher, there are less free holes. Hence, some PMT instances might require more moves. Moreover, for a fixed density, the number of moves varies in a large range. This is due to the following reasons:

- Generally, for a fixed density, the plan length increases with the number of pebbles. Instances with a larger number of pebbles may require plans with a larger number of moves.
- In same cases, even if the density is high, the initial configuration may be close to the final one, so that the solution contains few moves.

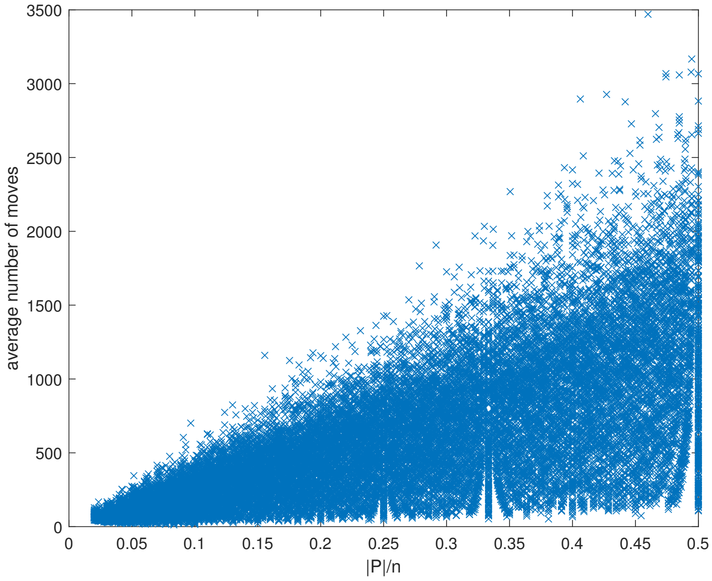


Figure 17: Number of moves for the algorithm for PMT as the graph density varies.

8. Conclusion

In this paper we proposed two algorithms with improved length complexity for the motion planning problem and the pebble motion problem on trees. Denoting by n the number of nodes, c the maximum length of corridors and k the number of pebbles, the CATERPILLAR algorithm solves the motion planning problem with $O(nc)$ moves, while the *Leaves procedure* solves the PMT problem in $O(knc + n^2)$ moves.

Moreover, we discuss a variant of the PMT problem, the PMT with trans-shipment vertices (ts-PMT), which considers a new type of vertex that cannot host pebbles. This problem is very interesting since MAPF instances on graphs can be reduced to it, and we proved that it can be solved with the *Leaves procedure for* PMT with some minor modifications.

As a topic for future research, we will study pebble motion, also known as Multi Agent Path Finding (MAPF), on general graphs. As already mentioned (see Section 6), the solution of MAPF on a general graph can be obtained by first converting it on the trans-shipment variant of PMT, and then converting back the obtained solution over the general graph. An upper bound for the solution length can be derived by exploiting the complexity results of this paper.

Appendix A. List of Main Symbols

This is the list of the main symbols employed throughout the paper.

$T = (V, E)$: tree with set of vertices V and set of edges E;

P: set of pebbles;

H: set of holes;

$\mathcal{A}$: a configuration, i.e., the position of pebbles and holes over V;

$\mathcal{A}(q)$: vertex occupied by $q \in P \cup H$;

$\mathcal{A}[u, v]$: the configuration obtained by $\mathcal{A}$ exchanging pebbles or holes on u and v;

$\tilde{\mathcal{A}}$: the equivalence class containing all the configurations that are in the same positions of $\mathcal{A}$, but without constraints on the positions of the holes;

$\mathcal{C}$: set of all the valid configurations;

f: generic plan, i.e., a sequence of moves;

E^*: set of all possible plans;

$\rho(\mathcal{A}, f)$: the configuration obtained by applying plan f to the initial configuration $\mathcal{A}$;

J: set of junctions, i.e., nodes in T with degree greater than two;

$C(T)$: set of all the corridors in tree T;

$\bar{C}(T)$: set of corridors with two junctions as endpoints;

$c_1(T)$, $c_2(T)$: maximum length of the corridors in $C(T)$ and $\bar{C}(T)$, respectively;

$c(T)$: maximum between $c_1(T) + 1$ and $c_2(T) + 2$;

π_{uv}: set of vertices of the unique path in T from u to v;

$d(u, v)$: length of the unique path in T from u to v.

References

Achá, R. A., López, R., Hagedorn, S., & Baier, J. A. (2022). Multi-agent path finding: A new boolean encoding. *Journal of Artificial Intelligence Research*, *75*, 323–350.

Alotaibi, E. T. S., & Al-Rawi, H. (2016). Push and spin: A complete multi-robot path planning algorithm. *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 1–8.

Ardizzoni, S., Saccani, I., Consolini, L., & Locatelli, M. (2022). Multi-agent path finding on strongly connected digraphs. *2022 IEEE 61st Conference on Decision and Control (CDC)*, 7194–7199.

Auletta, V., Monti, A., Parente, M., & Persiano, P. (1999). A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, *23*(3), 223–245.

Auletta, V., & Persiano, P. (2001). Optimal pebble motion on a tree. *Information and Computation*, *165*(1), 42–68.

Botea, A., Bonusi, D., & Surynek, P. (2018). Solving multi-agent path finding on strongly biconnected digraphs. *Journal of Artificial Intelligence Research*, *62*, 273–314.

Botea, A., & Surynek, P. (2015). Multi-agent path finding on strongly biconnected digraphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *29*(1).

Călinescu, G., Dumitrescu, A., & Pach, J. (2008). Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, *22*(1), 124–138.

De Wilde, B., Ter Mors, A. W., & Witteveen, C. (2014). Push and rotate: A complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, *51*, 443–492.

Dumitrescu, A. (2013). Mover problems. *Thirty Essays on Geometric Graph Theory*, 185–211.

Goraly, G., & Hassin, R. (2010). Multi-color pebble motion on graphs. *Algorithmica*, *58*(3), 610–636.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15).

Khorshid, M. M., Holte, R. C., & Sturtevant, N. R. (2011). A polynomial-time algorithm for non-optimal multi-agent pathfinding. *Fourth Annual Symposium on Combinatorial Search*.

Kornhauser, D. M., Miller, G., & Spirakis, P. (1984). *Coordinating pebble motion on graphs, the diameter of permutation groups, and applications* [PhD thesis, M. I. T., Dept. of Electrical Engineering and Computer Science]. M. I. T., Dept. of Electrical Engineering and Computer Science.

Krontiris, A., Luna, R., & Bekris, K. (2013). From feasibility tests to path planners for multi-agent pathfinding. *International Symposium on Combinatorial Search*, *4*(1).

Ma, H., & Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In C. M. Jonker, S. Marsella, J. Thangarajah, & K. Tuyls (Eds.), *Proceedings of the 2016 international conference on autonomous agents & multiagent systems, singapore, may 9-13, 2016* (pp. 1144–1152). ACM. http://dl.acm.org/citation.cfm?id=2937092

Nebel, B. (2020). On the computational complexity of multi-agent pathfinding on directed graphs. *Proceedings of the International Conference on Automated Planning and Scheduling*, *30*, 212–216.

Papadimitriou, C. H., Raghavan, P., Sudan, M., & Tamaki, H. (1994). Motion planning on a graph. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 511–520.

Röger, G., & Helmert, M. (2012). Non-optimal multi-agent pathfinding is solved (since 1984). *Proceedings of the International Symposium on Combinatorial Search*, *3*(1), 173–174.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, *219*, 40–66.

Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, T., et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. *Proceedings of the International Symposium on Combinatorial Search*, *10*(1), 151–158.

Wu, Z., & Grumbach, S. (2009). Feasibility of motion planning on directed graphs. *Theory and Applications of Models of Computation: 6th Annual Conference, TAMC 2009, Changsha, China, May 18-22, 2009. Proceedings 6*, 430–439.

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

Yu, J., & LaValle, S. M. (2013). Structure and intractability of optimal multi-robot path planning on graphs. *Twenty-Seventh AAAI Conference on Artificial Intelligence*.