Discrete Mathematics, Algorithms and Applications

On Reachability in Graphs with Obstacles

Udit Agarwal*
Department of Computer Science, University of Texas at Austin, USA
udit@cs.utexas.edu

Kalpesh Kapoor
Department of Mathematics, Indian Institute of Technology Guwahati, India
kalpesh@iitg.ernet.in

Received Day Month Year
Revised Day Month Year
Accepted Day Month Year
Published Day Month Year

Given a graph, G, a configuration of G represents that there is a robot at a vertex and obstacles at some other vertices. The remaining vertices of G not having the robot or an obstacle are said to be empty. The robot or an obstacle can be moved from its place to an adjacent vertex if it is empty. We give two definitions of reachability and show that they are equivalent in case of undirected graphs. We give an $\mathcal{O}(n+m)$ time algorithm for computing the minimum value of k for (v,k)-reachability in undirected graphs. However, we show that the two notions of reachability are different in case of directed graphs. We present a linear time algorithm for computing the minimum value of k for (v,k)-reachability in directed trees.

Keywords: Reachability in graphs; Obstacles on vertices; Graph configuration; Graph algorithms; Games involving graphs

Mathematics Subject Classification: 68R10, 05C85, 91A43

1. Introduction

Consider the following scenario: Given a directed/undirected graph, G, with a robot placed at one vertex and obstacles at some of the other vertices. A vertex at which the robot or an obstacle is not placed is said to have a hole or equivalently empty. Assume that we are allowed to move an obstacle or the robot to an adjacent empty vertex in G. Thus as the robot (or an obstacle) moves to the next node, the hole moves in direction opposite to that of the motion of robot (or obstacle). The problem

*Part of this work was done while the author was a student at Indian Institute of Technology Guwahati.

of moving the robot from some source vertex to a fixed destination node in a given graph using a set of moves of robot or obstacles is referred to as reachability problem.

In this paper we investigate the problem of finding an optimal initial configuration of obstacles so that the robot placed at the source vertex can move to every other reachable vertex (from the source vertex in absence of any obstacle) in a graph. We define the notion of (complete) (v, k)-reachability where k is the number of holes in an optimal initial configuration. We show that in case of undirected and strongly connected directed graphs, all the configurations of a graph with the same number of obstacles are reachable from each other. We propose linear time algorithms in case of undirected and strongly connected directed graphs for computing k that ensures (v, k)-reachability in a graph. We also give a linear time algorithm for directed trees.

The rest of the paper is organized as follows. In next section we give an overview of the prominent work done in this area. In Section 3, we define two types of reachability and prove some important properties. In Sections 4 and 5, we give algorithms to compute the minimum number of holes for (v, k)-reachability in strongly connected directed and undirected graphs, and directed trees, respectively. Finally, the conclusions are presented in Section 6.

2. Related Work

Motion planning is a fundamental problem in many areas such as robotics, track transportation system, communication networks and computational biology. Researchers have studied combinatorial aspects of motion planning by abstracting away the geometric considerations of the general motion planning problem.

Kornhauser, Miller and Spirakis [14] studied the reachability problem in undirected graphs to determine the feasibility of obtaining one pebble configuration from another. They showed that if a pair of legal pebble configurations are reachable from one another, then it can be done in $\mathcal{O}(|V|^3)$ moves, where $|V|$ is number of vertices in a graph. Oded Goldreich [9] further showed that the problem of finding the shortest sequence of moves is NP-Hard.

Auletta and Persiano [3] introduced the concept of configuration reachability (that is, reachability of one configuration from another) in trees. It has been shown that the feasibility can be decided in linear time for the configuration reachability in trees. Similar configuration reachability problems are studied in the context of graphs [15,10,11,7,9,14,4,3,2,8], pebbling game [13], Peg Solitaire game [12] and in protein folding [1]. Earlier work on the analysis of games that are based on graphs has also been focused on complexity issues such as for grid graph [17].

The problem of motion planning in graphs was introduced by Papadimitriou et al. [16] They defined the Graph Motion Planning with One Robot problem (GMP1R). The objective of GMP1R problem is to move the robot from a given source vertex s to a destination vertex t. The feasibility version of the problem is concerned with the reachability of robot from s to t. Whereas, the optimiza-

tion problem deals with minimizing the number of moves of taking the robot from s to t. It has been shown that the feasibility can be decided in polynomial time and the optimization problem, that is, finding the minimum number of moves, is NP-Complete.

The feasibility of the motion planning problem on directed graphs was solved by Wu and Grumbach [19,20]. They gave solution for the configuration reachability problem, specifically, whether the destination vertex is reachable from a source vertex in given a configuration of a robot and some obstacles. They also noticed that the irreversibility of moves in directed graphs make the problem much more difficult as compared to undirected graphs. They proposed linear time algorithms to solve the feasibility problem in acyclic and strongly connected directed graphs, and gave an $\mathcal{O}(|V|^2|E|)$ algorithm for solving the problem in general directed graphs, where $|V|$ and $|E|$ are the number of vertices and edges in a graph.

3. (v, k)-Reachability

Let G be a directed or undirected graph with a robot placed at one vertex, obstacles at some of the other vertices and holes at the remaining vertices. A hole is also referred to as empty. We denote the set of vertices and edges of G by $V(G)$ and $E(G)$, respectively. By a configuration, C^u_H, of graph G, we mean that the robot is placed at the vertex u and $H \subseteq V(G) - \{u\}$ is the set of vertices that have holes. The rest of the vertices $V(G) - (H \cup \{u\})$ have obstacles.

A movement of an obstacle from a vertex x to an adjacent empty vertex y is denoted by $x \xrightarrow{o} y$ or $y \xleftarrow{o} x$. Similarly, a movement of robot from a vertex x' to an adjacent empty vertex y' is denoted by $x' \xrightarrow{r} y'$ or $y' \xleftarrow{r} x'$. We use notation $C^u_H \rightarrow C^{u'}_{H'}$ to indicate that there exists a set of moves that transforms configuration C^u_H to $C^{u'}_{H'}$, where it is assumed that the number of holes remains the same, that is, $|H| = |H'|$. The absence of any such set of moves is denoted by $C^u_H \nrightarrow C^{u'}_{H'}$. When we are interested only in final position of the robot and do not care about the obstacles/holes, we use the notation $C^u_H \rightarrow C^{u'}_{\_}$.

Is it possible that $C^x_S \rightarrow C^y_{\_}$ is feasible only from certain set S? Does the size of S affects reachability? The motivation for Definitions 3.2 and 3.3 given next is to illustrate that both the number of holes and their placement in a graph affects reachability. This has also been observed elsewhere [20]. We define the set of vertices that are reachable from a vertex v in the absence of any obstacle as $R(v, G)$. Formally, for a graph G and a vertex $v \in V(G)$, $R(v, G) = \{x \mid C^v_{V(G)-\{v\}} \rightarrow C^x_{\_}\}$. When there is no confusion about graph G, we write $R(v, G)$ simply as $R(v)$. Since the trivial case when $R(v, G) = \{v\}$ is not interesting, we assume $|R(v, G)| \geq 2$ in the rest of the discussion.

Definition 3.1 (v-reachable). *Let G be a graph and v be a vertex in $V(G)$ and $S \subset V(G)$. Then, G is said to be v-reachable if for all $x \in R(v)$, $C^v_S \rightarrow C^x_{\_}$.*

Definition 3.2 ((v, k)-Reachability). *Let G be a graph and v be a vertex in $V(G)$. Then, G is said to be (v, k)-reachable if*

(a) there exists a set $S \subset V(G)$ such that $|S| = k$ and for all $x \in R(v)$, $C_S^v \to C_-^x$.
(b) there is no set $Q \subset V(G)$ with $|Q| < k$ such that for all $x \in R(v)$, $C_Q^v \to C_-^x$.

Definition 3.3 (Complete (v, k)-Reachability). *Let G be a graph and v be a vertex in $V(G)$. Then, G is said to be complete (v, k)-reachable if*

(a) for all set $S \subset R(v)$ such that $|S| = k$ and for all $x \in R(v)$, $C_S^v \to C_-^x$.
(b) there exists $Q \subset R(v)$ with $|Q| = k - 1$ and $x \in R(v)$ such that $C_Q^v \nrightarrow C_-^x$.

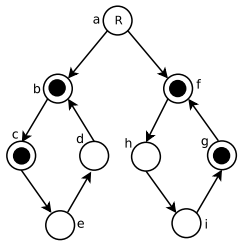

Fig. 1. A graph which is (a,4)-reachable but not complete (a,4)-reachable.

The premise in condition (a) of Definitions 3.2 and 3.3 ensures the reachability in absence of any obstacle. Note that complete (v, k)-reachability does not put any restriction on the initial placement of holes and obstacles in a given graph. The directed graph given in Figure 1 is $(a, 4)$-reachable with the placement of holes and obstacles as shown but it is not complete $(a, 4)$-reachable. This is because if obstacles are placed at nodes h and i instead of b and c, then it will not be possible to move the robot from vertex a to any of the vertex in the set $\{f, g, h, i\}$.

Note that, in general, the number of holes required for a subgraph of a graph G may be more than that required for G itself.

Proposition 3.4. *If G is a complete (v, k)-reachable graph then for all $i \in \{0, \ldots, k-1\}$ there exists $Q \subset R(v)$ with $|Q| = i$ and $x \in R(v)$ such that $C_Q^v \nrightarrow C_-^x$.*

Proof. On contrary, assume that there exists $j \in \{0, \ldots, k-1\}$ such that for all $Q \subset R(v)$ with $|Q| = j$ and for all $x \in R(v)$, we have $C_Q^v \to C_-^x$ and j is minimum among $\{0, \ldots, k-1\}$ to exhibit such property. Thus, for all $k - 1 \geq i \geq j$, we have, for all $Q \subset R(v)$ with $|Q| = i$ and for all $x \in R(v)$, satisfying $C_Q^v \to C_-^x$, which is a contradiction to Definition 3.3. □

Proposition 3.5. *If G is a (v, k)-reachable graph with an initial configuration C_H^v then $H \subset R(v)$.*

Proof. As G is (v, k)-reachable, hence for all $x \in R(v)$, $C_H^v \to C_-^x$. On contrary, assume that there exists at least one vertex in H that is not present in $R(v)$. Then

there exists $Q \subset H$ with $|Q| < k$ such that for all $x \in R(v)$, we have $C^v_Q \to C^x_-$. But according to Definition 3.2, this is a contradiction. Hence, H must be a subset of $R(v)$. □

Proposition 3.6. *Let G be an undirected (strongly connected directed) graph containing just holes and obstacles. If for some $x \in V(G)$, the vertex x contains a hole then this hole can be moved to all other vertices in G containing obstacle.*

Proof. Let v' be a vertex in G containing an obstacle. Let P be a path from v' to x in G. Thus the obstacle can be moved from v' to x along P and the hole will be moved to v'. As v' is arbitrary, hence the hole at x can be moved to all other vertices in G that contains obstacle. □

We will be using Proposition 3.6 in our discussion regarding the strongly connected directed graphs throughout this paper. The basic argument is that the holes can be moved anywhere in the graph.

Proposition 3.7. *In a given undirected graph or a strongly connected directed graph if $C^u_{H_u} \to C^v_{H_v}$ then $C^v_{H_v} \to C^u_{H_u}$.*

Proof. This follows from the observation that in an undirected graph we can reverse all the moves.

In case of strongly connected directed graphs also, every move can be reversed using the concept described in [20]. □

Proposition 3.8. *If G is an undirected (strongly connected directed) (v,k)-reachable graph then G is also (x,k)-reachable $\forall x \in R(v)$.*

Proof. From the definition of (v,k)-reachability, it is clear that there exists a set $S \subset V(G)$ such that $|S| = k$ and for all $x \in R(v)$,there exists H' such that $C^v_S \to C^x_{H'}$. From Proposition 3.7 for $C^v_S \to C^x_{H'}$, we have $C^x_{H'} \to C^v_S \to C^y_- \ \forall y \in R(v)$ and as $R(x) = R(v)$, $C^x_{H'} \to C^y_- \ \forall y \in R(x)$. Hence G is (x,k)-reachable. □

Proposition 3.9. *If G is an undirected (v,k)-reachable graph where v is not a cut vertex of G then $C^v_H \to C^v_{H'}$ $\forall H, H' \subset R(v)$ and $|H| = |H'| = k$.*

Proof. If v is not a cut vertex then either v is a pendent vertex or v is part of some cycle. And in both the cases, holes and obstacles can be moved without disturbing the robot placed at v. Therefore, $C^v_H \to C^v_{H'}$ $\forall H, H' \subset R(v)$ and $|H| = |H'| = k$. □

Proposition 3.10. *If G is a strongly connected directed (v,k)-reachable graph where v is not a cut vertex of G then $C^v_H \to C^v_{H'}$ $\forall H, H' \subset R(v)$ and $|H| = |H'| = k$.*

Proof. If v is not a cut vertex then either v is a part of a cycle of length 2 with both in-degree and out-degree equal to 1 or v is part of some cycle with at least 3 vertices.

In the former case, holes and obstacles can be moved without disturbing the robot placed at v. Therefore, $C^v_H \rightarrow C^v_{H'}$ $\forall H, H' \subset R(v)$ and $|H| = |H'| = k$.

Otherwise, let C be a cycle of length greater than 2 containing v. Then the holes and the obstacles can be moved by rotating the robot placed at v in C. (See Figure 2 for an illustration.) Therefore, $C^v_H \rightarrow C^v_{H'}$ $\forall H, H' \subset R(v)$ and $|H| = |H'| = k$. □

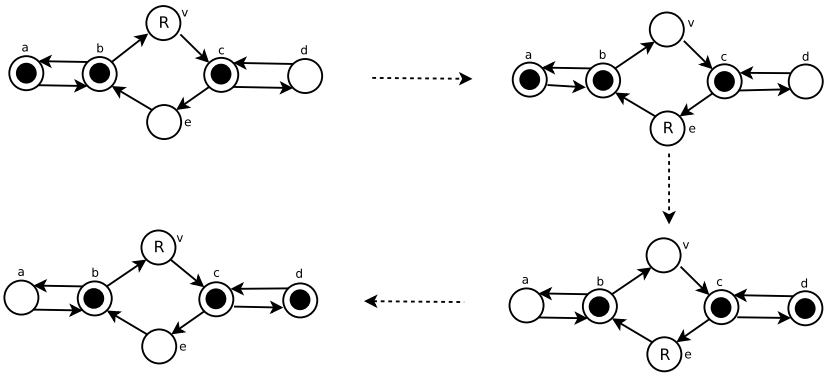

Fig. 2. Illustration for the proposition 3.10 when v is a part of a cycle with length greater than 2.

Lemma 3.11. *Let G be an undirected (strongly connected directed) (v,k)-reachable graph such that the vertex v is a cut-vertex. Further, let $G_1, \ldots, G_r$ be the components obtained after removing v from G. Consider the subgraphs G^v_i, for $1 \leq i \leq r$, with $V(G_i) \cup \{v\}$ and $E(G_i) \cup \{(x,v) \mid x \in V(G_i)$ and $(x,v) \in E(G)\} \cup \{(v,x) \mid x \in V(G_i)$ and $(v,x) \in E(G)\}$ as the set of vertices and edges, respectively. Then, every G^v_i, for $1 \leq i \leq r$, is complete (v,k')-reachable, for some $k' \leq k$.*

Proof. Consider the set $R(v, G^v_i)$ for some $i \in \{1, \ldots, r\}$. Since the graph G is (v,k) reachable, all the vertices in this set are also reachable in the presence of at most k holes.

Let x be an arbitrary vertex in $R(v, G^v_i)$. If there exists a set of moves in G that takes the robot from the vertex v to x without ever visiting a vertex in any other component $G_j, j \neq i$, and if it is true for all $x \in R(v, G^v_i)$ then the graph G^v_i is (v,k')-reachable for some $k' \leq k$. On the other hand, if this is not the case then there exists $y \in R(v, G^v_i)$ such that any set of moves that takes the robot from v to y visits some vertices that are not in $V(G^v_i)$. This situation arises when it is required to bring one or more holes from the other components to G_i through the vertex v. Since G is (v,k)-reachable, a maximum number of k holes can be moved to G_i, in which case too G^v_i is (v,k')-reachable for some $k' \leq k$.

Now as v is not a cut vertex of G^v_i, then from Proposition 3.9 and 3.10 it is always possible to have $C^v_{H'} \rightarrow C^v_H$ for any $H' \subset R(v, G^v_i)$ with $|H'| = |H| = k'$. Thus G^v_i is complete (v,k')-reachable for some $k' \leq k$ since $\forall H' \forall x\ C^v_{H'} \rightarrow C^v_H \rightarrow C^x_-$. □

Let G be a graph which is (v,k)-reachable, where $v \in V(G)$. Further, let S be a subgraph of G such that $v \in V(S)$ and S is (v,k') reachable. It is possible that the value k' may be greater than the value k. In other words, the number of holes required for a subgraph of a graph G may be more than that required for G itself. Thus, we cannot consider subgraphs independently in the proof of Lemma 3.11.

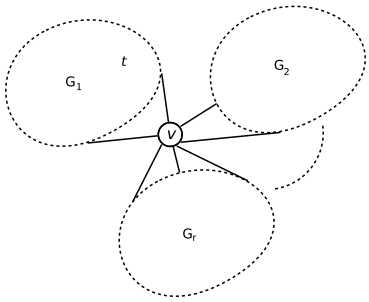

Fig. 3. Illustration for the proof of Lemma 3.12 when v is a cut vertex.

Lemma 3.12. *If G is an undirected (strongly connected directed) (v,k)-reachable graph then G is also complete (v,k)-reachable.*

Proof. Let G has some initial configuration C^v_H. By definition of (v,k)-reachability the number k is the minimal. From Proposition 3.5, $H \subset R(v)$. If v is not a cut vertex, then from Proposition 3.9 and 3.10 it is always possible to have $C^v_{H'} \rightarrow C^v_H$ for any $H' \subset R(v)$ with $|H'| = |H| = k$. Thus G is complete (v,k)-reachable since $\forall H' \forall x\ C^v_{H'} \rightarrow C^v_H \rightarrow C^x_{\_}$.

Consider the case when v is a cut vertex. Let $G_1, \ldots, G_r$, $r \geq 2$, be the components obtained after removing v from G. This structure is depicted in Figure 3. Without loss of generality, we first move the robot from v to one of the adjacent vertex (if possible). Assuming that the robot is currently placed at w, continue moving the robot forward in any of the $degree_w - 1$ directions provided that it does not move to any already visited vertex. Now the robot finally gets stuck at some vertex $u \in V(G)$. We again have two cases with respect to vertex u:

Case 1 If u is a non-cut vertex, then set $t = u$.

Case 2 If u is a cut vertex, then one of the component (say G_i) will contain all of the k holes. As G is (v,k)-reachable, thus by Proposition 3.8 G is also (u,k)-reachable. By Lemma 3.11, we can say that the subgraph G^u_i is complete (u,k')-reachable for some $k' \leq k$. Hence, we can move the robot from u to any non-cut vertex (say t) in G_i. This is always possible since every connected graph has at least one non-cut vertex.

We are given that G is (v,k)-reachable, thus it is possible to take the robot

to vertex t with configuration $C^t_{H_t}$ for some $H_t \subset R(v)$ with $|H_t| = k$, that is, $C^v_H \to C^t_{H_t}$. Starting with any arbitrary set of holes $H' \subset R(v)$ with $|H'| = k$, we can achieve H_t without affecting the robot at vertex t (by using Proposition 3.9 and 3.10 as t is a non-cut vertex), thus $C^v_{H'} \to C^u_{H''} \to C^t_{H'''} \to C^t_{H_t}$ for some $H'', H''' \subset R(v)$ with $|H''| = |H'''| = k$. Therefore, using Proposition 3.7 for $C^v_H \to C^t_{H_t}$, we have $\forall H' \forall x \exists u \exists H'' \exists t \exists H''' \exists H_t\ C^v_{H'} \to C^u_{H''} \to C^t_{H'''} \to C^t_{H_t} \to C^v_H \to C^x_-$. □

Theorem 3.13. *If G is an undirected (strongly connected directed) (v,k)-reachable for some vertex $v \in G$ then it is complete (x,k)-reachable, for all vertices $x \in R(v)$.*

Proof. Immediate from Proposition 3.8 and Lemma 3.12 . □

Theorem 3.14. *An undirected connected (strongly connected directed) graph G is complete (v,k)-reachable iff any two configurations C^w_H and $C^{w'}_{H'}$ are reachable from each other with $w, w' \in V(G)$ and $H, H' \subset V(G)$ and $|H| = |H'| \geq k$ and for all $j < k$ there exists a pair of configurations C^x_S and $C^y_{S'}$ with $x, y \in V(G)$ and $S, S' \subset V(G)$ and $|S| = |S'| = j$ such that $C^x_S \nrightarrow C^y_{S'}$.*

Proof. If G is complete (v,k)-reachable then, by Theorem 3.13, G is complete (x,k)-reachable $\forall x \in R(v)$. Let $w, w' \in V(G)$. We have $C^w_H \to C^t_{H_t}$ for some non cut-vertex t where $H, H_t \subset V(G)$ with $|H| = |H_t| \geq k$. We also have $C^{w'}_{H'} \to C^t_{H'_t} \to C^t_{H_t}$ for the same t with $H', H'_t \subset V(G)$ and $|H| = |H'| = |H_t| = |H'_t| \geq k$ using Proposition 3.9 and 3.10 as t is a non cut vertex. Thus, using Proposition 3.7, we can conclude that $C^w_H \to C^t_{H_t} \to C^t_{H'_t} \to C^{w'}_{H'}$. By the definition of complete (v,k)-reachability, for some $j < k$ there exists $H \subset V(G)$ with $|H| = j$ and at least one vertex x such that $C^v_H \nrightarrow C^x_-$.

Now for the other way round, assume that any two configurations C^w_H and $C^{w'}_{H'}$ are reachable from each other for $H, H' \subset V(G)$ and $|H| = |H'| \geq k$ where $w, w' \in V(G)$. Then $\forall H \subset V(G)$ with $|H| = k$ and $\forall x \in V(G)$, we have $C^v_H \to C^x_-$. Thus, G is complete (x,j)-reachable for some $j \leq k$. If $j < k$ then there exists $w, y \in V(G)$ and $H, H' \subset V(G)$ with $|H| = |H'| = j$ such that $C^w_H \nrightarrow C^y_{H'}$. As G is complete (v,j)-reachable, by Theorem 3.13, G is complete (w,j)-reachable as well as complete (y,j)-reachable. Let t be a non cut vertex. Thus, $C^w_H \to C^t_{H_t}$ and $C^y_{H'} \to C^t_{H'_t}$ for some $H_t, H'_t \subset V(G)$ with $|H| = |H_t| = |H'_t| = j$. Using Proposition 3.7 for $C^y_{H'} \to C^t_{H'_t}$ and Proposition 3.9 and 3.10 for t, we have $C^w_H \to C^t_{H_t} \to C^t_{H'_t} \to C^y_{H'}$ which is a contradiction. Therefore, $j = k$ and G is complete (v,k)-reachable. □

4. (v,k)-Reachability in Undirected and Strongly Connected Directed Graphs

We start our exploration with a special class of directed graphs known as strongly connected directed graphs. We first give an algorithm for computing the minimum number of holes for (v,k)-reachability for these graphs. Some of the notations and definitions that are given below is taken from [20].

Definition 4.1 (Underlying graph [20]). *Let $D = (V, E)$ be a directed graph. Then the underlying graph of D, denoted by $\mathcal{G}(D)$, is the graph obtained from D by omitting the directions of arcs, namely $\mathcal{G}(D) = (V, \{\{v, w\} | (v, w) \in E\})$.*

Definition 4.2 (Strongly biconnected graph [20]). *Let $D = (V, E)$ be a directed graph. D is said to be strongly biconnected if D is strongly connected and $\mathcal{G}(D)$ is biconnected.*

Definition 4.3 (Strongly biconnected-component graph [20]). *Let D be a strongly connected directed graph, the strongly-biconnected-component graph of D, denoted $\mathcal{G}_{sbc}(D) = (V_{sbc}, W_{sbc}, E_{sbc})$, is $\mathcal{G}_{bc}(\mathcal{G}(D))$, namely the biconnected-component graph of the underlying graph of D where*

- *V_{sbc} is strongly biconnected components of D;*
- *W_{sbc} vertices of D shared by at least two distinct strongly biconnected components of D;*
- *E_{sbc} let $B \in V_{sbc}$ and $w \in W_{sbc}$, then $(B, w) \in E_{sbc}$ iff $w \in V(B)$.*

Lemma 4.4 ([20]). *Let D be a strongly connected directed graph. Then $\mathcal{G}_{sbc}(D)$ is a tree.*

Definition 4.5 (w-side of a vertex [20]). *Let $D = (V, E)$ be a strongly connected directed graph and $\mathcal{G}_{sbc}(D) = (V_{sbc}, W_{sbc}, E_{sbc})$ be the strongly biconnected-component graph of D and u, v, $w \in V(\mathcal{G}_{sbc}(D))$, $v \neq w$. Then, u is said to be on the w-side of v, if $u \neq v$ and u, w are in the same connected component of $\mathcal{G}_{sbc}(D) - v$.*

Definition 4.6 (Chains [20]). *Let $D = (V, E)$ be a strongly connected directed graph, and $\mathcal{G}_{sbc}(D) = (V_{sbc}, W_{sbc}, E_{sbc})$ be the strongly-biconnected-component graph of D. Let $v \in V$, v is called a branching vertex if $v \in W_{sbc}$ and the degree of v is greater than 2 in $\mathcal{G}_{sbc}(D)$, that is, v is shared by at least three distinct strongly biconnected components. A chain of $\mathcal{G}_{sbc}(D)$ is a path $B_0 v_1 B_1, \ldots, B_{k-1} v_k B_k$ ($k \geq 1$) in $\mathcal{G}_{sbc}(D)$ such that*

(a) for all $1 \leq i \leq k-1$, $|V(B_i)| = 2$;
(b) for all $1 < i < k$, v_i is not a branching vertex.

Note that the length of a chain is the number of vertices in W_{sbc} on the chain.

A general algorithm MinNum(D, v, w) defined in [20] computes the minimal number of holes used to move the robot from v to w in a strongly connected directed graph D over all instances of the motion planning on D where v, w are the source and the destination vertex, respectively. The procedure MinNum(D, v, w) works as follows. If v is equal to w then it returns 0; if v and w belong to the same strongly biconnected component of D then it returns 1; Otherwise let $P = B_0 v_1 B_1 \ldots B_{r-1} v_r B_r$ ($r \geq 1$) be the path in $\mathcal{G}_{sbc}(D)$ such that $v \in B_0, w \in B_r, v \neq v_1$ and $w \neq v_r$, and l be the maximal length of the chains of $\mathcal{G}_{sbc}(D)$ such that they are contained in P. In this case return the value $l + 1$.

It can be observed from the above discussion that if l corresponds to the length of the longest chain in $\mathcal{G}_{sbc}(D)$, then at most $l+1$ holes are used to move the robot from s to t in a strongly connected directed graph D over all instances of the motion planning on D where s, t are the source and the destination vertex, respectively.

Proposition 4.7. *Let $D = (V, E)$ be a strongly connected directed graph, and $\mathcal{G}_{sbc}(D) = (V_{sbc}, W_{sbc}, E_{sbc})$ be the strongly-biconnected-component graph of D. Let l correspond to the length of the longest chain in $\mathcal{G}_{sbc}(D)$. If D is (v, k)-reachable, then $k = l + 1$.*

Proof. Assume $k \neq l + 1$. If $k \leq l$ then there exists at least one pair of vertices (s, t) such that there are not enough holes to move the robot from s to t. Then D is not (s, k)-reachable. But D is already (v, k)-reachable and by Proposition 3.8, D must be (s, k)-reachable as well. This results in a contradiction.

Otherwise, if $k > l + 1$ then there will be at least one hole that will not be used for moving the robot from one vertex to another in D as by Proposition 3.6, a hole can be moved to every other vertex in the graph. Thus D is (v, k')-reachable for $k' \leq k - 1$, which is a contradiction. Hence, $k = l + 1$. □

Given a strongly connected graph, G, and a vertex v on which the robot currently resides, the procedure minHolesSCC given as Algorithm 1 computes the minimum number of holes, k, required to ensure (v, k)-reachability in G. In line 2, it constructs the strongly biconnected-component graph using depth-first search technique [6]. All graphs are assumed to be represented using adjacency lists. The procedure then computes (in lines 3–28) the length of the longest chain ending at corresponding vertices (or the adjacent vertex) with the head lying on the r-side (here r is an arbitrary leaf vertex of $\mathcal{G}_{sbc}(G)$) of the corresponding vertex. In line 27, the variable k maintains the value of the longest chain in $\mathcal{G}_{sbc}(G)$.

The procedure randomly picks a leaf vertex in the graph and attaches several additional attributes to each vertex. The attribute $u.discovered$ stores the information whether u has been discovered or not. For $u \in W_{sbc}$, we store the length of the longest chain with u as it's penultimate vertex and the head lying on the r-side of u in the attribute $u.l$. For $u \in V_{sbc}$, the attribute $u.l$ stores the length of the longest chain ending at u with the head lying on the r-side of u. For $u \in V_{sbc}$, the attribute $u.isEndVertex$ holds the information whether the longest chain ending at u with the head lying on the r-side of u is maximal or not. The algorithm uses a queue, Q, to manage the set of discovered vertices stored in the order of their discovery.

The **while** loop of lines 11-28 iterates as long as there remain discovered vertices that have not yet had their adjacency lists fully examined. The **for** loop of lines 13-28 considers each vertex w in the adjacency list of u.

The line 17 checks if the strongly connected component u is of size greater than 2 as in that case u can not be an internal vertex of any chain in T. Hence, $w.l$ is updated to 1. The lines 23–26 are executed if $w \in V_{sbc}$ and $u \in W_{sbc}$. The **if**

Algorithm 1 For computing the minimum number of holes for a SCC graph.

```
1:  procedure MINHOLESSCC(G, v)                    ▷ SCC graph G and vertex v ∈ G
2:      Construct strongly biconnected component graph
                    T = G_sbc(G) = (V_sbc, W_sbc, E_sbc)
3:      k ← 0
4:      for all u ∈ T.V do
5:          (u.l, u.isEndVertex, u.discovered) ← (−1, false, false)
6:      Let r be one of the leaf vertices of T
7:      r.l ← 0
8:      Q ← φ
9:      Enqueue(Q, r)
10:     r.discovered ← true
11:     while Q ≠ φ do
12:         u ← Dequeue(Q)
13:         for all w ∈ T.Adj[u] do
14:             if w.discovered = true then continue
15:             w.discovered ← true
16:             if w ∈ W_sbc then
17:                 if |u| > 2 then
18:                     w.l ← 1
19:                 else if u.isEndVertex = true then
20:                     w.l ← 2
21:                 else
22:                     w.l ← u.l + 1
23:             else
24:                 w.l ← u.l
25:                 if |w| > 2 or u is a branching vertex then
26:                     w.isEndVertex ← true
27:             k ← MAXIMUM(k, w.l + 1)
28:             Enqueue(Q, w)
29:     return k
```

condition in line 25 checks if u is a branching vertex or the strongly connected component w is of size greater than 2.

Now we analyze the running time of the algorithm MINHOLESSCC. The construction of the strongly biconnected-component graph can be done by using depth-first-search technique [6] in $\mathcal{O}(|V|+|E|)$. The overhead for initialization in lines 4-5 is $\mathcal{O}(|T.V|)$. During execution, the test in line 14 ensures that the procedure enqueues a vertex at most once, and hence dequeued at most once. The operations of enqueuing and dequeuing takes $\mathcal{O}(1)$ time, and so the total time devoted to queue operations is $\mathcal{O}(|T.V|)$. As the procedure scans the adjacency list of each vertex only when the vertex is dequeued, it scans each adjacency list at most once.

Since the sum of the lengths of all the adjacency lists is $\mathcal{O}(|T.E|)$, the total time spent in scanning adjacency lists is $\mathcal{O}(|T.E|)$. The **for** loop in lines 13-28 takes $\mathcal{O}(1)$ for each iteration. Hence, the overall running time for the $\mathcal{G}_{sbc}(G)$ graph is $\mathcal{O}(|\mathcal{G}_{sbc}(G).V| + |\mathcal{G}_{sbc}(G).E|)$. Thus, the overall running time of the algorithm MIN-HOLESSCC is $\mathcal{O}(|V| + |E|)$.

Definition 4.8 (Longest Chains). *Let $G = (V, E)$ be a strongly connected directed graph with $\mathcal{G}_{sbc}(G) = (V_{sbc}, W_{sbc}, E_{sbc})$ as the strongly biconnected-component graph of G. Let w, $r \in \mathcal{G}_{sbc}(G)$ and r be a leaf vertex in $\mathcal{G}_{sbc}(G)$. Depending on whether $w \in W_{sbc}$ or V_{sbc}, $l(w, r)$ can be defined as follows:*

(a) If $w \in W_{sbc}$, then $l(w, r)$ represents the length of the longest chain with w as the penultimate vertex and it's head lying on the r-side of w.
(b) If $w \in V_{sbc}$, then $l(w, r)$ represents the length of the longest chain ending at w with it's head lying on the r-side of w

Clearly $l(w, r) \geq 1$ in general. Now, we investigate some important properties of longest chains.

Lemma 4.9. *Let $G = (V, E)$ be a strongly connected directed graph with $\mathcal{G}_{sbc}(G) = (V_{sbc}, W_{sbc}, E_{sbc})$ as the strongly biconnected-component graph of G. Let r be a leaf vertex in $\mathcal{G}_{sbc}(G)$. Let $B \in V_{sbc}$ and $w \in W_{sbc}$ such that $(w, B) \in E_{sbc}$ and w lies on the r-side of B. Then $l(B, r) = l(w, r)$.*

Proof. Assume $l(B, r) < l(w, r)$. Let $P \ldots wQ$ ($P, Q \in V_{sbc}$) be the longest chain with w as the penultimate vertex and it's head P lying on the r-side of w. As $(w, B) \in E_{sbc}$ and B lies on the non r-side of w. Therefore, $P \ldots wB$ is also a chain in $\mathcal{G}_{sbc}(G)$. Hence $l(B, r) \geq l(w, r)$, which is a contradiction.

Otherwise assume $l(B, r) > l(w, r)$. Let $P \ldots zB$ ($P \in V_{sbc}$, $z \in W_{sbc}$) be the longest chain ending at B with the head lying on the r-side of B. But z must be equal to w, otherwise it will result in a cycle in $\mathcal{G}_{sbc}(G)$. Therefore $l(w, r) \geq l(B, r)$, which is a contradiction. □

Lemma 4.10. *Let $G = (V, E)$ be a strongly connected directed graph with $\mathcal{G}_{sbc}(G) = (V_{sbc}, W_{sbc}, E_{sbc})$ as the strongly biconnected-component graph of G. Let r be a leaf vertex in $\mathcal{G}_{sbc}(G)$. Let $z, w \in W_{sbc}$ and $B \in V_{sbc}$ such that $(z, B), (B, w) \in E_{sbc}$ with z lying on the r-side of B and B lying on the r-side of w. Then*

(a) $l(w, r) = 1$ if $|B| > 2$.
(b) $l(w, r) = 2$ if $|B| = 2$ and z is a branching vertex.
(c) $l(w, r) = l(B, r) + 1$ if $|B| = 2$ and z is not a branching vertex.

Proof.

(a) By definition of l, $l(w, r) \geq 1$. Assume $l(w, r) > 1$. Let $P \ldots RwQ$ ($P, R, Q \in V_{sbc}$) be the longest chain in $\mathcal{G}_{sbc}(G)$ with w as the penultimate vertex and the

head P lying on the r-side of w. Clearly, $R \neq B$ as $|B| > 2$ and thus B cannot be an internal vertex of any chain. But as both R and B lies on the r-side of w, there must exist a cycle in $\mathcal{G}_{sbc}(G)$, which is a contradiction. Hence, $l(w, r) = 1$.

(b) By definition of l, $l(w, r) \geq 1$. Assume $l(w, r) = 1$. Let BwQ ($Q \in V_{sbc}$) be the longest chain in $\mathcal{G}_{sbc}(G)$ with w as the penultimate vertex and the head B lying on the r-side of w. Let $B_1 \in V_{sbc}$ such that $(B_1, z) \in E_{sbc}$ and B_1 lies on the r-side of z. Then B_1zBwQ is also a chain in $\mathcal{G}_{sbc}(G)$. Hence, $l(w, r) \geq 2$, which is a contradiction.

Let $l(w, r) = n$ for some $n > 2$ and $P_1x_1 \dots P_{n-1}x_{n-1}BwQ$ be the longest chain in $\mathcal{G}_{sbc}(G)$ with w as the penultimate vertex and the head P_1 lying on the r-side of w. Clearly, $x_{n-1} = z$ (otherwise there must exist a cycle in $\mathcal{G}_{sbc}(G)$). But z is a branching vertex and is contrary to point (b) mentioned in Definition 4.6. Hence, our assumption was wrong. Therefore, $l(w, r) = 2$.

(c) Assume $l(w, r) > l(B, r) + 1$. Let $P \dots xRwQ$ ($P, Q, R \in V_{sbc}$, $x \in W_{sbc}$) be the longest chain in $\mathcal{G}_{sbc}(G)$ with w as the penultimate vertex and the head P lying on the r-side of w. Here $R = B$ and $x = z$ (otherwise there exists a cycle in $\mathcal{G}_{sbc}(G)$ which is a contradiction). Then, $P \dots zB$ is also a chain of $\mathcal{G}_{sbc}(G)$, which implies that $l(B, r) \geq l(w, r) - 1$, that is, $l(w, r) \leq l(B, r) + 1$, which is a contradiction.

Assume $l(w, r) < l(B, r) + 1$. Let $P \dots zB$ ($P \in V_{sbc}$) be the longest chain of $\mathcal{G}_{sbc}(G)$ with the head P lying on the r-side of B. Then $P \dots zBwQ$ ($Q \in V_{sbc}$), where $(w, Q) \in E_{sbc}$ and Q lies on the non r-side of w, is also a chain in $\mathcal{G}_{sbc}(G)$. It implies $l(w, r) \geq l(B, r) + 1$, which is a contradiction. Hence, $l(w, r) = l(B, r) + 1$. □

Now, we will prove the correctness of the procedure minHolesSCC in Theorem 4.11.

Theorem 4.11. *Let $G = (V, E)$ be a strongly connected directed graph with $\mathcal{G}_{sbc}(G) = (V_{sbc}, W_{sbc}, E_{sbc})$ as the strongly biconnected-component graph of G, and suppose that* minHolesSCC *is run on $\mathcal{G}_{sbc}(G)$ with r (one of the leaf vertices of $\mathcal{G}_{sbc}(G)$) as the starting vertex. Then upon termination, for each vertex $u \in (V_{sbc} \cup W_{sbc}) - \{r\}$, the value $u.l$ computed by* minHolesSCC *satisfies $u.l = l(u, r)$.*

Proof. Assume for the sake of contradiction, that some vertex receives a l value not equal to it's corresponding longest chain length. Let w be a vertex in $\mathcal{G}_{sbc}(G)$ such that all the vertices on the path from r to w (excluding w) in $\mathcal{G}_{sbc}(G)$ have correct l values.

Case 1: **If** $w \in V_{sbc}$. Let $z \in W_{sbc}$ such that $(z, w) \in E_{sbc}$ and z lies on the r-side of w. After w is discovered during the scanning of the adjacency list of z, the value of $w.l$ is set to $z.l$ in line 24 and this value is not updated further till the termination of the algorithm. As $z.l = l(z, r)$, it implies that

$w.l = l(z, r)$. And by Lemma 4.9, $l(w, r) = l(z, r)$. Hence, $w.l = l(w, r)$, which is a contradiction.

Case 2: **If** $w \in W_{sbc}$. This case can also be proved on the similar grounds and by using Lemma 4.10, we can show that $w.l$ has correct value.

Hence no such vertex w exists in $(V_{sbc} \cup W_{sbc}) - \{r\}$ such that $w.l$ has incorrect value, that is, $u.l = l(u, r)\ \forall u \in (V_{sbc} \cup W_{sbc}) - \{r\}$. □

We can now prove that the procedure minHolesSCC correctly finds the value of k for ensuring (v, k)-reachability in a strongly connected directed graph by proving Theorem 4.12.

Theorem 4.12. *The Algorithm 1,* minHolesSCC*, computes the minimum number of holes required to ensure (v, k)-reachability in a given SCC graph G.*

Proof. Let $\mathcal{G}_{sbc}(G) = (V_{sbc}, W_{sbc}, E_{sbc})$ be the strongly biconnected-component graph of G. From Proposition 4.7, it is clear that for G to be (v, k)-reachable the value of k must be equal to $L + 1$ where L refers to the length of the longest chain in $\mathcal{G}_{sbc}(G)$. We claim that the Algorithm 1 indeed returns the same value of k.

On the contrary, assume that the algorithm returns a different value of k, that is, $k \neq L + 1$. There are two cases depending on whether k is less than or greater than $L + 1$.

Case 1: If $k < L + 1$ then $k \leq L$ and hence $\forall u \in \mathcal{G}_{sbc}(G) - \{r\}$ (where r is one of the leaf vertex of $\mathcal{G}_{sbc}(G)$ which was used as a root/source in the execution of procedure minHolesSCC) $u.l < L$.

Let this longest chain has endpoints P and Q where $P, Q \in V_{sbc}$. Then either P or Q lies on the r-side of Q or P respectively. Without loss of generality, assume P lies on the r-side of Q. By the correctness of Theorem 4.11, we have $Q.l = l(Q, r)$. But $Q.l < L$ and $l(Q, r) = L$, which is a contradiction.

Case 2: If $k > L + 1$ then there exists $w \in \mathcal{G}_{sbc}(G) - \{r\}$ such that $w.l > L$. By Theorem 4.11, we have $u.l = l(u, r) \forall u \in \mathcal{G}_{sbc}(G) - \{r\}$ and hence $l(w, r) = w.l > L$ where L represents the length of the longest chain in $\mathcal{G}_{sbc}(G)$, this results in a contradiction.

Hence, the value returned by the algorithm is indeed $L + 1$. □

Computing k for (v, k)-Reachability in Undirected Graphs

Consider an undirected graph $G = (V, E)$. To obtain the value of k for ensuring (v, k)-reachability in G, we first construct the corresponding strongly connected directed graph $G_D = (V, E_D)$ from G by replacing every edge $\{a, b\} \in E$ by two directed edges (a, b) and (b, a). Then we can apply the procedure minHolesSCC on G_D to obtain the value of k. As the number of edges in G_D is exactly double of those in G, the overall running time complexity remains the same which is $\mathcal{O}(|V| + |E|)$.

5. Computing k for (v, k)-Reachability in Directed Trees

Definition 5.1 (Edge Contraction [18]). *In a graph G, a **contraction** of edge e with endpoints u, v is the replacement of u and v with a single vertex whose incident edges are the edges other than e that were incident to u or v. The resulting graph $G.e$ has one less edge than G.*

Throughout this paper whenever we refer to edge contraction of an edge $e = (u, v)$, we will assume that the vertex v is replaced by u.

Definition 5.2 (Edge Contraction Graph). *Let G be a directed tree and S be the set of edges in G that need to be contracted. Then the edge contraction graph $ECG(G, S)$ is defined as the tree obtained after **contraction** of edges in set S.*

Definition 5.3 (Induced Subgraph [5]). *Let G be a graph with $V(G)$ as the vertex set. Suppose that V' is a nonempty subset of $V(G)$. Then, the subgraph of G whose vertex set is V' and whose edge set is the set of those edges of G that have both ends in V' is called the subgraph of G induced by V' and is denoted by $G[V']$.*

In this section, we will discuss the problem of computing k for (v, k)-reachability in directed trees.

The procedure minHolesTree follows a bottom-up approach to find an optimal initial configuration of obstacles with the robot placed at v. It outputs the minimum number of holes required to ensure (v, k)-reachability in a given directed tree.

The algorithm visits each vertex u in a bottom-up approach with level-order traversal such that all it's descendants have been discovered before it's own discovery. The algorithm then decides whether to keep u empty or not. This process continues until it discovers all the vertices that are reachable from the source vertex v.

Whenever the algorithm discovers a new vertex u, it updates the corresponding entries in arrays *available* and *longestPath* using the corresponding array entries of the vertices in the adjacency list of u. For computing the number of holes available to u, the algorithm adds up the number of holes available to each of the vertices in the adjacency list of u. Similarly for computing the length of the longest path from u, the algorithm adds one to the maximum of the longest path length from each of the vertices in the adjacency list of u.

After all the vertices are discovered, an optimal configuration is obtained. The algorithm then calculate the number of empty vertices in the tree, which is the value of k for (v, k)-reachability in the given tree. The pseudocode described in Algorithm 2 is the algorithm for computing k for (v, k)-reachability in directed trees.

The procedure minHolesTree works as follows. In line 2, depth-first search is performed on G to compute the set $R(v)$ and then an induced subgraph G' is obtained with vertex set $R(v)$. G' is then traversed using breadth-first search in line 6 and a unique id is given to each of the vertices in increasing order according to their discovery order in the breadth-first traversal such that the *id* assigned to v is 1.

Algorithm 2 For computing the minimum number of holes for directed tree.

1: **procedure** MINHOLESTREE(G, v) ▷ G is a directed tree
▷ $v \in V(G)$ and the robot is kept at v.
2: Use DFS on G to compute $R(v)$
3: **if** $|R(v)| = 1$ **then**
4: **return** 0 ▷ $R(v)$ contains just the vertex v
5: $G' \leftarrow G[R(v)]$
6: Do a breadth-first traversal of G'
7: Create a globally accessible hash map id to assign unique id's to each vertex of G' in increasing order according to their order in the breadth-first traversal such that the id assigned to v is 1.
8: $num \leftarrow |R(v)|$
9: Create a globally accessible integer array $allocated[1..num]$
10: Create a globally accessible integer array $available[1..num]$
11: Create a globally accessible integer array $longestPath[1..num]$
12: **for** $i \leftarrow num$ **downto** 2 **do**
13: Let u be such that $id(u) = i$
14: ALLOCATEHOLE(G', u)
15: $totalAllocatedHoles \leftarrow 0$
16: **for** $i \leftarrow 2$ **to** num **do**
17: $totalAllocatedHoles \leftarrow totalAllocatedHoles + allocated[i]$
18: **return** $totalAllocatedHoles$ ▷ Returns k for (v, k)-reachability in G

The array *allocated* stores the information whether a hole is allocated to a given vertex in G' or not and array *available* stores the number of holes available to a given vertex in G'. While the array *longestPath* stores the length of the longest path from a given vertex in G'.

In **for** loop in lines 12-14, the procedure ALLOCATEHOLE is called for every vertex $u \in V(G') - \{v\}$ in the decreasing order of their id to decide whether u requires to be empty or not in order to ensure $ECG(G', S_u)$ where $S_u = \{(x, y) \mid x, y \in V(G'), 1 < id(y) < id(u), (x, y) \in E(G')\}$ is (v, k)-reachable where k is the number of empty vertices in $ECG(G', S_u)$. When the loop execution terminates G' becomes (v, k)-reachable where k is the number of empty vertices in G'.

The procedure ALLOCATEHOLE described in Algorithm 3 is used to decide whether a hole needs to be allocated at a particular vertex. The **for** loop in lines 6-8 computes the length of the longest path from u and the number of holes available to u from it's descendants. The **if** condition in line 9 decides whether a hole needs to be allocated to u.

Now we give an upper bound on the time complexity of Algorithm 2 in terms of the number of vertices n. The DFS and BFS in lines 2 and 6, respectively, can be done in $\mathcal{O}(|R(v)|)$ [6]. The **for** loop in lines 12-14 runs for $|R(v)| - 1$ times and

Algorithm 3 To check whether it is necessary to allocate hole at the specified vertex in the tree.

```
1: procedure allocateHole(G', u)
                                          ▷ G' is a directed tree and u ∈ V(G')
                  ▷ id is the hash map containing unique ids of each of the vertices in G'
                          ▷ allocated is an array containing the present allocation of holes
                    ▷ available[i] contains the present count of holes available to vertex i
                ▷ longestPath[i] contains the present length of longest path from vertex i
2:     i ← id(u)
3:     longestPath[i] ← 0
4:     allocated[i] ← 0
5:     available[i] ← 0
6:     for each w ∈ G'.Adj[u] do
7:         longestPath[i] ← maximum(longestPath[i], longestPath[id(w)] + 1)
8:         available[i] ← available[i] + available[id(w)]
9:     if available[i] ≤ longestPath[i] then
10:        allocated[i] ← 1
11:        available[i] ← available[i] + 1
12:    return          ▷ Returns after setting the longest path length from u
                        ▷ as well as the decision whether to allocate hole to u or not
```

the method allocateHole is called once in every iteration which takes $c_1 + d_u c_2$ time for some constants c_1, c_2 and d_u corresponds to the outgoing degree of vertex (for which this method is executed). Thus, the total time complexity of the loop is $(|R(v)|-1)c_1 + c_2 \sum_{u \in R(v)} d_u < (|R(v)|-1)(c_1+c_2)$ and is bounded by $\mathcal{O}(|R(v)|)$.

The **for** loop in lines 16-17 takes $\mathcal{O}(|R(v)|)$ time. Thus, the overall complexity of Algorithm 2 is $\mathcal{O}(|R(v)|)$. As $|R(v)| \leq n$, hence the running time can be stated as $\mathcal{O}(n)$. The correctness of the algorithm is proved in Theorem 5.5.

Theorem 5.4. *In a given directed tree G' with the robot placed at $v \in V(G')$, the Algorithm 3 decides whether a hole has to be allocated at a specific vertex (say $u \in V(G')$) such that the edge contraction graph $ECG(G', S)$ is (v, k)-reachable for some k where set $S = \{(w, z) \mid w, z \in V(G'), 1 < id(z) < id(u)\}$ and id is the hash map as defined in line 7 of Algorithm 2. In the current allocation of holes, the edge contraction graph $ECG(G', S_1)$ is already (v, k')-reachable for some k' where $S_1 = S \cup \{(w, u) \mid w \in V(G'), (w, u) \in E(G')\}$.*

Proof. Let $A = \{x \mid x = u$ or x is a descendant of u $\}$ and $A' = V(ECG(G', S))$ - A - $\{v\}$. Let H_{new} and H_{old} be the set of vertices which are empty in the current allocation and the previous allocation respectively. Let $H_{1,new} = H_{new} \cap A$ and $H_{2,new} = H_{new} \cap A'$. Similarly, $H_{1,old} = H_{old} \cap A$ and $H_{2,old} = H_{old} \cap A'$.

Based on whether a hole is allocated to vertex u or not, we need to show that the edge contraction graph $ECG(G', S)$ is indeed (v, k)-reachable where $k = k'$ or

$k' + 1$ (depending on whether a hole is allocated to u or not). There are, in total, two possible cases.

Case 1: *When the algorithm decides to keep u empty.* In order to prove (v, k)-reachability, we need to check that the required conditions as described in definition 3.2 are met.

- *With the current allocation of holes and for all $x \in R(v)$, $C^v_{H_{new}} \rightarrow C^x_-$.*
  Here $H_{new} = H_{old} \cup \{u\}$. As u already contains a hole coupled with the fact that $ECG(G', S_1)$ is already (v, k')-reachable for some k' implies that $ECG(G', S)$ is v-reachable.
- *There is no set $Q \subset V(ECG(G', S))$ with $|Q| < k$ such that for all $x \in R(v)$, $C^v_Q \rightarrow C^x_-$.*
  Here $k = k' + 1$. On contrary, assume that $\exists Q \subset V(ECG(G', S))$ with $|Q| \leq k'$ such that $ECG(G', S)$ is v-reachable. Let $Q_1 = Q \cap A$ and $Q_2 = Q \cap A'$. Clearly, $u \notin Q$. Otherwise, the allocation $Q' = Q - \{u\}$ with size at most $k' - 1$ for $ECG(G', S_1)$ satisfies the condition (a) in definition 3.2, resulting in contradiction of (v, k')-reachability of $ECG(G', S_1)$.
  We claim that $|Q_1| \geq |H_{1,old}|$ and $|Q_2| \geq |H_{2,old}|$. On contrary, assume that $|Q_1| < |H_{1,old}|$. Then the allocation $Q' = Q_1 \cup H_{2,old}$ with size at most $k' - 1$ for $ECG(G', S_1)$ satisfies the condition (a) in definition 3.2, resulting in contradiction of (v, k')-reachability of $ECG(G', S_1)$. Hence, $|Q_1| \geq |H_{1,old}|$. On similar grounds, we can show that $|Q_2| \geq |H_{2,old}|$.
  Thus, $|Q| \geq |H_{old}| = k'$. But from our assumption we have, $|Q| \leq k'$ implying that $|Q| = k'$. Also, $|Q_1| = |H_{1,old}|$ and $|Q_2| = |H_{2,old}|$. But in this allocation Q, the number of holes available to u will be similar to that in case of allocation H_{old} which is a contradiction. The reason is mentioned before in reference to the **if** condition in line 9, that these holes are not sufficient to allow the robot to travel to the longest path from u in $ECG(G', S)$. Thus, no such Q exists.
  Thus, there exists no set $Q \subset V(ECG(G', S))$ with $|Q| < k$ such that for all $x \in R(v)$, $C^v_Q \rightarrow C^x_-$.

Case 2: *When the algorithm decides to keep an obstacle at u.* In order to prove (v, k)-reachability, we need to check that the required conditions as described in definition 3.2 are met.

- *With the current allocation of holes and for all $x \in R(v)$, $C^v_{H_{new}} \rightarrow C^x_-$.*
  Here $H_{new} = H_{old}$. As the algorithm has decided to keep an obstacle at u, it implies that the if condition in lines 9-11 is not satisfied and hence the number of holes available to u is at least one more than the length of the longest path from u in $ECG(G', S)$.
  For each $x \in V(ECG(G', S))$, $x \in A'$, there exists a sequence of

moves such that $C^v_{H_{new}} \rightarrow C^x_-$ (this is guaranteed by the fact that $ECG(G', S_1)$ is (v, k')-reachable).
For each $x \in V(ECG(G', S))$, $x \in A$, $x \neq u$, there exists at least one hole in H_{new} which will not be used by the robot to reach out to x in $ECG(G', S_1)$. This hole can be borrowed by u in $ECG(G', S)$ and the robot can then move to u and then the same sequence of moves as was used in reaching out to x in $ECG(G', S_1)$ from v can be used to reach out to x in $ECG(G', S)$.
However if $x = u$, then there exists at least one hole in H_{new} which can be borrowed by u in $ECG(G', S)$ and then the robot at v can move to u.
Thus, $\forall x \in R(v)$ in $ECG(G', S)$, $C^v_{H_{new}} \rightarrow C^x_-$.

- *there is no set $Q \subset V(ECG(G', S))$ with $|Q| < k$ such that for all $x \in R(v)$, $C^v_Q \rightarrow C^x_-$.*
Here $k = k'$. On contrary, assume that $\exists Q \subset V(ECG(G', S))$ with $|Q| < k'$ such that $ECG(G', S)$ is v-reachable. Consider a set $Q' = Q - \{u\}$ with size at most $k' - 1$. Clearly, $ECG(G', S_1)$ is v-reachable with Q' as the set of holes which is a contradiction as $ECG(G', S_1)$ is already (v, k')-reachable.
Thus, there exists no set $Q \subset V(ECG(G', S))$ with $|Q| < k$ such that for all $x \in R(v)$, $C^v_Q \rightarrow C^x_-$. □

Theorem 5.5. *The Algorithm 2 computes the minimum number of holes required to ensure (v, k)-reachability in a given directed tree G with the robot placed at $v \in V(G)$.*

Proof.

We will make use of the following loop invariant to prove the theorem.

Before the start of each iteration of the for loop in lines 12-14, the edge contraction graph $ECG(G', S'_u)$ where $S'_u = \{(x, y) \mid x, y \in V(G'), 1 < id(y) \leq id(u), (x, y) \in E(G')\}$ is (v, k)-reachable where k is the number of empty vertices in $ECG(G', S'_u)$.

Initialization: Initially $S'_u = E(G')$. Therefore, $ECG(G', S'_u)$ contains only v which contains the robot. Hence, $ECG(G', S'_u)$ is $(v, 0)$-reachable and the loop invariant holds.

Maintenance: Assume the statement to be true for all $j < i \leq num$ where $j \geq 2$. Now we need to prove it for $i = j = id(u)$. Let $x \in V(G')$ such that $id(x) = j + 1$. As the loop invariant holds for $i = j + 1$, the edge contraction graph $ECG(G', S'_x)$ is (v, k)-reachable where k is the number of empty vertices in $ECG(G', S'_x)$. In the iteration with $i = j+1$, the procedure ALLOCATEHOLE is called to decide whether a hole needs to be allocated to x or not and by theorem 5.4, the edge contraction graph $ECG(G', S'_u)$ is (v, k')-reachable

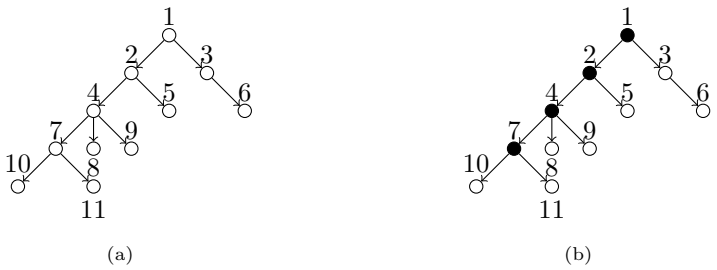

Fig. 4. A directed tree with labels allocated using Algorithm 2. (a) The tree before allocation. (b) The tree after allocation.

where k' denotes the number of empty vertices in $ECG(G', S'_u)$. Hence, the loop invariant is maintained.

Termination: The loop is terminated when $i = 1$, that is, $u = v$. The set $S'_v = \phi$ and hence the edge contraction graph $ECG(G', S'_v) = G'$ and with our invariant, the directed tree G' is (v, k)-reachable where k is the number of vertices that are empty in G'. □

We illustrate Algorithm 2 for a tree shown in Figure 4 in which the robot is placed at the vertex with *id* 1. The algorithm first assigns a numeric *id* to each vertex in the order of discovery (see Figure 4(a)). It then allocates hole to vertices numbered 10 and 11 as they do not have other holes available to them and the maximum length of a path from these vertices is 1 (including the vertex itself). On reaching vertex with *id* 7, the algorithm decides not to allocate hole to it as the maximum length of a path from this vertex is 2 and it already has two holes available from the vertices with *id* 10 and 11. The complete allocation of holes is shown in Figure 4(b).

6. Conclusions

We considered robot motion planning problem for undirected and directed graphs. We defined two variants of reachability viz. (v, k)-reachability and complete (v, k)-reachability and proved that they are equivalent for undirected graphs. We also proved that considering complete reachability from a node is equivalent to any other node in a given undirected connected graph. We have given algorithms to compute minimum number of holes required for complete reachability in undirected and directed graphs. We proved correctness of our algorithms and showed that the complexity of our algorithm for (v, k)-reachability is linear in terms of the number of vertices and edges in case of undirected graphs and strongly connected directed graphs and it is linear in the number of vertices in case of directed trees.

References

[1] Nancy M. Amato and Guang Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, 2002.

[2] V. Auletta, A. Monti, M. Parente, and P. Persiano. A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, 23:223–245, 1999.

[3] Vincenzo Auletta, Angelo Monti, Mimmo Parente, and Pino Persiano. A linear time algorithm for the feasibility of pebble motion on trees. In *Scandinavian Workshop on Algorithm Theory*, volume 1097 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 1996.

[4] Vincenzo Auletta and Pino Persiano. Optimal pebble motion on a tree. *Information and Computation*, 165:42–68, February 2001.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory With Applications*. Elsevier Science Ltd/North-Holland, 1976.

[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. In *Latin American theoretical informatics conference*, volume 3887 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 262-273 2006.

[8] Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215 – 228, 1999.

[9] Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 2011.

[10] J. E. Hopcroft, J. T. Schwartz, and M Sharir. On the complexity of motion planning for multiple independent objects; pspace hardness of the "warehouseman's problem". Technical report, Courant Institute of Mathematical Sciences, New York University, 1984.

[11] John E. Hopcroft and Gordon Wilfong. Reducing Multiple Object Motion Planning To Graph Searching. Technical report, Cornell University, June 1984.

[12] Christopher Jefferson, Angela Miguel, Ian Miguel, and S. Armagan Tarim. Modelling and solving english peg solitaire. *Computers and Operations Research*, 33(10):2935–2959, 2006.

[13] L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(3):205–218, 1986.

[14] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science, 1984*, pages 241–250. IEEE Computer Society, 1984.

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

[16] Christos H. Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion Planning on a Graph. In *Foundations of computer science*, pages 511–520. IEEE, 1994.

[17] Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, July 1990.

[18] Douglas B. West. *Introduction to Graph Theory*. Pearson, 2nd edition, 2000.

[19] Zhilin Wu and Stéphane Grumbach. Feasibility of Motion Planning on Directed Graphs. In *Annual Conference on Theory and Applications of Models of Computation*, pages 430–439. Springer-Verlag, 2009.

[20] Zhilin Wu and Stéphane Grumbach. Feasibility of Motion Planning on Acyclic and Strongly Connected Directed Graphs. *Discrete Applied Mathematics*, 158(9):1017–1028, 2010.