Bipartite Roots of Graphs
(Extended Abstract)

Lap Chi Lau
Department of Computer Science
University of Toronto
chi@cs.toronto.edu

Abstract

Graph H is a root of graph G if there exists a natural number k such that $xy \in E(G) \Leftrightarrow d_H(x,y) \leq k$ where $d_H(x,y)$ is the length of a shortest path in H from x to y. In such a case, H is a k-th root of G and we write $G = H^k$ and call G the k-th power of H. Motwani and Sudan proved that it is NP-complete to recognize squares of graphs and believed it is also NP-complete to recognize squares of bipartite graphs. In this paper, we show, rather surprisingly, that squares of bipartite graphs can be recognized in polynomial time. Also, we show that counting the number of different bipartite square roots of a graph can be done in polynomial time although this number could be exponential in the size of the input graph. Furthermore, we can generate all bipartite roots of a graph G in time $\mathcal{O}(\max\{\Delta(G) \cdot M,\ r(G)\})$ where $\Delta(G)$ is the maximum degree of G, M is the time complexity to do matrix multiplication, and $r(G)$ is the number of different bipartite square roots of G. By using the tools developed, we are able to give a new and simpler linear time algorithm to recognize squares of trees and a new algorithmic proof that tree square roots, when they exist, are unique up to isomorphism. Finally, we prove the NP-completeness of recognition of cubes of bipartite graphs.

1 Introduction

Root and *root finding* are concepts familiar to most branches of mathematics. In graph theory, H is a *root* of $G = (V, E)$ if there exists a positive integer k such that

$$xy \in E(G) \Leftrightarrow d_H(x,y) \leq k,$$

where $d_H(x,y)$ is the length of a shortest path in H from x to y. If H is a kth root of G, then we write $G = H^k$ and call G the kth power of H. Note that the terms "power" and "root" are used because of their close relationship with matrix multiplication. Also, graph roots are associated with problems in distributed computing [13] and computational biology [17, 11] where graph roots are useful in the reconstruction of phylogeny.

For any class of graphs, recognition is a fundamental structural and algorithmic problem; in this paper, we study the recognition problems on graph powers. Ordinarily, it is a difficult task to determine whether a given graph G has a k-th root or not. Also, the number of k-th roots could be exponential in the size of the input graph. In 1960, Ross and Harary [20] characterized squares of trees and showed that tree square roots, when they exist, are unique up to isomorphism. In 1967, Mukhopadhyay [16] characterized general graphs which possess a square root and in the following year Geller [7] solved the problem for general digraphs. In 1974, Escalante, Montejano and Rojano [4] characterized graphs and digraphs with a k-th root. However, all characterization on powers of general graphs are not polynomial in the sense that they do not yield a polynomial time algorithm. The complexity of graph power recognition was unresolved until 1994 when Motwani and Sudan [15] proved the NP-completeness of recognizing squares of graphs and stated that they believed it is also NP-complete to recognize squares of bipartite graphs. About the same time, Lin and Skiena [12] gave linear time algorithms to recognize squares of trees and, based on the characterization given by Harary, Karp and Tutte [8], to find square roots of planar graphs.

1.1 Our results In this paper, we will present a $\mathcal{O}(\Delta(G) \cdot M)$ algorithm to recognize squares of bipartite graphs. Our algorithm is constructive in the sense that it will construct a bipartite square root of the input graph if it has one. First, in section 2, given a graph G, we show that if we specify the neighborhood of one vertex in the bipartite roots of G, then there is at most one bipartite root of G that satisfies this condition and that bipartite root can be computed in $\mathcal{O}(M)$ time. Then, in section 3, we reduce the general problem to at most $2 \cdot \Delta(G)$ instances of the above special case and thus the complexity of recognizing squares of bipartite graphs is $\mathcal{O}(\Delta(G) \cdot M)$. With the same complexity, in section 4, we can count the number of different bipartite square roots of a graph although this number could

be exponential in the size of the graph. Furthermore, we can generate all bipartite roots of a graph in time $\mathcal{O}(\Delta(G) \cdot M,\ r(G))$. Finally, we present a new and simpler linear time algorithm to recognize squares of trees, a new algorithmic proof that tree square roots, when they exist, are unique up to isomorphism and the NP-completeness proof of recognizing cubes of bipartite graphs. Our basic notation and terminology reference is [22].

1.2 Related Research The literature is rich with results on graph roots and powers. Given a graph G with property P, does G^k have property P? Substantial work has been done on closure properties of powers of special classes of graphs, such as chordal graphs [1], interval graphs [18], co-comparability graphs [3], strongly chordal graphs [19], circular arc graphs [19], and AT-free graphs [18]. Given a graph G, what can be said about the properties of G^k? Since the number of edges increases with the index of the power of a graph, it is natural to expect that sufficiently large powers do possess some Hamiltonian type properties. For instance, Fleischner [5] proved that the square of every 2-connected graph is Hamiltonian; Sekanina [21] proved that the cube of every non-trivial connected graph is Hamiltonian connected. Besides the mathematical property questions on graph powers, the following is an obvious question to ask from an algorithmic point of view. Given a graph G, can we solve some optimization problems in G^k efficiently? Many optimization problems remain difficult in the case of powers of graphs [12]. On the other hand, the chromatic number of the square of a planar graph can be approximated within a constant factor in polynomial time [14].

2 Preliminaries

In general, the theoretically most efficient way to compute the square of a graph is to do matrix multiplication. By means of a linear time reduction, one can show that computing the square of a bipartite graph has the same complexity as computing the square of a general graph [10]. We will show, however, that computing a bipartite square root of a graph is much easier than computing an arbitrary square root of a graph. The problem is formally stated and some observations are presented in the following.

Problem	SQUARE OF BIPARTITE GRAPH (SB)
Instance	A graph $G = (V, E)$.
Question	Does there exist a *bipartite graph* $B = (X, Y, E')$ such that $B^2 = G$?

Without loss of generality, we assume G and B are connected. By observing that vertices on different sides of a bipartite graph do not share common neighbors, we claim the following easy result which is fundamental in later proofs.

PROPOSITION 2.1. *Let B be a bipartite graph such that $B^2 = G$. If $uv \in E(G)$ and u, v are on different sides of B, then $uv \in E(B)$.*

Now, we show that if we specify the neighborhood of one vertex in the bipartite roots, there is at most one such bipartite root and that root can be constructed in linear time.

Problem	SQUARE OF BIPARTITE GRAPH WITH ONE SPECIFIED NEIGHBORHOOD (SBN)
Instance	A graph $G = (V, E)$, $v \in V(G)$ and $U \subseteq N_G(v)$.
Question	Does there exist a bipartite graph $B = (X, Y, E')$ such that $B^2 = G$ and $N_B(v) = U$?

The following is a linear time algorithm to construct the unique solution (if some solution exists).

ALGORITHM 2.1. SBN

$C_1 \leftarrow \{v\}$
$C_2 \leftarrow U$
$V_2 \leftarrow C_1 \cup C_2$
$k \leftarrow 2$
while $(V_k \subset V(G))$ **do**
    $C_{k+1} \leftarrow N_G(C_{k-1}) - V_k$
    $V_{k+1} \leftarrow V_k \cup C_{k+1}$
    $k \leftarrow k + 1$
$X \leftarrow \bigcup_{i \geq 1} C_{2i-1}$
$Y \leftarrow \bigcup_{i \geq 1} C_{2i}$
$E' \leftarrow \{xy \mid x \in X, y \in Y \text{ and } xy \in E(G)\}$

LEMMA 2.1. *Given an instance of* SBN, *the algorithm constructs the unique solution in linear time, if some solution exists.*

Proof. (Sketch) Starting from $v \in X$ and $U \subseteq Y$, we enlarge the *forced bipartition* of B in each iteration. In the first iteration, $C_3 = N_G(v) - N_B[v]$ and it is not difficult to see that C_3 must be placed in X. In the $(k-1)$-th iteration of the algorithm for $k \geq 2$, V_k is the set of vertices that are forced to be on one side of B (while $C_1, C_3, \ldots$ must be placed in X and $C_2, C_4, \ldots$ must be placed in Y). If $V_k \subset V(G)$ and G is connected, $C_{k+1} \neq \emptyset$ and it is forced to be on the opposite side as C_k in B (or otherwise $B^2 \neq G$). Since G is connected, an execution of the above algorithm will terminate. By Proposition 2.1, $E' = \{uw \mid u \in X, w \in Y \text{ and } uw \in E(G)\}$ is the *forced edge set* for any bipartite graph B such that $B^2 = G$ and $N_B(v) = U$; this proves the lemma. Note that Algorithm 2.1 can be implemented

to run in linear time. An illustration of Algorithm 2.1 is shown in Figure 1.

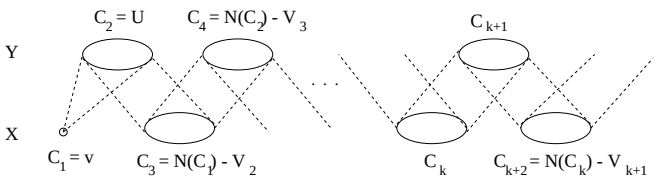

Figure 1: An illustration of Algorithm 2.1

LEMMA 2.2. SBN *can be solved in* $\mathcal{O}(M)$ *time.*

Proof. Note that by Algorithm 2.1, B is constructed for any input graph G. Therefore, we have to verify if $B^2 = G$ and this requires matrix multiplication.

In fact, Lemma 2.2 implies a polynomial time algorithm to recognize squares of bipartite graphs with a small degree vertex (e.g., trees, bipartite planar graphs) by exhaustively "guessing" a small degree vertex and its neighborhood.

3 Squares of bipartite graphs

We say a vertex v in a bipartite graph is *maximal* if $N_B(v) \not\subset N_B(u)$ for all $u \in V(B)$ on the same side as v; an edge $e = uv$ in a bipartite graph is *maximal* if both u and v are maximal vertices. When a vertex in a bipartite graph is adjacent to all vertices on the opposite side, we say it is a *universal* vertex. By extending the idea of SBN, we introduce the following problem:

Problem	SQUARE OF BIPARTITE GRAPH WITH ONE SPECIFIED MAXIMAL EDGE (SBE)
Instance	A graph $G = (V, E)$, $xy \in E(G)$.
Question	Does there exist a bipartite graph $B = (X, Y, E')$ such that $B^2 = G$ and xy is a *maximal edge* in B?

Notice that if SBE is polynomial time solvable, then so is SB. To see this, given an instance of SB, we pick a vertex x in G with maximum degree and note that it must be a maximal vertex in B for any bipartite square root B of G. Also, if G is the square of a bipartite graph B, there is at least one vertex $y \in N_G(x)$ such that $y \in N_B(x)$ and y is maximal in B (consider a vertex y with maximum degree amongst vertices in $N_B(x)$). Since $N_G(x)$ is of cardinality at most $\Delta(G)$, the time complexity of SB is at most $\Delta(G)$ times the complexity of SBE. Our main result is that SBE can be solved in $\mathcal{O}(M)$ time. In particular, we reduce SBE to at most 2 instances of SBN. Given the partial solution where initially x and y are on different sides, the algorithm incrementally enlarges the partial solution until it can be reduced to at most 2 instances of SBN.

3.1 Preliminaries We say that a graph G with $\theta(G) \leq 2$ ($\theta(G)$ is the clique cover number of G) a *co-bipartite graph* (the complement of a bipartite graph). Given a co-bipartite graph H; we denote the connected components of $\overline{H}$ by $\overline{C}_1, \ldots, \overline{C}_k$ and we say they are the *co-components* of H (the connected components of the complement). A co-component is *trivial* if its vertex set is of size 1; otherwise it is *non-trivial*. Henceforth, when we say a co-component, we mean a non-trivial co-component. For any co-component $\overline{C}_i$ of a co-bipartite graph H, $\overline{H}[\overline{C}_i]$ is a connected bipartite graph, we call the vertex set that corresponds to a partite set of $\overline{H}[\overline{C}_i]$ a *part*. Let H be an induced co-bipartite subgraph of G and let $\overline{C}_1, \ldots, \overline{C}_k$ be its co-components. A vertex $v \in V(G) - V(H)$ is of:

- *type 0* to $\overline{C}_i$ if v is not adjacent to any vertex in $\overline{C}_i$;
- *type 1* to $\overline{C}_i$ if v is adjacent to some vertices in exactly one part, called *adjacent part*, of $\overline{C}_i$;
- *type 2* to $\overline{C}_i$ if v is adjacent to some vertices in both parts of $\overline{C}_i$.

A vertex v is type 1 *universal* to $\overline{C}_i$ if it is adjacent to all vertices in the adjacent part; similarly, v is type 2 *universal* to $\overline{C}_i$ if it is adjacent to all vertices in $\overline{C}_i$. Suppose x and y are the vertices that are part of the input to SBE; without loss of generality, we assume $x \in X$ and $y \in Y$ in B. Let $P_x = N_G(x) - N_G(y)$, $P_y = N_G(y) - N_G(x)$ and we say $P_{xy} = P_x \cup P_y$ is the set of *private neighbors*. Also, we say $C_{xy} = N_G(x) \cap N_G(y)$ is the set of *common neighbors*.

LEMMA 3.1. *Suppose* $B = (X, Y, E)$, $xy \in E(B)$ *and* $B^2 = G$; *then* $(N_B(x) - y) \cup (N_B(y) - x) = C_{xy}$ *and* $G[C_{xy}]$ *is a co-bipartite graph.*

By Lemma 3.1, $G[C_{xy}]$ is a co-bipartite graph. We let the trivial co-components of $G[C_{xy}]$ be $\overline{c_1}, \ldots, \overline{c_l}$ and the co-components be $\overline{C}_1, \ldots, \overline{C}_k$. Before we proceed to the outline of the algorithm, we first show an important lemma of placing co-components.

LEMMA 3.2. *Suppose* $B = (X, Y, E)$, $xy \in E(B)$ *and* $B^2 = G$. *Given a co-component* $\overline{C}$ *of* $G[C_{xy}]$; *all vertices in one part of* $\overline{C}$ *must be on one side of* B *while all vertices in the other part of* $\overline{C}$ *must be on the other side of* B.

Proof. Let U and W be the two parts of $\overline{C}$ and $U = U_1 \cup U_2$ and $W = W_1 \cup W_2$. Suppose that, in B, U_1 and W_1 are in X while U_2 and W_2 are in Y. Since U_1

and W_1 are in X, by Proposition 2.1, y is adjacent to every vertex in U_1 and W_1 in B and thus $G[U_1 \cup W_1]$ is a clique. By the same argument, $G[U_2 \cup W_2]$ is a clique. Also $G[U]$ and $G[W]$ are cliques by the definition of a part of a co-component. So in G we have all possible edges between $\{U_1 \cup W_2\}$ and $\{U_2 \cup W_1\}$ and thus they are disconnected in $\overline{G}$. Now we have contradicted the fact that $\overline{C}$ is a co-component unless one of $\{U_1 \cup W_2\}$ or $\{U_2 \cup W_1\}$ is empty and thus the lemma holds.

In order words, given a co-component $\overline{C} = \{U \cup W\}$ of $G[X_{xy}]$ where U and W are the two parts of $\overline{C}$, we just have to consider the *orientation* of $\overline{C}$ in B (i.e. whether U is placed in X and W is placed in Y *or* U is placed in Y and W is placed in X). If the orientation of a co-component is fixed (i.e. it was placed by the algorithm), we refer to it as a *fixed co-component*; otherwise, it is a *free co-component*. At the beginning, every co-component is free.

3.2 Outline of the algorithm Our algorithm will place $N_G(x) \cup N_G(y)$ in B in several stages; in each stage some vertices will be placed in B. In the first stage, all vertices in P_{xy} of G will be placed in B. Then, all trivial co-components of $G[C_{xy}]$ will be placed in B in the second stage. Note that if there is just one free co-component $\overline{C}$ left, then we are finished since we just have to consider at most two possibilities (two instances of SBN). The difficult case is when we have many free co-components. So, henceforth, when we place co-components in B, we make the assumption that there are at least two free co-components left.

Using the position of vertices in P_{xy} in B and the edges in G between vertices in P_{xy} and the free co-components, we place the free co-components in B. In particular, we first look for some easy configurations in G to place some or all free co-components in B. Then, if there are still at least two free co-components left, the unresolved graph G has some very special structures and we use them to place the remaining free co-components in B. When finished, by Lemma 3.1, we are able to determine $N_B(x)$ and $N_B(y)$ and thus the problem is reduced to SBN. The algorithm outline is as follows.

ALGORITHM 3.1. SBE

1 PLACE PRIVATE NEIGHBORS

Description: Place P_x in X and P_y in Y.

2 CHECK CO-BIPARTITENESS

Description: Ensure that $G[C_{xy}]$ is a co-bipartite graph.

3 PLACE TRIVIAL CO-COMPONENTS

Description: Place all trivial co-components.

Post-condition: (3.1) In B, there is no edge between any trivial co-component and vertices in P_{xy}.

4 NO TYPE 0 VERTEX

Assumption: (3.1)

Description: Ensure that (4.1) happens.

Post-condition: (4.1) For $v \in P_{xy}$, v is not a type 0 vertex to any co-component.

5 TYPE 2 FORCING

Assumption: (4.1)

Description: If $v \in P_{xy}$ is a type 2 vertex to a co-component, then we place all co-components such that v is not a universal type 2 vertex to them.

Post-condition: (5.1) If $v \in P_{xy}$ is a type 2 vertex to a co-component in G, then v is a universal type 2 vertex to all free co-components in G.

Post-condition: (5.2) If $v \in P_{xy}$ is a type 1 vertex to a free co-component in G, then v is a type 1 vertex to all co-components in G.

6 TYPE 1 FORCING

Assumption: (3.1), (5.2)

Description: We place all co-components *unless* (6.1) happens.

Post-condition: (6.1) For all $v \in P_{xy}$, if v is a type 1 vertex to a free co-component in G, then v is a universal type 1 vertex to all co-components and v's adjacent part of all fixed co-components were placed on the same side as v.

7 MORE TYPE 1 FORCING

Assumption: (5.2)

Description: If there exist type 1 vertices in P_{xy}, then we place all co-components *unless* (7.1) happens.

Post-condition: (7.1) Type 1 vertices in P_{xy} that are on the same side have the same neighborhood and type 1 vertices in P_{xy} that are on different sides have disjoint neighborhood.

8 PLACE INCOMPLETE CO-COMPONENT

Assumption: (5.2), (6.1)

Description: If $v \in P_{xy}$ is a type 1 vertex, then we place all incomplete co-components.

Post-condition: (8.1) If there exists a type 1 vertex in P_{xy}, then every free co-component is a complete co-component.

9 FINAL PLACEMENT I

Assumptions: (4.1), (5.1), (5.2), (6.1), (7.1), (8.1).

Description: Place all co-components if $V(G) = N_G(x) \cup N_G(y)$.

10 FINAL PLACEMENT II

Assumptions: (4.1), (5.1), (5.2), (6.1).

Description: Place all co-components if $V(G) \subset N_G(x) \cup N_G(y)$.

3.3 Algorithm Now we present the details of each part. If at any point the **return** command is executed, the output is returned and the execution halts. We omit the details of PLACE PRIVATE NEIGHBORS and CHECK CO-BIPARTITENESS since they are trivial.

3.3.1 Place Trivial co-Components We can assume $G[C_{xy}]$ is a co-bipartite graph by Lemma 3.1. The following two technical lemmas are useful in PLACE TRIVIAL CO-COMPONENTS.

LEMMA 3.3. *Suppose $B = (X, Y, E)$, $xy \in E(B)$ and $B^2 = G$. Let $\overline{c}$ be a trivial co-component of $G[C_{xy}]$ in G. If $\overline{c} \in X$, then $N_B[x] \subseteq N_B[\overline{c}]$ and $N_G[x] \subseteq N_G[\overline{c}]$. Otherwise, if $\overline{c} \in Y$, then $N_B[y] \subseteq N_B[\overline{c}]$ and $N_G[y] \subseteq N_G[\overline{c}]$. (Proof omitted.)*

LEMMA 3.4. *Suppose $B = (X, Y, E)$ and $B^2 = G$. Let u and v be vertices in different partite sets of B. Then $N_G[u] = N_G[v]$ if and only if u and v are both universal vertices in B. (Proof omitted.)*

ALGORITHM 3.2. PLACE TRIVIAL CO-COMPONENTS

Case 1 : $P_{xy} \neq \emptyset$
  for any trivial co-component $\overline{c}$
    if $N_G[\overline{c}] = N_G[x]$
      then place $\overline{c}$ in X
    else if $N_G[\overline{c}] = N_G[y]$
      then place $\overline{c}$ in Y
    else return "NO"

Case 2 : $P_{xy} = \emptyset$
  for any trivial co-component $\overline{c}$
    place $\overline{c}$ *arbitrarily* in X or Y

LEMMA 3.5. PLACE TRIVIAL CO-COMPONENTS *is correct.*

Proof. (Sketch) **Case 1:** $P_{xy} \neq \emptyset$. By Lemma 3.3 and the maximality of x, if $N_G[\overline{c}] \neq N_G[x]$, then $\overline{c}$ cannot be placed in X. Also, by Lemma 3.4 and the maximality of y, if $N_G[\overline{c}] = N_G[x]$, then $\overline{c}$ must be placed in X. The same argument applies when x is replaced by y and thus the lemma holds.

Case 2: $P_{xy} = \emptyset$. In this case, x and y are both universal vertices in B. It can be proven that regardless of the position of $\overline{c}$, the square graph is the same.

COROLLARY 3.1. *In B, there is no edge between any trivial co-component $\overline{c}$ of $G[C_{xy}]$ and P_{xy}.*

Proof. By PLACE TRIVIAL CO-COMPONENTS, $N_G[\overline{c}]$ is either equal to $N_G[x]$ or $N_G[y]$. This implies that $N_B[\overline{c}]$ is either equal to $N_B[x]$ or $N_B[y]$ and thus $\overline{c}$ is not adjacent to any vertex in P_{xy} in B.

3.3.2 No Type 0 Vertex We now show that for any vertex $v \in P_{xy}$, it is either a type 1 or a type 2 vertex to a co-component.

LEMMA 3.6. *Suppose $B = (X, Y, E)$, $xy \in E(B)$, $B^2 = G$ and $v \in P_{xy}$ is adjacent to $z \in C_{xy}$ in B. If $\overline{C}$ is a co-component such that $z \notin \overline{C}$ and there is a vertex $u \in \overline{C}$ but $uv \notin E(G)$, then $\overline{C}$ can not be placed such that u and v are on the same side of B.*

Proof. Suppose, by way of contradiction, that u and v are on the same side of B. Since z is in a different co-component than $\overline{C}$, $zu \in E(G)$. Since $zu, zv \in E(G)$, by Proposition 2.1, $zu, zv \in E(B)$. Hence, $uv \in E(B^2)$ but this contradicts the assumption that $B^2 = G$.

LEMMA 3.7. NO TYPE 0 VERTEX *is correct.*

Proof. Suppose, by way of contradiction, that v is a type 0 vertex to a co-component $\overline{C}$ in G. Without loss of generality, we assume that $v \in P_x$ in B. By Corollary 3.1, v is not adjacent to any trivial co-component in B. Since $v \in P_x$, v and x have a common neighbor z in B. So z must be in a co-component $\overline{C}'$ such that $\overline{C}' \neq \overline{C}$. Since $\overline{C}$ is a co-component, there are vertices u, w on different parts of $\overline{C}$ that are not adjacent to v in G. Since $vz \in E(B)$ and $z \notin \overline{C}$, by Lemma 3.6, neither u nor w can be placed on the same side as z or otherwise the assumption that $B^2 = G$ is contradicted. Therefore, if $B^2 = G$, v does not exist and the lemma holds.

COROLLARY 3.2. *After the execution of* NO TYPE 0 VERTEX*, for any vertex $v \in P_{xy}$ and any co-component $\overline{C}$, v is either a type 1 vertex to $\overline{C}$ or a type 2 vertex to $\overline{C}$.*

3.3.3 Type 2 Forcing

ALGORITHM 3.3. TYPE 2 FORCING

for $v \in P_{xy}$
  if v is a type 2 vertex to a co-component $\overline{C}'$
    for any free co-component $\overline{C} \neq \overline{C}'$
      if there is a vertex $u \in \overline{C}$ s.t. $uv \notin E(G)$
        place $\overline{C}$ s.t. u is on the side opposite v
    if there are at least two free co-components left
      if there is a vertex $u \in \overline{C}'$ s.t. $uv \notin E(G)$
        place $\overline{C}'$ s.t. u is on the side opposite v

LEMMA 3.8. *Suppose $B = (X, Y, E)$, $xy \in E(B)$, $B^2 = G$ and $v \in P_{xy}$ is a type 2 vertex to a co-component $\overline{C}'$. If $\overline{C}$ is a co-component such that $\overline{C} \neq \overline{C}'$ and there is a vertex $u \in \overline{C}$ that is not adjacent to v in G, then $\overline{C}$ can not be placed such that u and v are on the same side of B.*

Proof. Let w and z be vertices of different parts of $\overline{C}'$ such that they are adjacent to v. By Lemma 3.2, exactly one of w or z is on the side opposite v in B. Without loss of generality, we assume z is on the side opposite v in B. By Proposition 2.1, $zv \in E(B)$. Hence, by Lemma 3.6, this lemma follows.

LEMMA 3.9. TYPE 2 FORCING *is correct.*

Proof. The first **for** loop is justified by Lemma 3.8. If there are at least two free co-components left, there exists a co-component $\overline{C} \neq \overline{C}'$ such that v is a universal type 2 vertex to $\overline{C}$. By applying Lemma 3.8 with $\overline{C}'$ is replaced by $\overline{C}$, the lemma follows.

COROLLARY 3.3. *After the execution of* TYPE 2 FORCING, *if v is a type 2 vertex to a co-component in B and there are at least two free co-components left, then v is a universal type 2 vertex to all free co-components.*

COROLLARY 3.4. *After the execution of* TYPE 2 FORCING, *if v is a type 1 vertex to a free co-component in G, then v is a type 1 vertex to all co-components in G.*

3.3.4 Type 1 Forcing

LEMMA 3.10. *Suppose $B = (X, Y, E)$, $xy \in E(B)$ and $B^2 = G$. If $v \in P_{xy}$ is a type 1 vertex to all co-components in G, there is exactly one co-component $\overline{C}$ such that the adjacent part of $\overline{C}$ to v is on the opposite side as v in B. Furthermore, v is a universal type 1 vertex to any co-component $\overline{C}'$ in G where $\overline{C}' \neq \overline{C}$.*

Proof. (Sketch) First it is easy to see that there is at least one co-component in B such that the adjacent part of v is on the opposite side as v; otherwise, v does not share a common neighbor with x or y, a contradiction. Suppose there are more than one such co-component or v is not a universal type 1 vertex to a co-component with the adjacent part of v on the same side as v, by similar argument as in Lemma 3.8, we have $B^2 \neq G$; thus the lemma holds.

Let $v \in P_{xy}$ be a type 1 vertex to all co-components in G. By Lemma 3.10, if $xy \in E(B)$ and $B^2 = G$, then there is exactly one co-component $\overline{C}$ such that the adjacent part of $\overline{C}$ to v is on the opposite side as v in B; we call $\overline{C}$ the *active co-component* of v in B.

ALGORITHM 3.4. TYPE 1 FORCING

for $v \in P_{xy}$ that is a type 1 vertex to a free co-component
  if there exists a fixed co-component $\overline{C}$ s.t. $\overline{C}$'s adjacent part to v is on the side opposite v
    for any free co-component $\overline{C}'$
      place $\overline{C}'$ s.t. the adjacent part to v is on the same side as v
    return SBN
  if there is one free co-component $\overline{C}$ s.t. v is not a universal type 1 vertex to $\overline{C}$
    place $\overline{C}$ as the active co-component of v in B
    for any free co-component $\overline{C}' \neq \overline{C}$
      place $\overline{C}'$ s.t. $\overline{C}'$'s adjacent part to v is on the same side as v
    return SBN

LEMMA 3.11. TYPE 1 FORCING *is correct.*

Proof. (Sketch) In either case, $\overline{C}$ must be the active co-component by Lemma 3.10 and the lemma follows.

COROLLARY 3.5. *After the execution of* TYPE 1 FORCING, *if $v \in P_{xy}$ is a type 1 vertex to a free co-component in G, then v is a universal type 1 vertex to all co-components in G. Furthermore, for any fixed co-component $\overline{C}$, the adjacent part of $\overline{C}$ to v is on the same side as v in B.*

By Corollary 3.3 and Corollary 3.5, after the execution of TYPE 1 FORCING, a vertex $v \in P_{xy}$ is either a universal type 2 vertex to all free co-components or a universal type 1 vertex to all co-components. For the sake of simplicity, henceforth, we refer a vertex of the former case a *type 2 vertex* and a vertex of the latter case a *type 1 vertex.*

3.3.5 More Type 1 Forcing

ALGORITHM 3.5. MORE TYPE 1 FORCING

1. **if** u and v are type 1 vertices in P_{xy} and are on the same side in B and there exists a free co-component $\overline{C}$ s.t. u's and v's adjacent parts on $\overline{C}$ are different
   for each orientation of $\overline{C}$
     place free co-components by Lemma 3.10
     if SBN **return** TRUE
   return FALSE
2. **if** u and v are type 1 vertices in P_{xy} and are on different sides in B and there exists a free co-component $\overline{C}$ s.t. u's and v's adjacent parts on $\overline{C}$ are the same
   for each orientation of $\overline{C}$
     place free co-components by Lemma 3.10
     if SBN **return** TRUE
   return FALSE

LEMMA 3.12. MORE TYPE 1 FORCING *is correct.*

Proof. (Sketch) In either case, $\overline{C}$ must be the active co-component of some vertex by Corollary 3.5 and thus the lemma follows from Lemma 3.10.

COROLLARY 3.6. *If after the execution of* MORE TYPE 1 FORCING *we still have two free co-components left, then we have the following. For type 1 vertices in P_{xy}: if they are on the same side, their neighborhood on the free co-components are the same; if they are on different sides, their neighborhoods on the free co-components are disjoint.*

3.3.6 Place Incomplete co-Components We say a co-component $\overline{C}$ is *complete* if there is no edge between the two parts of $\overline{C}$ (i.e. a complete bipartite graph in the complement); otherwise, $\overline{C}$ is an *incomplete* co-component. Now, if there is a type 1 vertex $v \in P_{xy}$, then we will fix the orientations of all free incomplete co-components.

ALGORITHM 3.6. PLACE INCOMPLETE CO-COMPONENT

if there is a type 1 vertex $v \in P_{xy}$
  for any free co-component $\overline{C}$
    if $\overline{C}$ is an incomplete co-component in G
      place $\overline{C}$ s.t. the adjacent part of $\overline{C}$ to v is on the same side as v

LEMMA 3.13. PLACE INCOMPLETE CO-COMPONENT *is correct.*

Proof. Let w, z be two vertices in different parts of $\overline{C}$ and $wz \in E(G)$. Without loss of generality, we assume w is on the adjacent part of v. Suppose, $\overline{C}$ is placed so that w and v are on different sides in B. By Corollary 3.5, $vw \in E(G)$. Since $vw, wz \in E(G)$, by Proposition 2.1, $vw, wz \in E(B)$. Therefore, $vz \in E(B^2)$; a contradiction.

COROLLARY 3.7. *After the execution of* PLACE INCOMPLETE CO-COMPONENT, *if there is a type 1 vertex in P_{xy}, then every free co-component is complete.*

3.3.7 Final Placement I Notice that by Corollary 3.6, the unresolved graph has some very special structures. For example, by Corollary 3.6, if there exist type 1 vertices in P_{xy}, the active co-component (we do not know which one at this point) is the same for all type 1 vertices. We called an orientation of the free co-components such that exactly one free co-component is the active co-component *a valid orientation.* Now we use the special structures of the graph to place all free co-components. We have two cases to consider; in FINAL PLACEMENT I, we consider the case when $N_G[x] \cup N_G[y] = V(G)$ (the case when $N_G[x] \cup N_G[y] \subset V(G)$ is handled by FINAL PLACEMENT II).

ALGORITHM 3.7. FINAL PLACEMENT I: when $N_G[x] \cup N_G[y] = V(G)$

Case 1: there exists a type 1 vertex in P_{xy}
  find an *arbitrary* valid orientation

Case 2: there is no type 1 vertex in P_{xy}
  find an *arbitrary* orientation

return SBN

LEMMA 3.14. FINAL PLACEMENT I *is correct.*

Proof. (Sketch) By the special structures of the unresolved graph as in Corollary 3.6, it can be shown that any two placements by the algorithm yield the same square graph.

3.3.8 Final Placement II In FINAL PLACEMENT II, we consider the case when $N_G[x] \cup N_G[y] \subset V(G)$. We will use the neighborhood of a carefully chosen "outside" vertex to decide the orientations of free co-components.

ALGORITHM 3.8. FINAL PLACEMENT II: when $N_G[x] \cup N_G[y] \subset V(G)$

pick a vertex $u \in N_G[C_{xy}] - N_G[x] - N_G[y]$
Case 1: u is a universal type 1 vertex to all free co-components in G
  for orientation of the free co-components s.t. adjacent parts of u are on the same side
    if SBN **return** TRUE
  return FALSE
Case 2: u is a universal type 1 vertex to exactly one free co-component $\overline{C}$ in G
  find an orientation with only $\overline{C}$ active
  if SBN **return** TRUE
  else return FALSE
Otherwise:
  return "NO"

LEMMA 3.15. FINAL PLACEMENT II *is correct.*

Proof. (Sketch) First, u must exist or otherwise B is disconnected. By the choice of u, it must be adjacent to a vertex $v \in P_{xy}$.

Case 1: there exists $v \in N_B(u) \cap P_{xy}$ that is a type 2 vertex in G. By Corollary 3.3, v is a universal type 2 vertex in G and thus it is universally adjacent to exactly one part of each free co-component in B. So u is a universal type 1 vertex to all free co-components in $B^2 = G$. Those parts are on the opposite side as v in B. So there are only two possibilities left (correspond to possible positions of v), we try both.

Case 2: all vertices in $N_B(u) \cap P_{xy}$ are type 1 vertices in G. By Lemma 3.10, v is adjacent to exactly one free co-component in B. So u is adjacent to exactly

one free co-component in B^2. Then $\overline{C}$ must be the active co-component in B and the lemma follows from Lemma 3.10.

Note that in either case, all free co-components are placed and thus SBE is reduced to at most two instances of SBN.

THEOREM 3.1. SBE *can be solved in* $\mathcal{O}(M)$ *time.*

Proof. The correctness of this algorithm follows from the correctness of all steps. Also, notice that except the final verification step, all steps can easily be implemented in $\mathcal{O}(n^2)$.

THEOREM 3.2. SQUARE OF BIPARTITE GRAPH *can be solved in* $\mathcal{O}(\Delta(G) \cdot M)$.

4 Counting and generating bipartite square roots of a graph

It is natural to ask how many different bipartite square roots a graph can have. In fact, by looking at the SBE algorithm carefully, the only flexibility in the algorithm of placing vertices is in PLACING TRIVIAL CO-COMPONENT when $P_{xy} = \emptyset$ and FINAL PLACEMENT I when $N_G[x] \cup N_G[y] = V(G)$. In the former case, a trivial co-component can be placed in either X or Y; in the latter case, depending on the existence of type 1 vertices, either an arbitrary valid orientation or an arbitrary orientation is considered. As mentioned before, in either case, two arbitrary placements will have the same square graph. Hence, when we are just concerned with the existence of a bipartite square root, it suffices to test for an arbitrary placement (i.e. a representative). When we are concerned with the number of different bipartite square roots of G, if the representative is checked to be a bipartite square root of G, then we have to count the number of arbitrary placements, denoted by n_p. We denote the number of trivial co-components by t and the number of free co-components by f. Fortunately, it is easy to count the number of arbitrary placements, summarized as follows:

Case 1 $P_{xy} = \emptyset$: $n_p = 2^{t+f}$

Case 2 $P_{xy} \neq \emptyset$, $V(G) = N_G[x] \cup N_G[y]$ and there is no type 1 vertex: $n_p = 2^f$

Case 3 $P_{xy} \neq \emptyset$, $V(G) = N_G[x] \cup N_G[y]$ and there is a type 1 vertex: $n_p = f$

THEOREM 4.1. *Given* G*, the number of different bipartite roots* $r(G)$ *of* G *can be counted in* $\mathcal{O}(\Delta(G) \cdot M)$.

Proof. First we pick a vertex x with maximum degree in G; we know that it must be a maximal vertex in any bipartite square root B of G. Then we sort the vertices in $N_G(x)$ by non-increasing degree. Suppose the resulting ordering is $\{y_1, \ldots, y_k\}$; we reduce SB to k instances of SBE by following the order of the sorted vertices. Consider an instance of SBE of x and y_i; we add additional constraints that $\{y_1, \ldots, y_{i-1}\}$ must be on the same side as x. Notice that by adding these additional constraints, they will not affect the execution of the algorithm. In fact, these just help the algorithm to to fix the free co-components.

It is clear that we do not over-count. The only possible solutions that we may exclude are where xy_j is a maximal edge in the solution but some y_i is also in Y where $i < j$. Consider the smallest such i, we argue that those solutions are included when we count the solutions of SBE with xy_i as a maximal edge. The crucial point is that the only place we use the maximality of y_i in the algorithm is in PLACE TRIVIAL CO-COMPONENTS, where we forbid a trivial co-component $\overline{c}$ with $N_G(y_i) \subset N_G(\overline{c})$ to be placed in Y by maximality of y_i. Since $i < j$, $N_G(y_i) \not\subset N_G(y_j)$. Hence, those solutions are included earlier and thus the algorithm counts correctly.

Finally, notice that the additional constraints do not increase the complexity of the algorithm. Also, the additional counting step can be performed in linear time (by just counting the number of trivial co-components and free co-components left); the theorem follows.

THEOREM 4.2. *Given* G*, all different bipartite roots of* G *can be generated in* $\mathcal{O}(\max\{\Delta(G) \cdot M,\ r(G)\})$.

5 Squares of Trees

Clearly a tree is a bipartite graph. We will use our tools developed for bipartite graphs to give new proofs of the existing results for trees. We will reduce SQUARE OF TREE to one instance of SBN. By doing so, we first show that if $G = T^2$, then a maximal clique S in G corresponds to $N_T[v]$ for a vertex $v \in S$. In such a case, we call v the *center* of S in T.

LEMMA 5.1. *Suppose* $T^2 = G$*; if* $S \subseteq V(G)$ *induces a maximal clique in* G*, then* $S = N_T[v]$ *for a vertex* $v \in S$. *(Proof omitted)*

Given a maximal clique S in G, by Lemma 5.1, if we can deduce the center of S, then the problem is reduced to an instance of SBN.

LEMMA 5.2. *Given a maximal clique* S *in* G*, if* $v_1, v_2 \in S$ *share a common neighbor* w *in* $G - S$*, then either* $N_T[v_1] = S$ *or* $N_T[v_2] = S$. *In other words, either* v_1 *or* v_2 *is the center of* S. *(Proof omitted)*

THEOREM 5.1. *(see also [12])* SQUARE OF TREES *can be solved in linear time.*

Proof. First of all, we find an arbitrary maximal clique S in G. By Lemma 5.1, S corresponds to $N_T[v]$ for a $v \in S$.
Case 1: $S = V(G)$. In this case, G is a complete graph and any complete star is a tree square root of G.
Case 2: $S \subset V(G)$. By Lemma 5.2, if two vertices v_1, v_2 in S share a common neighbor in $G - S$ in G, then one of them is the center. It is easy to see, if T is connected, there is at least one such pair of vertices.
Case 2a: There are at least two distinct pairs of vertices. We pick two arbitrary distinct pairs. By Lemma 5.2, if $G = T^2$, there is exactly one vertex v (the center) that appears in more than one pair and thus $N_T[v] = S$. So, in this case, the problem is reduced to an instance of SBN.
Case 2b: There is only one distinct pair of vertices. Suppose the center of S in T is v, it is easy to see that all vertices in $S - v$ are leaves in T except exactly one internal vertex u. So $N_{T^2}(v) \subseteq N_{T^2}(u)$. Let v_1, v_2 be the only pair of vertices. Since $G = T^2$, if $N_G(v_1) \subset N_G(v_2)$, then v_1 must be the center and thus the problem is reduced to an instance of SBN. The only case left is when $N_G(v_1) = N_G(v_2)$. In this case, any tree root is of diameter 3 (a double star), and thus v_1 and v_2 are indistinguishable.

Now we show that the algorithm can be implemented in linear time. First, a maximal clique S in G can be found in linear time. To find a pair of vertices in S that share a common neighbor in $G - S$, it suffices to check the neighborhood in S for every vertex in $G - S$; every edge is visited at most once. Once we find two distinct pairs, we can reduce the problem to an instance of SBN. So at any time of the algorithm, we just have to store one such pair of vertices. Notice that in any case, the problem is reduced to at most one instance of SBN. By Lemma 2.2, the unique solution can be constructed in linear time, if some solution exists. The final verification step can be done in linear time [12] and we are done.

THEOREM 5.2. *(see also [20]) Tree square roots of a graph, when they exist, are unique up to isomorphism.*

Proof. From the proof of Theorem 5.1, there are only two cases where we can not pin down exactly the center of the maximal clique. The first case is when the tree square root is a star while the second case is when the tree square root is a double star. In both cases, the tree square roots of G are isomorphic. Note that if we can pin down the center of the maximal clique, then by Lemma 2.2, the solution is unique.

5.1 Cubes of bipartite graphs Since SQUARE OF BIPARTITE GRAPH is polynomial time solvable, it is natural to ask if we can find a bipartite k-th root of a graph in polynomial time for $k \geq 3$. We observe that Proposition 2.1 does not hold for $k \geq 3$. In fact, we show that it is NP-complete to determine if a given graph G is the cube of a bipartite graph. In our reduction we use SET SPLITTING as formulated in [6]. The idea of the reduction is similar to that in [15].

Given an instance of SET SPLITTING, we construct an instance of CUBE OF BIPARTITE GRAPH. Let $S = \{u_1, \ldots, u_n\}$ be the set of elements and let $C = \{c_1, \ldots, c_m\}$ denote the set of subsets of S. The SET SPLITTING problem is to find a partition of S into S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2. The graph G is constructed as follows:

Vertices of G

- *Element vertices:* U_i: $1 \leq i \leq n$ for each element u_i.
- *Subset vertices:* C_j for each subset $c_j \in C$ and tail vertices C_j^1, C_j^2, C_j^3 for each c_j.
- *Partition vertices:* S_1 and S_2.
- *Connection vertex:* X.

Edges of G

- *Edges of tail vertices of subset vertices:* $\forall c_j \in C$, $C_j^3 \leftrightarrow C_j^2, C_j^3 \leftrightarrow C_j^1, C_j^3 \leftrightarrow C_j$ $C_j^2 \leftrightarrow C_j^1, C_j^2 \leftrightarrow C_j, C_j^2 \leftrightarrow U_i$ for all $u_i \in c_j$, $C_j^1 \leftrightarrow C_j, C_j^1 \leftrightarrow C_i$ iff $c_j \cap c_i \neq \emptyset, C_j^1 \leftrightarrow U_i$ iff $u_i \in c_j, C_j^1 \leftrightarrow S_1, C_j^1 \leftrightarrow S_2, C_j^1 \leftrightarrow X$.
- *Edges of subset vertices:* $\forall c_j \in C$, $C_j \leftrightarrow S_1, C_j \leftrightarrow S_2, C_j \leftrightarrow X, C_j \leftrightarrow U_i$ for all $i, C_j \leftrightarrow C_i$ iff $c_j \cap c_i \neq \emptyset$.
- *Edges of element vertices:* $\forall u_j \in S$, $U_j \leftrightarrow U_i$ for all i, $U_j \leftrightarrow S_1$, $U_j \leftrightarrow S_2$, $U_j \leftrightarrow X$.
- *Edges of partition vertices:* $S_1 \leftrightarrow X$, $S_2 \leftrightarrow X$.

LEMMA 5.3. *If there is a partition of S into two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2, then there exists a bipartite graph B such that $B^3 = G$.*

Proof. (Sketch) **Edges of B.**

- *Edges of subset vertices and its tail vertices:* $C_j^3 \leftrightarrow C_j^2, C_j^2 \leftrightarrow C_j^1, C_j^1 \leftrightarrow C_j$ and $C_j \leftrightarrow U_i$ if and only if $u_i \in c_j$.
- *Edges of partition vertices:* $S_k \leftrightarrow U_i$ if and only if $u_i \in S_k$.

- *Edges of connection vertex:*
  $X \leftrightarrow U_i$ for all i.

It is a routine matter to check that $E(B^3) = E(G)$ and B is a bipartite graph.

We now show that if G has a cube root H (H is not necessarily a bipartite graph), then there is a partition of S into two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 and S_2. First, we need the following technical proposition.

PROPOSITION 5.1. *If H is a cube root of G, then, in H, C_j is adjacent to U_i if and only if $u_i \in c_j$. Also, in H, C_j^3 is only adjacent to C_j^2, C_j^2 is only adjacent to C_j^1 and C_j^3 and C_j^1 is only adjacent to C_j and C_j^2. (Proof omitted)*

LEMMA 5.4. *If H is a cube root of G, then there is a partition of S into two sets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2.*

Proof. In H, Proposition 5.1 forces the adjacencies of the tail vertices and subset vertices to its own elements only. So in H, S_1 and S_2 only have neighbors in the element set and X. Since $S_1 \not\Leftrightarrow S_2$ in G, they have no common element neighbor in H and so it is a partition. And since S_1 and S_2 are adjacent to all C_j^1, S_1 and S_2 must reach C_j in exactly two steps and thus each of S_1, S_2 must have a common neighbor with C_j in the element set for all j. Therefore, $N_H(S_1) \cap S$ and $N_H(S_2) \cap S$ is the desired partition; this completes the proof.

THEOREM 5.3. CUBE OF BIPARTITE GRAPH *is NP-complete.*

6 Further research

An obvious question is whether a bipartite square root of a graph can be computed in $\mathcal{O}(M)$ time. Another question is whether a tree k-th root can be computed in time less than $\mathcal{O}(n^3)$ [9]. Also, the complexity of recognition of squares of planar graphs is unknown.

Acknowledgment

I am grateful to my supervisor Derek Corneil for helpful suggestions.

References

[1] R. Balakrishnan, P. Paulraja, Powers of chordal graphs, *Australian Journal of Mathematics Series A*, 35, pp. 211-217, 1983.

[2] V.N. Bhat-Nayak, Powers and roots of graphs: a survey, *Optimization, design of experiments and graph theory (Bombay, 1986), Indian Inst. Tech., Bombay*, 246–282, 1988.

[3] P. Damaschke, Distances in cocomparability graphs and their powers, *Discrete Applied Mathematics*, 35, pp. 67-72, 1992.

[4] F. Escalante, L. Montejano, T. Rojano, Characterization of n-path graphs and of graphs having nth root, *J. Combin. Theory B*, 16, pp. 282-289, 1974.

[5] H. Fleischner, The square of every two-connected graph is Hamiltonian, *J. Combin. Theory B*, 16, pp. 29-34, 1974.

[6] M.R. Garey, D.S. Johnson, Computers and Intractability - A Guide to the Theory of NP-completeness, *Freeman, Oxford*, 1979.

[7] D.P. Geller, The square root of a digraph, *J. Combin. Theory B*, 5, pp. 320-321, 1968.

[8] F. Harary, R.M. Karp, W.T. Tutte, A criterion for planarity of the square of a graph, *J. Combin. Theory*, 2, pp. 395-405, 1967.

[9] P. Kearney, D. Corneil, Tree powers, *Journal of Algorithms*, 29, pp. 111-131, 1998.

[10] L.C. Lau, Bipartite roots of graphs, *http://www.cs.toronto.edu/~chi/sb.ps.*

[11] Guo-Hui Lin, Paul E. Kearney, Tao Jiang, Phylogenetic k-Root and Steiner k-Root, *Lecture Notes in Computer Science*, 1969, pp. 539-551, 2000.

[12] Y.L. Lin, S. Skiena, Algorithms for square roots of graphs, *SIAM J. Disc. Math.*, Vol. 8, No. 1, pp. 99-118, 1995.

[13] N. Linial, Locality in distributed graph algorithms, *SIAM J. Comput.*, 21, pp. 193-201, 1992.

[14] M. Molloy and M.R. Salavatipour, Frequency Channel Assignment on Planar Networks, *Proceedings of 10th Annual European Symposium on Algorithms (ESA 2002)*, LNCS 2461, pp. 736-747

[15] R. Motwani, M. Sudan, Computing roots of graphs is hard, *Discrete Applied Mathematics*, 54, pp. 81-88, 1994.

[16] A. Mukhopadhyay, The square root of a graph, *J. Combin. Theory*, 2, pp. 290-295, 1967.

[17] N. Nishimura, P. Ragde, D. Thilikos, On graph powers for leaf-labeled trees, *Journal of Algorithms*, 42, pp. 69-108, 2002.

[18] A. Raychaudhuri, On powers of interval and unit interval graphs, *Congr. Numer.*, 59, pp. 235-242, 1987.

[19] A. Raychaudhuri, On powers of strongly chordal and circular arc graphs, *Ars Combin.*, 34, pp. 147-160, 1992.

[20] I.C. Ross, F. Harary, The square of a tree, *Bell System Tech. J.*, 39, pp. 641-647, 1960.

[21] M. Sekanina, On an ordering of the vertices of a graph, *Časopis Pěst. Math.*, 88, pp. 265-282, 1963.

[22] D. West, Introduction to graph theory, *Prentice Hall*, Ed. 2, 2001.