Counting Spanning Trees in Cographs: An Algorithmic Approach

Stavros D. Nikolopoulos* Charis Papadopoulos† ‡

Abstract

In this paper we present a new simple linear-time algorithm for determining the number of spanning trees in the class of complement reducible graphs, also known as cographs; for a cograph G on n vertices and m edges, our algorithm computes the number of spanning trees of G in $O(n+m)$ time and space, where the complexity of arithmetic operations is measured under the uniform cost criterion. The algorithm takes advantage of the cotree of the input cograph G and works by contracting it in a bottom-up fashion until it becomes a single node; then, the number of spanning trees of G is computed as the product of a collection of values which are associated with the vertices of G and are updated during the contraction process. The correctness of our algorithm is established through the Kirchhoff matrix tree theorem, and also relies on structural and algorithmic properties of the class of cographs. We also extend our results to a proper superclass of cographs, namely the P_4-reducible graphs, and show that the problem of finding the number of spanning trees of a P_4-reducible graph has linear-time solution.

Keywords: Cographs, P_4-reducible graphs, number of spanning trees, modular decomposition, combinatorial problems, algorithms, complexity.

1 Introduction

We consider finite undirected graphs with no loops or multiple edges. Let G be such a graph on n vertices. A *spanning tree* of G is a connected acyclic $(n-1)$-edge subgraph of the graph G. The number of spanning trees of a graph (network) G, is an important, well-studied quantity in graph theory, and

*Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece; e-mail address: stavros@cs.uoi.gr

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway; e-mail address: charis@ii.uib.no

‡Work by Charis Papadopoulos was carried while he was a graduate student at the Department of Computer Science, University of Ioannina.

appears in a number of applications. Most notable application fields are network reliability (in a network modeled by a graph, intercommunication between all nodes of the network implies that the graph must contain a spanning tree; thus, maximizing the number of spanning trees is the key to maximizing reliability) [18], computing the total resistance along an edge in an electrical network [5], and enumerating certain chemical isomers [6].

Thus, both for theoretical and practical purposes, we are interested in deriving formulas for or computing the number of spanning trees of a graph G, and also of the K_n-complement of the graph G; if G is a subgraph of the complete graph K_n, then the K_n-complement of G, denoted by $K_n - G$, is defined to be the graph obtained from K_n by removing the edges of G (note that, if G has n vertices, then $K_n - G$ coincides with the graph $\overline{G}$, the complement of G). Many cases have been examined depending on the choice of G. It has been studied when G is a labeled molecular graph [6], when G is a circulant graph [27, 28], when G is a complete multipartite graph [25], when G is a cubic cycle and quadruple cycle graph [26], when G is a quasi-threshold graph [20], and so on (see Berge [2] for an exposition of the main results; also see [8, 20, 21, 22, 23, 24, 25]).

The purpose of this paper is to study the problem of finding the number of spanning trees in the class of *complement reducible* graphs, or so-called *cographs*, a well-known class of perfect graphs [4, 12]. The cographs were introduced in the early 1970s by Lerchs [16] and defined as the class of graphs formed from a single vertex under the closure of the operations of union and complement. More precisely, the class of cographs is defined recursively as follows: (i) a single-vertex graph is a cograph; (ii) the disjoint union of cographs is a cograph; (iii) the complement of a cograph is a cograph. Lerchs [16] has also shown that the class of cographs coincides with the class of graphs which contain no induced subgraph isomorphic to a P_4 (chordless path on four vertices), and that a cograph G admits a unique tree representation, up to isomorphism, called a *cotree* $T_{co}(G)$.

Traditionally, the problem of finding the number of spanning trees of a graph is solved by means of the classic *Kirchhoff matrix tree theorem* [3]. This result expresses the number of spanning trees of a graph G in terms of the determinant of a cofactor of the so-called Kirchhoff matrix that can be easily constructed from the adjacency relation (adjacency matrix, adjacency lists, etc) of the graph G; the Kirchhoff matrix tree theorem states that the number of spanning trees of a graph is equal to any of the cofactors of the Kirchhoff matrix. An alternative way for computing the number of spanning trees of a graph, can be achieved through the calculation of the eigenvalues of the Kirchhoff matrix [15]. We point out that both approaches can be used for computing the number of spanning trees of any graph G (see [2, 8, 11, 21, 25]), but they necessitate $\Theta(n^3)$ time and $\Theta(n^2)$ space, where n is the number of vertices of the graph G.

For some classes of graphs, such as threshold graphs, quasi-threshold graphs, cographs, circulant graphs, etc, there are nice characterizations for the eigenvalues of their Kirchhoff matrix through their constructive properties which hold

for these classes of graphs [13, 26, 27, 28]. Thus, in these cases, one has the advantage of associating the number of spanning trees through known formulas for the eigenvalues of their Kirchhoff matrix. On the other hand, for other classes of graphs it is common to compute the cofactor of the Kirchhoff matrix of a given graph G by using structural properties of G along with standard techniques from linear algebra and matrix theory; see for example [8, 11, 18, 20, 21, 22, 24, 25].

Based on structural properties of the class of cographs, Hammer and Kelmans [13] proposed a linear-time algorithm for the number of spanning trees of a cograph G using the approach of computing the eigenvalues of the Kirchhoff matrix of G. In particular, the computation of the eigenvalues of the Kirchhoff matrix is achieved by applying, recursively, a polynomial which corresponds to the operations of union and complement of a cograph.

In this paper we propose a different approach for solving the problem of finding the number of spanning trees of a cograph G. We use tree contraction operations and show that the stated problem can be efficiently solved by successively applying these operations on the cotree $T_{co}(G)$ of a cograph G. Our approach avoids to compute the determinant of a cofactor of the Kirchhoff matrix or the eigenvalues described in a certain polynomial.

In particular, we present a linear-time algorithm for determining the number of spanning tree of a cograph; for an input cograph G on n vertices and m edges our algorithm constructs first the cotree $T_{co}(G)$ of the graph G, and then computes the number of spanning trees of G in $O(n+m)$ time and space. The algorithm relies on tree contraction operations which are applied in a systematic fashion from bottom to top in order to shrink the $T_{co}(G)$ into a single node, while at the same time certain parameters are appropriately updated; since the cotree of a graph can be constructed in time and space linear in the size of the graph [7], and each tree contraction operation takes time linear in the size of the contracted part of the tree, the above computation takes time and space linear in the size of G. The number of spanning trees of G is obtained as the product of $n-1$ numbers, where n is the number of vertices of G; this takes $O(n)$ time under the uniform cost criterion [1]. The correctness of our algorithm is established by means of the Kirchhoff matrix tree theorem along with standard techniques from linear algebra and matrix theory.

The algorithmic approach we use in this paper for solving the problem of finding the number of spanning trees of a cograph allows us to design a simple and efficient algorithm: it does not make use of advanced data structures, it is easy to implement and its complexity analysis and its correctness are straightforward. Moreover, our approach can be extended to other classes of graphs and, thus, allows us to beat the $O(n^{2.376})$ time complexity for the problem of finding their number of spanning trees. Indeed, in this paper we extend our results to a proper superclass of cographs, namely the P_4-reducible graphs, and show that the number of spanning trees of a P_4-reducible graph G on n vertices and m edges can be computed in $O(n+m)$ time and space.

The paper is organized as follows. In Section 2 we establish the notation and related terminology and we present background results. In Sections 3 we

propose a linear-time algorithm for determining the number of spanning trees of a cograph, while in Section 4 we extend our results to a proper superclass of cographs, namely the P_4-reducible graphs. Finally, Section 5 concludes the paper and presents possible future extensions.

2 Definitions and Background Results

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The subgraph of a graph G induced by a subset S of the vertex set $V(G)$ is denoted by $G[S]$. For a vertex subset S of G, we define $G - S := G[V(G) - S]$; we denote $G - v$ the graph $G[V(G) - \{v\}]$. The neighborhood $N(v)$ of a vertex $v \in V(G)$ is the set of all the vertices of G which are adjacent to v; the closed neighborhood of x is defined as $N[v] := \{v\} \cup N(v)$.

2.1 Cographs

As mentioned above, Lerchs proved that cographs are the family of graphs constructed from a single-vertex under the closure of the operations of union and complement. These operations uniquely define a tree representation referred to as a cotree. The cotree of a cograph G, denoted $T_{co}(G)$, is a rooted tree such that:

(i) each internal node, except possibly for the root, has at least two children;

(ii) the internal nodes are labeled by either 0 (*0-nodes*) or 1 (*1-nodes*); the children of a 1-node (0-node resp.) are 0-nodes (1-nodes resp.), i.e., 1-nodes and 0-nodes alternate along every path from the root to any node of the cotree;

(iii) the leaves of the cotree are in a 1-to-1 correspondence with the vertices of G, and two vertices v_i, v_j are adjacent in G if and only if the least common ancestor of the leaves corresponding to v_i and v_j is a 1-node.

Lerchs' definition required that the root of a cotree be a 1-node; if however we relax this condition and allow the root to be a 0-node as well, then we obtain cotrees whose internal nodes all have at least two children, and whose root is a 1-node if and only if the corresponding cograph is connected.

Let G be a connected cograph, and let $T_{co}(G)$ be its corresponding cotree. We define the following node/vertex sets on the cotree $T_{co}(G)$:

- L_i, which is the set of nodes/vertices on the ith level of $T_{co}(G)$, and
- $ch(u_i)$, which is the set of children on the node $u_i \in T_{co}(G)$.

The parent of a node/vertex w in $T_{co}(G)$ is denoted by $p(w)$. Figure 1 features a cotree $T_{co}(G)$ with the corresponding level sets.

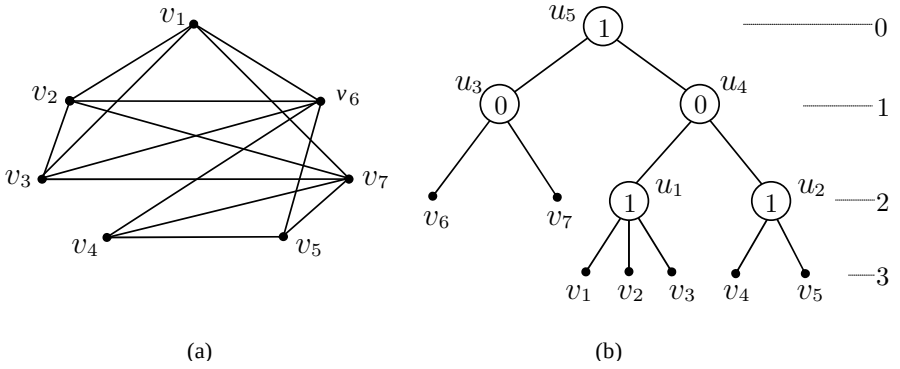

Figure 1: (a) A cograph on 7 vertices and (b) the corresponding cotree.

A set S of leaves of $T_{co}(G)$, or, equivalently, a subset $S \subseteq V(G)$ is called

- *strong block* if all the vertices in S have the same parent u in $T_{co}(G)$, and $\text{ch}(u) = S$.

Hereafter, we use $p(S)$ to denote the parent of the vertices of the strong block S, and we call it *strong node*; note that, $p(S)$ is a node in $T_{co}(G)$. In Figure 1, the only strong blocks of $T_{co}(G)$ are formed by nodes u_1, u_2 and by u_3; nodes u_1 and u_2 are strong 1-nodes, while u_3 is strong 0-node.

Based on the structural properties of a cograph G and its corresponding cotree $T_{co}(G)$, it is easy to see that the following lemma holds:

Lemma 2.1. *A cograph G contains at least one strong block.*

2.2 Modular Decomposition Tree

A subset $M \subset V(G)$ is said to be a *module* of the graph G, if every vertex outside M is either adjacent to all the vertices in M or to none of them. The empty set, the singletons, and the vertex set $V(G)$ are *trivial* modules and whenever G has only trivial modules it is called a *prime graph* (or *indecomposable*). A non-trivial module is also called a *homogeneous* set. A module M of G is called *strong*, if for any module $M' \neq M$ of G, either $M' \cap M = \emptyset$ or $M' \subset M$.

The modular decomposition of a graph G is represented by a tree $T(G)$ which we call the *modular decomposition tree* of G; the leaves of $T(G)$ are the vertices of G, whereas each internal node u corresponds to a strong module, denoted $M(u)$, which is induced by the set of vertices/leaves of the subtree rooted at u. Thus, $T(G)$ represents all the strong modules of G. Each internal node is labeled by either 0 (or P) for *parallel* module, 1 (or S) for *series* module, or 2 (or N) for *neighborhood* module. The module corresponding to a 0-node induces a disconnected subgraph of G, that of a 1-node induces a connected subgraph of G whose complement is a disconnected subgraph and that of a 2-node induces a connected subgraph of G whose complement is also a connected subgraph.

Specifically, let u be an internal node of the modular decomposition tree $T(G)$. If u has children $u_1, u_2, \ldots, u_p$, then we define the *representative graph* $G(u)$ of the module $M(u)$ as follows: $V(G(u)) = \{u_1, u_2, \ldots, u_p\}$, and $E(G(u)) = \{u_i u_j \mid v_i v_j \in E(G),\ v_i \in M(u_i) \text{ and } v_j \in M(u_j)\}$.

Note that by the definition of a module, if a vertex of $M(u_i)$ is adjacent to a vertex of $M(u_j)$ then every vertex of $M(u_i)$ is adjacent to every vertex of $M(u_j)$. Thus $G(u)$ is isomorphic to the graph induced by a subset of $M(u)$ consisting of a single vertex from each maximal submodule of $M(u)$ in $T(G)$. Then: (i) if u is a 0-node, $G(u)$ is an edgeless graph, (ii) if u is a 1-node, $G(u)$ is a complete graph, and (iii) if u is a 2-node, $G(u)$ is a prime graph.

The modular decomposition tree $T(G)$ of a graph G is constructed recursively as follows: parallel modules are decomposed into their connected components, series modules into their co-connected components, and neighborhood modules into their strong submodules. The efficient construction of the modular decomposition tree of a graph has received a great deal of attention. It is well known that for any graph G the tree $T(G)$ is unique up to isomorphism and it can be constructed in linear time [9, 17]. Note that if $T(G)$ does not contain any internal 2-node then G is a cograph and $T(G)$ is its cotree, i.e., $T(G) \equiv T_{co}(G)$.

2.3 Kirchhoff Matrix

For an $n \times n$ matrix A, the $(n-1)$-st order *minor* μ_j^i is the determinant of the $(n-1) \times (n-1)$ matrix obtained from A after having deleted row i and column j. The i-th *cofactor* equals μ_i^i. The *Kirchhoff matrix* K for a graph G on n vertices is an $n \times n$ matrix with elements

$$k_{i,j} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and } (i,j) \in E, \\ 0 & \text{otherwise,} \end{cases}$$

where d_i is the degree of the vertex v_i in the graph G. The Kirhhoff matrix tree theorem is one of the most famous results in graph theory. It provides a formula for the number of spanning trees of a graph G, in terms of the cofactors of its Kirhhoff matrix.

Theorem 2.1. (Kirchhoff Matrix Tree Theorem [3]): *For any graph G with K defined as above, the cofactors of K have the same value, and this value equals the number of spanning trees of G.*

3 The Number of Spanning Trees

Let G be a cograph on n vertices and m edges and let $T_{co}(G)$ be its cotree. In order to compute the number of spanning trees of the graph G we use Theorem 2.1; that is, we delete an arbitrary vertex x of the set $V(G)$ and all the edges

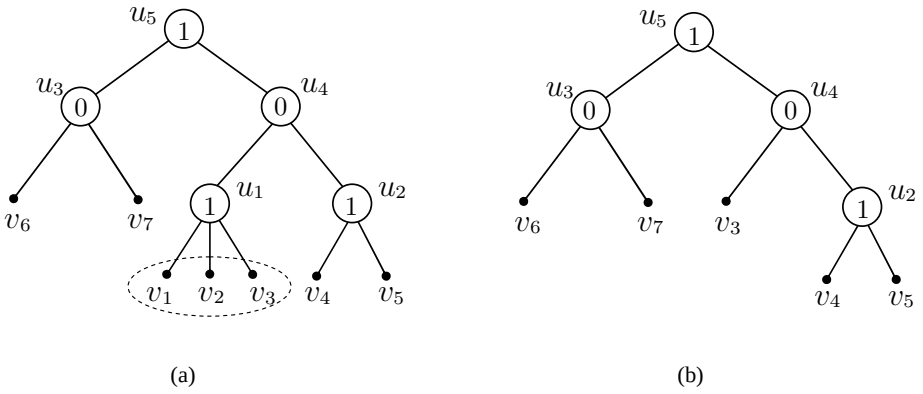

Figure 2: (a) The cotree $T_{co}(G)$ of a cograph G, and (b) the results of the function $Update\_Replace(u_1, T_{co}(G))$.

incident on vertex x. Now the vertex set of the resulting cograph $G - x$ is of size $n - 1$.

3.1 The Algorithm

We set $s(v) := d_v$ for every vertex $v \in V(G) - \{x\}$, where d_v is the degree of the vertex v in the input graph G; we call these labels of the vertices their *s-labels*.

The algorithm works by contracting in a systematic fashion the strong nodes of the cotree of the graph $G - x$ and by assigning to the leaf that is produced the highest-index vertex of G which is a child of the strong node (see Figure 2). The contractions are done by means of a function, namely, *Update_Replace*( ), which also update the s-labels of the children/vertices of the strong node. We note that the s-labels are assumed to be global variables in our algorithm.

Let u be a strong node of $T_{co}(G)$, and let $\mathrm{ch}(u) = \{v_1, v_2, \ldots, v_p\}$. The function Update_Replace is applied to note u, and works as follows:

- increase the s-labels $s(v_1), s(v_2), \ldots, s(v_p)$ by 1, in the case where u is a 1-node;
- compute the parameter $e(u) := \sum_{i=1}^{p} \frac{1}{s(v_i)}$;
- update the s-labels $s(v_{p-1})$ and $s(v_p)$ of the vertices v_{p-1} and v_p, using the parameter $e(u)$;
- delete the vertices $v_1, v_2, \ldots, v_{p-1}$ from $T_{co}(G)$, and replace the node u with the vertex v_p.

Figure 2 shows the application of the function Update_Replace on node $u_1 \in T_{co}(G)$. First, it computes the s-labels $s(v_1), s(v_2), s(v_3)$ and the parameter

$e(u_1)$, then recomputes the values $s(v_2), s(v_3)$, and finally deletes the vertices v_1 and v_2, and replaces the strong node u_1 with the vertex v_3. The formal description of the function Update_Replace is given in Algorithm 1.

Update_Replace(u, T)

---

1. Compute the vertex set $\mathrm{ch}(u) = \{v_1, v_2, \ldots, v_p\}$;

2. **if** u is 1-node **then**
for every vertex $v_i \in \mathrm{ch}(u)$ **do** $s(v_i) := s(v_i) + 1$;

3. Compute $e(u) := \sum_{i=1}^{p} \frac{1}{s(v_i)}$;

4. Update the s-label $s(v_{p-1})$ as follows:
$s(v_{p-1}) := s(v_{p-1}) \cdot s(v_p) \cdot e(u)$;

5. Update the s-label $s(v_p)$ as follows:
if u is 0-node **then** $s(v_p) := \frac{1}{e(u)}$
else $s(v_p) := \frac{1}{e(u)} - 1$;

6. Delete vertices $v_1, v_2, \ldots, v_{p-1}$ from T, and replace node u with vertex v_p;

7. Return the resulting tree;

Algorithm 1: *Update_Replace*(u, T)

We next describe our algorithm Number_Spanning_Trees which computes the number of spanning trees of a cograph G; it works as follows: First it computes the degree d_i of each vertex $v_i \in V(G)$ and assigns $s(v_i) := d_i$, $1 \leq i \leq n$. Then, it computes the graph $G := G - v_n$, where $v_n \in V(G)$, and constructs its cotree $T_{co}(G)$; recall that, v_i is a leaf of $T_{co}(G)$, $1 \leq i \leq n-1$. Next, it repeatedly applies the function Update_Replace( ) to each strong node u, and computes the s-labels $s(v_1), s(v_2), \ldots, s(v_{n-1})$ of the vertices of $T_{co}(G)$. Finally, it computes the number of spanning trees $\tau(G) := \prod_{i=1}^{n-1} s(v_i)$. The formal description of the above algorithm is given in Algorithm 2.

3.2 Correctness

The correctness of the algorithm Number_Spanning_Trees is established thought the Kirchhoff matrix tree theorem (Theorem 2.1), which implies that the number of spanning trees of a graph G is equal to any of the cofactors of the Kirchhoff matrix.

Number_Spanning_Trees

Input: A cograph G on n vertices $v_1, v_2, \ldots, v_n$ and m edges;

Output: The number of spanning trees $\tau(G)$ of the cograph G;

---

1. Compute the degree d_i for each vertex $v_i \in V(G)$;
2. Set $s(v_i) := d_i$, for $1 \leq i \leq n$;
3. Compute $G := G - v_n$, where $v_n \in V(G)$;
4. Construct the cotree $T := T_{co}(G)$ of the cograph G;
5. Compute the sets $L_0, L_1, \ldots, L_{h-1}$;
6. **for** $i = h-1$ down to 0 **do**
for every node $u \in L_i$ **do** $T := Update\_Replace(u, T)$;
7. Compute $\tau(G) := \prod_{i=1}^{n-1} s(v_i)$;

Algorithm 2: Number_Spanning_Trees

Let G be a cograph on n vertices and m edges and let $T_{co}(G)$ be the cotree of G consisting of n vertices (leaves) and k nodes (internal nodes). Let $L_0, L_1, \ldots, L_h$ be the level sets of $T_{co}(G)$ and let $S = \{v_1, v_2, \ldots, v_p\}$ be a strong block; note that $v_1, v_2, \ldots, v_p \in L_h$. We form the Kirhhoff matrix K of the cograph G, which is an $n \times n$ matrix, and then we compute the n-th cofactor of the matrix K. Recall that the n-th cofactor of K equals the $(n-1)$-st order *minor* μ_n^n, that is, the determinant of the $(n-1) \times (n-1)$ matrix K_{nn} obtained from K after deleting its n-th row and column. Substituting the values $s(v_i) = d_i$, $1 \leq i \leq n-1$, the matrix K_{nn} becomes

$$K_{nn} = \begin{bmatrix} s(v_1) & & & & & & & \\ & \ddots & & & & & (-1)_{ji} & \\ & & s(v_p) & & & & & \\ & & & s(v_{p+1}) & & & & \\ & & & & s(v_{p+2}) & & & \\ & (-1)_{ij} & & & & & \ddots & \\ & & & & & & & s(v_{n-1}) \end{bmatrix}, \quad (1)$$

where, according to the definition of the Kirchhoff matrix, $s(v_i)$ is equal to the degree of vertex v_i in G and the entries $(-1)_{ij}$ and $(-1)_{ji}$ of the off-diagonal positions (i, j) and (j, i) are both -1 if the vertices v_i and v_j are adjacent in G and are equal to 0 otherwise, $1 \leq i,\ j \leq n-1$.

The matrix K_{nn} corresponds to a cograph G on $n-1$ vertices $v_1, v_2, \ldots, v_{n-1}$, where each vertex v_i has an s-label $s(v_i) = d_i$, $1 \leq i \leq n-1$. The first p rows of K_{nn} correspond to the p vertices $v_1, v_2, \ldots, v_p$ of the strong block S; the next $(n-1)-p$ rows correspond to the vertices $v_{p+1}, v_{p+2}, \ldots, v_{n-1}$. Recall that, all the vertices of the strong block S have the same parent $p(S)$ and, also, the same degree in graph G.

We next focus on the computation of the determinant of the matrix K_{nn}, as it is known from the Kirchhoff matrix tree theorem (see Theorem 2.1) that

$$\tau(G) = \det(K_{nn}). \tag{2}$$

In order to compute the determinant $\det(K_{nn})$, we start by focusing on the first p rows and p columns of the matrix K_{nn} and work as follows:

We first multiply the p-th row of K_{nn} by -1 and add it to rows $1, 2, \ldots, p-1$. Then, in the first $p-1$ rows of K_{nn}, the non-zero entries are found only in positions (i,i) and (i,p), $1 \leq i \leq p-1$, and have values $s(v_i)$ and $-s(v_p)$, respectively, for the case where $p(S)$ is a 0-node, and have values $s(v_i)-1$ and $-s(v_p)-1$, respectively, for the case where $p(S)$ is a 1-node.

Next, we focus on the non-zero positions (i,p) of K_{nn}, $1 \leq i \leq p-1$, and do the following: if the parent $p(S)$ of the strong block $S = \{v_1, v_2, \ldots, v_p\}$ is a 1-node, then we multiply the column j by $\frac{s(v_p)+1}{s(v_j)+1}$ and add it to column p; otherwise, we multiply the column j by $\frac{s(v_p)}{s(v_j)}$ and add it to column p, $1 \leq j \leq p-1$. Now, in the first $p-1$ rows of K_{nn}, only the diagonal positions (i,i) have non-zero values. Additionally, the p-th row remains unchanged, except the entry in the position (p,p), and all the non-zero elements in positions (i,p) of column p, for $i = p+1, p+2, \ldots, n-1$, which were equal to -1 in the initial matrix K_{nn}, now have the same value

$$k \;=\; -\left(1 + (s(v_p)+a)\cdot \sum_{i=1}^{p-1} \frac{1}{s(v_i)+a}\right) \;=\; -(s(v_p)+a)\cdot\left(\sum_{i=1}^{p} \frac{1}{s(v_i)+a}\right),$$

where $a = 1$ if $p(S)$ is a 1-node, and $a = 0$ otherwise. Finally, we multiply the column p by $-\frac{1}{k}$ and the column $p-1$ by $-k$.

It is important to note that the above operations do not change the value of the determinant of matrix K_{nn}. Moreover, in the matrix that results after these operations, we have:

(i) all the off-diagonal elements in positions (i,j), $1 \leq i \leq p-1$ and $1 \leq j \leq n-1$, are equal to 0,

(ii) the off-diagonal elements in positions (i,j), $p \leq i, j \leq n-1$, have their initial values, that is, the values in the initial matrix K_{nn}, and

(iii) the diagonal elements in positions (i,i), $1 \leq i \leq p$, have values $s'(v_i)$ which are equal to:

$$\begin{aligned} s'(v_i) &= s(v_i)+1, \quad 1 \leq i \leq p-2 \\ s'(v_{p-1}) &= (s(v_{p-1})+1)\cdot((s(v_p)+1)\cdot e(u)) \\ s'(v_p) &= \frac{1}{e(u)} - 1 \end{aligned}$$

if u is a 1-node, where $e(u) = \sum_{i=1}^{p} \frac{1}{s(v_i)+1}$, and

$$\begin{aligned} s'(v_i) &= s(v_i), \quad 1 \leq i \leq p-2 \\ s'(v_{p-1}) &= s(v_{p-1})\cdot s(v_p)\cdot e(u) \\ s'(v_p) &= \frac{1}{e(u)} \end{aligned}$$

if u is a 0-node, where $e(u) = \sum_{i=1}^{p} \frac{1}{s(v_i)}$.

Thus, expanding in terms of the first $p-1$ rows, we find that the determinant of the matrix K_{nn} becomes

$$\begin{aligned} \det(K_{nn}) &= \left(\prod_{i=1}^{p-1} s'(v_i)\right) \cdot \begin{vmatrix} s'(v_p) & & & & \\ & s(v_{p+1}) & & (-1)_{ji} & \\ & & s(v_{p+2}) & & \\ & (-1)_{ij} & & \ddots & \\ & & & & s(v_{n-1}) \end{vmatrix} \\ &= \left(\prod_{i=1}^{p-1} s'(v_i)\right) \cdot \det(K'), \end{aligned}$$

where K' is an $(n-p)\times(n-p)$ matrix similar to the initial matrix K_{nn}; in fact, it is identical to the submatrix of the initial matrix K_{nn} defined by rows $p, p+1, \ldots, n-1$ and columns $p, p+1, \ldots, n-1$, with the only exception that the value $s'(v_p)$ is different from $s(v_p)$. We note that, the matrix K' corresponds to a cograph G' on $n-p$ vertices, and, thus, it contains at least one strong block (see Lemma 2.1).

Thus, if we assume (in an inductive fashion) that the determinant of the matrix K' can be expressed as the product of appropriate values $s''(v_p)$, $s''(v_{p+1})$, $\ldots$, $s''(v_{n-1})$, then the determinant of the initial matrix K_{nn} is equal to the product of these values multiplied by the product of $s'(v_1), s'(v_2), \ldots, s'(v_{p-1})$, just as the algorithm Spanning_Trees_Number does by using function Update_Replace( ). This establishes the correctness of our algorithm, and, thus, we can state the following result.

Lemma 3.1. *The algorithm Number_Spanning_Trees correctly computes the number of spanning trees of a cograph G.*

3.3 Time and Space Complexity

We next compute the time and space complexity of the algorithm Number_Spanning_Trees. We prove the following lemma.

Lemma 3.2. *The algorithm Number_Spanning_Trees runs in $O(n+m)$ time, where n is the number of vertices and m the number of edges of the input cograph.*

Proof. The construction of a cotree $T_{co}(G)$ of a cograph G on n vertices and m edges can be implemented in $O(n+m)$ time complexity [7]. The computation of the level sets $L_1, L_2, \ldots, L_h$ of the cotree $T_{co}(G)$ in Step 5 of the algorithm Number_Spanning_Trees can be performed in $O(n)$ time, since the cotree $T_{co}(G)$ contains $O(n)$ nodes. Additionally, the function *Update_Replace* is applied on each of the nodes of the cotree $T := T_{co}(G)$, and when it is applied on a node u, it can be executed in $O(|\text{ch}(u)|)$ time, where $|\text{ch}(u)|$ is the cardinality of the set of children of node u in T. Given that the number of the nodes of the tree T is $O(n)$, Step 6 of the algorithm Number_Spanning_Trees requires $\sum_{u \in T} |\text{ch}(u)|$ time. Finally, Step 7 takes $O(n)$ time under the uniform cost criterion, according to which each instruction requires one unit of time and each register requires one unit of space, implying that, no matter how large the numbers are, an arithmetic operation involving k numbers takes $O(k)$ time. Therefore, the algorithm Number_Spanning_Trees takes $O(n+m)$ time. ∎

Remark 3.1. The time complexity of our algorithm is measured according to the uniform cost criterion. Under this criterion each instruction on our model requires one unit of time and each register requires one unit of space. Despite the fact that the arithmetic operations involve arbitrarily large integers, we count each operation as a single step. In our case, the uniform cost is realistic if a single computer word can store an integer as large as n^{n-2}, where n is the number of vertices of a graph (the number of spanning trees of a graph G on n vertices is at most n^{n-2}; the complete graph K_n has n^{n-2} spanning trees). Note however that if the quantity n^{n-2} is larger than what can be stored in one computer word, then even the logarithmic cost criterion (this takes into account the limited size of a real memory word which is logarithmic in the number stored) is somewhat unrealistic, since it assumes that two integers i and j can be multiplied in time $O(\log(i) + \log(j))$, which is not known to be possible (see [1]). □

It is not difficult to see that the space needed by the algorithm Number_Spanning_Trees is $O(n+m)$. Recall that, the cotree of a cograph on n vertices and m edges can also be constructed in linear time and space [7]. Thus, the results of this section are summarized in the following theorem.

Theorem 3.1. *The number of spanning trees of a cograph G on n vertices and m edges can be computed in $O(n+m)$ time and space.*

4 Extending the Approach to other Classes of Graphs

In this section we extend our results to a proper superclass of cographs, namely the P_4-reducible graphs. The class of P_4-reducible graphs was introduced by Jamison and Olariu in [14] as an extension of the class of cographs, and defined as the class of graphs for which no vertex belongs to more than one induced P_4.

A *bull* graph is a graph on five vertices obtained by a P_4 with an additional vertex which is adjacent to the midpoints and non-adjacent to the endpoints of the P_4. The modular decomposition tree of P_4-reducible graphs has a structural property, which is shown by the following result (Theorem 4.2 in [10]):

Theorem 4.1. (Giakoumakis and Vanherpe [10]): *Let G be a graph, $T(G)$ its modular decomposition tree and u an internal 2-node of $T(G)$. The graph G is a P_4-reducible graph iff for every u of $T(G)$, $G(u)$ is either a P_4, or a bull graph. Moreover, the vertices of any P_4 of the graph $G(u)$ are leaf vertices in $T(G)$.*

The above theorem implies that we can easily distinguish the endpoints and the midpoints of every P_4 of a P_4-reducible graph G. Moreover, the two endpoints and the two midpoints of a P_4 in G have the same degrees, denoted by d_s and d_k, respectively.

It is not difficult to see that modular decomposition tree $T(G)$ of a P_4-reducible graph contains at least one strong block; a 0-node, 1-node or 2-node u is called strong if the set $\mathrm{ch}(u)$ contains only leaves in $T(G)$.

Let u be a strong 2-node that forms the strong block $S = \{v_1, v_2, \ldots, v_p\}$. Note that, the strong 2-node u induces either a graph on four vertices (i.e., a P_4), or a graph on five vertices (i.e., a bull graph) in $T(G)$; this is the graph $G(u)$ (see Theorem 4.1). Let $G(u)$ be a bull graph denoted by $v_1v_2v_3v_4v_5$, and let $K_{nn}(2)$ be the $(n-1)\times(n-1)$ matrix obtained from K after deleting its n-th row and column. Substituting the values d_s, d_k and $s(v_i) = d_i$, $5 \le i \le n-1$, the matrix $K_{nn}(2)$ becomes

$$\begin{bmatrix}
d_s & -1 & & & & & & & & \\
-1 & d_k & -1 & & -1 & & & & & \\
 & -1 & d_k & -1 & -1 & & & [-1]_{j'i} & & \\
 & & -1 & d_s & & & & & & \\
 & -1 & -1 & & s(v_5) & & & & & \\
 & & & & & s(v_{p+1}) & & & (-1)_{ji} & \\
 & & & & & & s(v_{p+2}) & & & \\
 & & [-1]_{ij'} & & & & & \ddots & & \\
 & & & & & (-1)_{ij} & & & s(v_{n-1}) &
\end{bmatrix},$$

where, according to the definition of the Kirchhoff matrix, the entries $(-1)_{ij}$ and $(-1)_{ji}$ of the off-diagonal position $(i-p, j-p)$ and $(j-p, i-p)$ of the

matrix $K_{nn}(2)$ are both -1 if the vertices v_i and v_j are adjacent in G and 0 otherwise, $p+1 \leq i,\ j \leq n-1$. The entries $[-1]_{ij'}$ and $[-1]_{j'i}$ of the off-diagonal positions (i,j) and (j,i) of the matrix $K_{nn}(2)$ correspond to 1×5 and 5×1 matrices, respectively, with all their subelements having value -1, if vertices v_i and v_j are adjacent in G, and 0 otherwise, $p+1 \leq i \leq n-1$ and $1 \leq j \leq 5$. Note that, if a vertex $v \in G(u)$ is adjacent to another vertex $v' \in G - G(u)$ then all the vertices in $G(u)$ are adjacent to vertex v', since $G(u)$ is a module in G.

Next, we focus on the computation of the determinant of the matrix $K_{nn}(2)$. We start by focusing on its first and fourth rows and columns. We multiply the first and the fourth columns of the matrix $K_{nn}(2)$ by $1/d_s$ and add them to the second and to the third columns, respectively. Then, we multiply the fourth row of the matrix $K_{nn}(2)$ by -1 and add it to the first row. Finally, we add the first column of the matrix $K_{nn}(2)$ to the fourth column. We point out that after applying these operations to matrix $K_{nn}(2)$ only the diagonal position of the first row have non-zero entries.

We now focus on the second and third rows and columns of the matrix $K_{nn}(2)$. Here, we multiply by -1 the third row of the matrix K_{nn} and add it to the second row. Then, we add the second column of the matrix $K_{nn}(2)$ to the third column, and, thus, only the diagonal positions of the second row have non-zero entries with value $d_k + (d_s - 1)/d_s$. Thus, the first and the second rows have zero elements in the off-diagonal positions.

We apply a similar technique in order to make zero the elements in the off-diagonal positions of the fourth and fifth rows of the matrix $K_{nn}(2)$. We first multiply the third row of matrix $K_{nn}(2)$ by -1 and add it to the fourth row. Then, we multiply the fourth column of matrix by α, where $\alpha = (d_k - (d_s + 1)/d_s)/(d_s + 1)$, and add it to the third column. We also multiply the fourth column of matrix $K_{nn}(2)$ by $-\alpha'$, where $\alpha' = 1/(d_s + 1)$ and add it to the fifth column. Now, only the diagonal position of the fourth row of matrix $K_{nn}(2)$ has non-zero entry.

We continue working in order to make zero the off-diagonal elements of the fifth row of matrix $K_{nn}(2)$. To this end, we first multiply the third row of matrix $K_{nn}(2)$ by -1 and add it to the fifth row, and, then, we multiply the fifth column of matrix $K_{nn}(2)$ by α'', where $\alpha'' = \frac{2+(d_k-(d_s+1)/d_s)\ d_s/(d_s+1)}{s(v_5)+(d_s/(d_s+1))}$, and add it to the third column. Thus, we obtain

$$\begin{aligned}
\det(K_{nn}(2)) &= \prod_{\substack{i=1 \\ i\neq 3}}^{5} s'(v_i) \cdot \begin{vmatrix} s'(v_3) & & & & \\ & s(v_{p+1}) & & (-1)_{ji} & \\ & & s(v_{p+2}) & & \\ & (-1)_{ij} & & \ddots & \\ & & & & s(v_{n-1}) \end{vmatrix} \\
&= \prod_{\substack{i=1 \\ i\neq 3}}^{5} s_2(v_i) \cdot \det(D_{nn}(2)),
\end{aligned}$$

where $D_{nn}(2)$ is a $(n-p) \times (n-p)$ matrix, and the s-labels $s'(v_i)$, $1 \leq i \leq 5$, have values according to following equations:

$$\begin{aligned}
s'(v_1) &= d_s && (3)\\
s'(v_2) &= d_k + \frac{d_s - 1}{d_s} && (4)\\
s'(v_3) &= \frac{d_s}{\gamma} \cdot \frac{s(v_5) \cdot (d_k - \frac{d_s+1}{d_s}) - 2}{s(v_5) + \frac{d_s}{d_s+1}} && (5)\\
s'(v_4) &= \gamma && (6)\\
s'(v_5) &= s(v_5) + \frac{d_s}{d_s + 1} && (7)
\end{aligned}$$

where

$$\gamma = 2 \cdot (d_s + d_k + 1) - (d_s - 1) \cdot \frac{2 + \frac{d_s}{d_s+1} \cdot (d_k - \frac{d_s+1}{d_s})}{s(v_5) + \frac{d_s}{d_s+1}}. \quad (8)$$

Recall that $G(u)$ is a bull graph (or, equivalently, $p = |B| = 5$). Note that, if $G(u)$ is a P_4 (or, equivalently, $p = |B| = 4$) then $D_{nn}(2)$ is a $(n-p) \times (n-p)$ matrix, and the s-labels $s'(v_i)$, $1 \leq i \leq 4$, have values according to following equations:

$$\begin{aligned}
s'(v_1) &= d_s && (9)\\
s'(v_2) &= d_k + \frac{d_s - 1}{d_s} && (10)\\
s'(v_3) &= \frac{d_s}{\gamma} \cdot (d_k - \frac{d_s + 1}{d_s}) && (11)\\
s'(v_4) &= \gamma && (12)
\end{aligned}$$

where

$$\gamma = 2 \cdot (d_s + d_k + 1). \quad (13)$$

In conclusion, matrix $D_{nn}(2)$ can be obtained from matrix $K_{nn}(2)$ by setting value $s'(v_3)$ at position $(3,3)$ and by deleting the first, second and fourth rows and columns of $K_{nn}(2)$, and also by deleting the fifth row and column of $K_{nn}(2)$ in the case where $p = 5$. Thus, the matrices $D_{nn}(2)$ and $K_{nn}(2)$ are of the same form, and $D_{nn}(2)$ represents a tree $T'(G)$ on $n - p$ vertices.

It follows that we can describe a function, similar to function Update_Replace(u, T), in the case where u is a 2-node and T is the modular decomposition tree of a P_4-reducible graph G. Such a function relies basically on the following:

- if $|\text{ch}(u)| = 5$, then update the s-labels of the vertices of $\text{ch}(u)$ based on Eqs. (3–7);
- if $|\text{ch}(u)| = 4$, then update the s-labels of the vertices of $\text{ch}(u)$ based on Eqs. (9–12);
- make vertex v_3 child of node $p(u)$ and delete the vertices v_1, v_2, v_4 and node u from the tree $T(G)$; if $|\text{ch}(u)| = 5$ then also delete the vertex v_5.

Thus, since $T(G)$ can be constructed in linear time in the size of the input graph G [9, 17] we can extend our algorithm Number_Spanning_Trees to P_4-reducible graphs; we contract a strong 2-node of $T(G)$ by applying the appropriate function Update_Replace( ) described previously. Thus, the result of this section is summarized in the following theorem.

Theorem 4.2. *The number of spanning trees of a P_4-reducible graph G on n vertices and m edges can be computed in $O(n + m)$ time and space.*

5 Concluding Remarks

In this paper we propose an approach for computing the number of spanning trees of a cograph or a P_4-reducible graph which takes advantage of the structural properties of the modular decomposition trees of these graphs and yields linear-time algorithms for the problem.

More general classes of cographs and P_4-reducible graphs, such as the classes of P_4-sparse graphs, $P4$-lite graphs, $P4$-tidy graphs, tree-cographs [4], also possess structural and algorithmic properties of their modular decomposition trees. Thus, it is reasonable to ask whether the same approach can also be used for computing the number of spanning trees of these, and other, classes of graphs.

It has been shown that a permutation graph $G[\pi]$ can be transform into a directed acyclic graph and, then, into a rooted tree by exploiting the inversion relation on the elements of the permutation π [19]; note that the permutation graphs are perfect and in fact form a proper superclass of cographs [4, 12]. Based on these results, one can work towards the investigation whether the class of permutation graphs $G[\pi]$ belongs to the family of graphs that have formulas or efficient algorithms regarding the number of their spanning trees. We pose it as an open problem.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] C. Berge, *Graphs and Hypergraphs*, North-Holland, 1973.

[3] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, London, 1974.

[4] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics and Applications, 1999.

[5] B. Bollobás, *Graph Theory, An Introductory Course*, Springer-Verlag, New York, 1979.

[6] T.J.N. Brown, R.B. Mallion, P. Pollak, and A. Roth, Some methods for counting the spanning trees in labelled molecular graphs, examined in relation to certain fullereness, *Discrete Appl. Math.* **67** (1996) 51–66.

[7] D.G. Corneil, Y. Perl, and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **14** (1985) 926–984.

[8] K.-L. Chung and W.-M. Yan, On the number of spanning trees of a multi-complete/star related graph, *Inform. Process. Lett.* **76** (2000) 113–119.

[9] E. Dalhaus, J. Gustedt and R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* **41** (2001) 360–387.

[10] V. Giakoumakis and J-M. Vanherpe, On extended P_4-reducible and P_4-sparse graphs, *Theoretical Comp. Science* **180** (1997) 269–286.

[11] B. Gilbert and W. Myrvold, Maximizing spanning trees in almost complete graphs, *Networks* **30** (1997) 23–30.

[12] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

[13] P.L. Hammer and A.K. Kelmans, Laplacian spectra and spanning trees of threshold graphs, *Discrete Appl. Math.* **65** (1996) 255–273.

[14] B. Jamison and S. Olariu, A linear-time recognition algorithm for P_4-reducible graphs, *Theoret. Comput. Sci.* **145** (1995) 329–344.

[15] A.K. Kelmans and V.M. Chelnokov, A certain polynomial of a graph and graphs with an extremal number of trees, *J. Combin. Theory* (B) **16** (1974) 197–214.

[16] H. Lerchs, On cliques and kernels, Technical Report, Department of Computer Science, University of Toronto, March 1971.

[17] R.M. McConnell and J. Spinrad, Modular decomposition and transitive orientation, *Discrete Math.* **201** (1999) 189–241.

[18] W. Myrvold, K.H. Cheung, L.B. Page, and J.E. Perry, Uniformly-most reliable networks do not always exist, *Networks* **21** (1991) 417–419.

[19] S.D. Nikolopoulos, Coloring permutation graphs in parallel, *Discrete Appl. Math.* **120** (2002) 165–195.

[20] S.D. Nikolopoulos and C. Papadopoulos, The number of spanning trees in K_n-complements of quasi-threshold graphs, *Graphs and Combinatorics* **20** (2004) 383–397.

[21] S.D. Nikolopoulos and P. Rondogiannis, On the number of spanning trees of multi-star related graphs, *Inform. Process. Lett.* **65** (1998) 183–188.

[22] P.V. O'Neil, The number of trees in a certain network, *Notices Amer. Math. Soc.* **10** (1963) 569.

[23] L. Petingi and J. Rodriguez, A new technique for the characterization of graphs with a maximum number of spanning trees, *Discrete Math.* **244** (2002) 351–373.

[24] L. Weinberg, Number of trees in a graph, *Proc. IRE.* **46** (1958) 1954–1955.

[25] W.-M. Yan, W. Myrvold, and K.-L. Chung, A formula for the number of spanning trees of a multi-star related graph, *Inform. Process. Lett.* **68** (1998) 295–298.

[26] X. Yong, Talip, Acenjian, The numbers of spanning trees of the cubic cycle C_n^3 and the quadruple cycle C_n^4, *Discrete Math.* **169** (1997) 293–298.

[27] Y. Zhang, X. Yong, and M.J. Golin, The number of spanning trees in circulant graphs, *Discrete Math.* **223** (2000) 337–350.

[28] Y. Zhang, X. Yong, and M.J. Golin, Chebyshev polynomials and spanning tree formulas for circulant and related graphs, *Discrete Math.* **298** (2005) 334–364.