Counting Spanning Trees using Modular Decomposition[*]

Stavros D. Nikolopoulos[†] Leonidas Palios[†] Charis Papadopoulos[‡]

Abstract

In this paper we present an algorithm for determining the number of spanning trees of a graph G which takes advantage of the structure of the modular decomposition tree of G. Specifically, our algorithm works by contracting the modular decomposition tree of the input graph G 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. In particular, when applied on a $(q, q-4)$-graph for fixed q, a P_4-tidy graph, or a tree-cograph, our algorithm computes the number of its spanning trees in time linear in the size of the graph, where the complexity of arithmetic operations is measured under the uniform-cost criterion. Therefore we give the first linear-time algorithm for the counting problem in the considered graph classes.

Keywords: number of spanning trees, modular decomposition, Kirchhoff matrix tree theorem, linear-time algorithms, $(q, q-4)$-graph, P_4-tidy graph, tree-cograph.

1 Introduction

A *spanning tree* of a connected undirected graph G on n vertices is a connected $(n-1)$-edge subgraph of G. The number of spanning trees of a graph G, also called the *complexity* of G, is an important, well-studied quantity in graph theory, and appears in a number of applications. Most notable application fields are network reliability [27], computing the total resistance along an edge in an electrical network [7], enumerating certain chemical isomers [10], and counting the number of Eulerian circuits in a graph [21]. In particular, counting spanning trees is an essential step in many methods for computing, bounding, and approximating network reliability [11]; in a network modeled by a graph, intercommunication between all nodes of the network implies that the graph must contain a spanning tree and, thus, maximizing the number of spanning trees is a way of maximizing reliability.

Thus, both for theoretical and for practical purposes, we are interested in deriving a formula for computing the number of spanning trees of a graph G, and also of the K_n-complement of G (for any subgraph H of the complete graph K_n, the K_n*-complement* of H, denoted by $K_n - H$, is

[*]A preliminary version of this work was presented at WALCOM 2011.

[†]Department of Computer Science, University of Ioannina, Greece; e-mails: {stavros, palios}@cs.uoi.gr

[‡]Department of Mathematics, University of Ioannina, Greece; e-mail: charis@cs.uoi.gr

defined as the graph obtained from K_n by removing the edges of H; note that, if H has n vertices, then $K_n - H$ coincides with the complement $\overline{H}$ of H). Many cases have been examined depending on the choice of G, such as when G is a labeled molecular graph [10], a circulant graph [1, 37], a complete multipartite graph [36], a multi-star related graph [30], a quasi-threshold graph [2, 28]; see Berge [5] for an exposition of the main results.

The purpose of this paper is to study the general problem of finding the number of spanning trees of an input graph. Traditionally, the number of spanning trees of a graph is computed by means of the classic *Kirchhoff matrix tree theorem* [21], which 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 G. Thus, counting spanning trees reduces to evaluating the determinant of an $((n-1) \times (n-1))$-size matrix, where n is the number of vertices of the input graph. This approach has been used for computing the number of spanning trees of families of graphs (see [2, 5, 19, 30, 36]), but it requires $\Theta(n^{2.376})$ arithmetic operations on matrix entries and $\Theta(n^2)$ space [13]; in fact, the algorithm that achieves this time and space complexity appears to have so large a constant factor that for practical combinatorial computations the naive $O(n^3)$-time algorithm turns out to be the sensible choice. We also mention that in some special classes of graphs, the determinant can be computed in $O(n^{1.5})$ time, using the planar separator theorem [23]. In a few cases, the number of spanning trees of a graph has been computed without the evaluation of a determinant. Colbourn et al. in [12] have proposed an algorithm which runs in $O(n^2)$ time for an n-vertex planar graph. Their algorithm is based on some particular transformations (known as the *delta-wye* technique) that can be applied on planar graphs; unfortunately, it is hard to study such or other kinds of transformations on general graphs (besides planar graphs).

In order to obtain an efficient solution for this problem, we take advantage of the modular decomposition (a form of graph decomposition which associates the graph with its maximal homogeneous sets) of the input graph G and especially the properties of its modular decomposition tree. The usage of modular decomposition has been proposed for solving several optimization problems (see the surveys [20, 26] on modular decomposition). To name a few of them we refer to the problem of computing the treewidth and the minimum fill-in [6]. Also, it has been proposed to obtain efficient algorithms by expressing optimization problems in monadic second-order logic [14]. Other application areas of modular decomposition arise in graph drawing [32] and in biological strategies of proteins [17].

Our algorithm uses the modular decomposition and relies on tree contraction operations which are applied in a systematic fashion from bottom to top in order to shrink the modular decomposition tree of the graph G into a single node, while at the same time certain parameters are appropriately updated; the updating essentially implements transformations on a cofactor of the Kirchhoff Matrix towards evaluating its determinant, yet the fact that we are dealing with modules (that is, any vertex outside the module is adjacent to either all or none of the vertices in the module) allows us to beat the $O(n^{2.376})$ time complexity for many classes of graphs. In the end, the number of spanning trees of G is obtained as the product of n numbers, where n is the number of vertices of G; this multiplication takes $O(n)$ time under the uniform-cost criterion [31]. Our algorithm is easy to implement; its correctness is established by means of the Kirchhoff

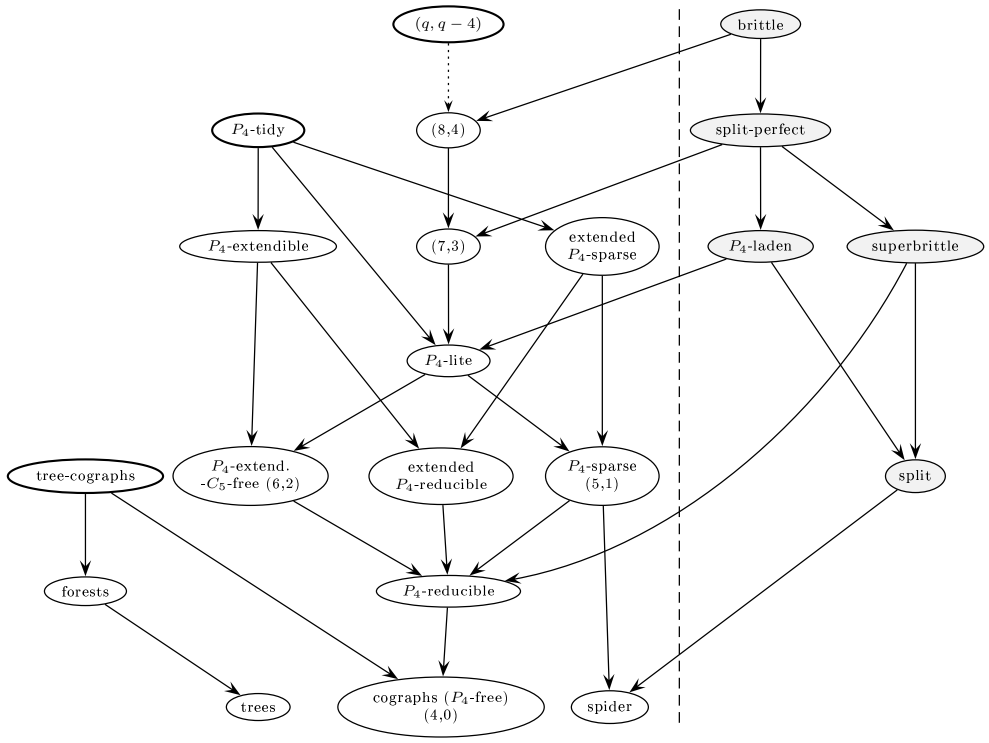

Figure 1: A Hasse diagram of class inclusions. For the classes to the left of the dashed line, the number of spanning trees can be computed in linear time.

matrix tree theorem along with standard techniques from linear algebra and matrix theory.

In particular, for certain classes of graphs, the structure of their modular decomposition trees (and in fact their prime graphs) ensures that each tree node can be processed in time linear in the size of the contracted part of the tree; thus, since the modular decomposition tree of a graph can be constructed in time and space linear in the size of the graph [15, 25, 34], the processing of the entire modular decomposition tree and consequently the number of its spanning trees takes time and space linear in the size of the input graph. Such classes are the classes of tree-cographs, $(q, q-4)$-graphs for fixed q, and P_4-tidy graphs, along with their numerous subclasses, such as, the cographs, the P_4-reducible, the extended P_4-reducible, the P_4-sparse, the P_4-lite, the P_4-extendible, and the extended P_4-sparse graphs. Our findings are summarized in Figure 1.

The paper is organized as follows. In Section 2 we establish the notation and related terminology and we present background results. In Section 3, we describe the algorithm for determining the number of spanning trees of an arbitrary graph. In Section 4 we prove its correctness, while in Section 5 we compute its time and space complexity. In Section 6 we show that the number of spanning trees of some classes of graphs can be computed in linear time. Finally, in Section 7, we conclude the paper and discuss possible extensions.

2 Preliminaries

We consider finite undirected graphs with no self-loops or multiple edges. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the subgraph $G[V(G) - S]$ and by $G - v$ the graph $G[V(G) - \{v\}]$. A *clique* is a set of pairwise adjacent vertices; a *stable set* is a set of pairwise non-adjacent vertices.

The *neighborhood* $N(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to x. The *degree* of a vertex x in the graph G, denoted $d(x)$, is the number of edges incident on x; thus, $d(x) = |N(x)|$. If two vertices x and y are adjacent in G, we say that x *sees* y; otherwise we say that x *misses* y. We extend this notion to vertex sets: $V_i \subseteq V(G)$ sees (misses) $V_j \subseteq V(G)$ if and only if every vertex $x \in V_i$ sees (misses) every vertex $y \in V_j$.

2.1 Modular Decomposition

A subset M of vertices of a graph G is said to be a *module* of 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 are *trivial* modules and whenever G has only trivial modules it is called a *prime graph*. We note that a chordless path on $n \geq 4$ vertices and a chordless cycle on $n \geq 5$ vertices are prime graphs. Furthermore, a module M of G is called *strong*, if there is no module M' such that $M' \cap M \neq \emptyset$, $M \setminus M' \neq \emptyset$, and $M' \setminus M \neq \emptyset$.

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 t corresponds to a strong module, denoted M_t, which is induced by the set of vertices/leaves of the subtree rooted at t. Thus, $T(G)$ represents all the strong modules of G. Each internal node is labeled by either P for *parallel* module, S for *series* module, or N for *neighborhood* module. The module corresponding to a P-node induces a disconnected subgraph of G, that of an S-node induces a connected subgraph of G whose complement is a disconnected subgraph and that of an N-node induces a connected subgraph of G whose complement is also a connected subgraph. Figure 2 shows a graph and its modular decomposition tree.

In particular, let t be an internal node of the modular decomposition tree $T(G)$. If t has children $u_1, u_2, \ldots, u_p$, then we define the *representative graph* G_t of the module M_t as follows:

- $V(G_t) = \{u_1, u_2, \ldots, u_p\}$, and
- $E(G_t) = \{u_i u_j \mid v_i v_j \in E(G),\ v_i \in M_{u_i} \text{ and } v_j \in M_{u_j}\}$.

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_t is isomorphic to the graph induced by a subset of M_t consisting of a single vertex from each maximal submodule of M_t in $T(G)$. Then: (i) if t is a P-node, G_t is an edgeless graph, (ii) if t is an S-node, G_t is a complete graph, and (iii) if t is an N-node, G_t 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

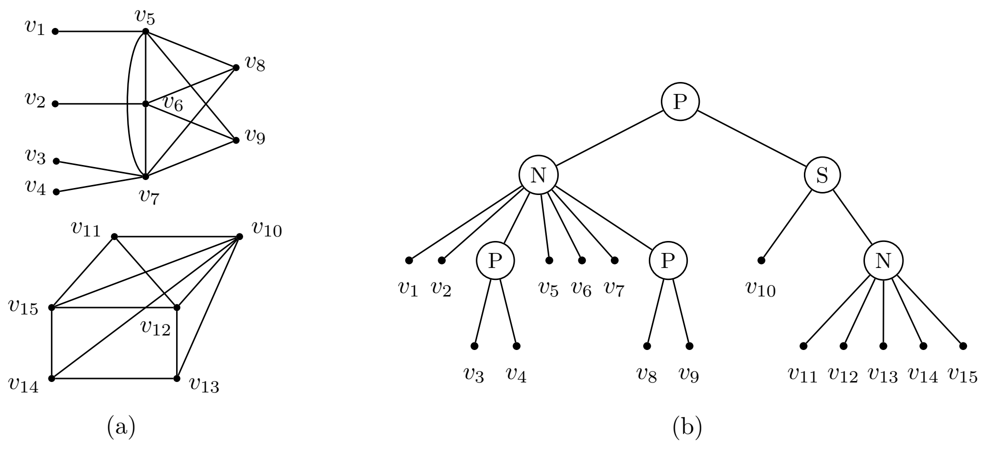

Figure 2: (a) A disconnected graph and (b) its modular decomposition tree.

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 [15, 25, 34]. A graph is called *cograph* if it has no induced path on four vertices. Note that a graph G is a cograph if and only if $T(G)$ contains no N-node, since every prime graph contains an induced path on four vertices (see for e.g., [20]).

2.2 Contractible Subtrees

In general, using modular decomposition for solving problems can be quite challenging. A typical algorithm for exploiting the modular decomposition tree starts with computing the optimal value of each interior node u by using the optimal values of all the children of u depending on the type of the node. Then the optimal value of the root is the optimal value of the problem for the input graph G. Thus, to specify such a modular-decomposition-based algorithm, we only have to describe how to obtain the value for the leaves and which formula to evaluate a P-node, an S-node, and an N-node in terms of the values of all children.

Definition 2.1. Let T be a tree. A subtree rooted at an internal node t of T is a *contractible subtree* iff all the children of t are leaves of T.

Note that every non-empty rooted tree has at least one contractible subtree and, thus, $T(G)$ has at least one contractible subtree. Furthermore the set of leaves of a contractible subtree in $T(G)$ form a module in G, since any two leaves of the set have the same neighborhood outside the set.

2.3 Basic and Non-Basic Graphs

Let $T(G)$ be the modular decomposition of a graph G and let t be an N-node. We will show that if the prime graph G_t belongs to a certain family $\mathcal{F}$ of graphs to be described later then its processing takes time linear in the size of G_t. Next we formally define the family of graphs $\mathcal{F}$.

A *path* in a graph G is a sequence of vertices $v_0v_1\cdots v_k$ such that $v_{i-1}v_i \in E(G)$ for $i = 1, 2, \ldots, k$. A path is called *simple* if none of its vertices occurs more than once. A path (simple path) $v_0v_1\cdots v_k$ is a *cycle* (*simple cycle*) if $v_0v_k \in E(G)$. A simple path (cycle) $v_0v_1\cdots v_k$ is *chordless* if $v_iv_j \notin E(G)$ for any two non-consecutive vertices v_i, v_j in the path (cycle). Throughout the paper, the chordless path (cycle) on k vertices is denoted by P_k (respectively C_k). In particular, a chordless path on 4 vertices is denoted by P_4; in a P_4 $abcd$, the vertices b, c are the *midpoints* and the vertices a, d are the *endpoints* of the P_4.

Let T be a rooted tree. The parent of a node x of T is denoted by $p(x)$, whereas the node set containing the children of x in T is denoted by $ch(x)$. We denote by L_i the node set containing the nodes of the i-th level of T, for each value of i from 0 to the height of the tree T.

A graph is called a *spider* if its vertex set admits a partition into sets S, K, and R such that:

(S1) $|S| = |K| \geq 2$, S is a stable set, and K is a clique;

(S2) the vertex set R sees K and misses S;

(S3) there exists a bijection $f : S \longrightarrow K$ such that exactly one of the following statements holds:
  (i) for each vertex $v \in S$, $N(v) \cap K = \{f(v)\}$;
  (ii) for each vertex $v \in S$, $N(v) \cap K = K - \{f(v)\}$.

The triple (S, K, R) is called the *spider partition.* A graph G is a *prime spider* if G is a spider with vertex partition (S, K, R) and $|R| \leq 1$. For the prime spiders, in order that cases (i) and (ii) in condition S3 are distinguished, they are referred to as the *thin spider* and the *thick spider*, respectively. Note that, the complement of a thin spider is a thick spider and vice versa.

Definition 2.2. The family $\mathcal{F}$ contains thin spiders, prime trees, chordless cycles of length greater than four, and the complements of thin spiders (thick spiders), the complements of prime trees, and the complements of chordless cycles of length greater than four; we call these graphs *basic graphs.* A prime graph which is not in $\mathcal{F}$ is called *non-basic graph.*

Next, we introduce the definitions of the non-basic cost of a graph and of a contractible subtree of the modular decomposition tree which we will need in our algorithm. We recall that evaluating the determinant of an $n \times n$ matrix (and subsequently the number of spanning trees of a graph on n vertices) requires $O(n^{2.376})$ time [13].

Definition 2.3. Let $\mathcal{F}$ be the family of basic graphs, G be a graph, $T(G)$ be its modular decomposition tree, and let $\alpha(G)$ be the set of those N-nodes of $T(G)$ whose representative graphs are not in $\mathcal{F}$. We define the *non-basic cost* of G as the value $\phi(G) = \sum_{t \in \alpha(G)} |V(G_t)|^{2.376} = \sum_{t \in \alpha(G)} |\mathrm{ch}(t)|^{2.376}$, where $\mathrm{ch}(t)$ denotes the set of children of node t in $T(G)$.

Note that the size of the modular decomposition tree is $O(n)$ which implies that for any n-vertex graph G, we have $\phi(G) = O(n^{2.376})$. Furthermore for a cograph we have $\phi(G) = 0$ since the corresponding modular decomposition tree has no N-nodes and $\alpha(G) = \emptyset$.

2.4 Kirchhoff Matrix

For an $n \times n$ matrix Z, the $(n-1)$-st order *minor* μ^i_j is the determinant of the $(n-1) \times (n-1)$ matrix obtained from Z after having deleted row i and column j. The i-th *cofactor* equals μ^i_i. For an undirected graph G on n vertices, let A be its adjacency matrix and D be its degree matrix, i.e., the diagonal matrix with the degrees of the vertices of G in its main diagonal. The *Kirchhoff* *matrix* K for the graph G is the matrix $D - A$. The Kirchhoff 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 G's Kirchhoff matrix.

Theorem 2.4 ((Kirchhoff Matrix Tree Theorem [21])). *For any graph G with matrix 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 Main Idea and the Contraction Process

Let G be a graph on n vertices and let the p vertices $v_1, v_2, \ldots, v_p$ be the leaves of a contractible subtree of $T(G)$. As already explained right after Definition 2.1, the set $\{v_1, v_2, \ldots, v_p\}$ forms a module in G. The Kirchhoff matrix tree theorem (Theorem 2.4) implies that the number of spanning trees of a graph G on n vertices is equal to any of the cofactors of the Kirchhoff matrix K, and thus, to the determinant of the $(n-1) \times (n-1)$ submatrix K' of K formed by the first $n-1$ rows and the first $n-1$ columns. In terms of the module $\{v_1, v_2, \ldots, v_p\}$, the form of matrix K' is as follows:

$$K' = \left[\begin{array}{ccc|cccc}
d(v_1) & & (-1)_{i,j'} & (-1)_{1,p+1} & & & (-1)_{1,n-1} \\
 & \ddots & & \vdots & & (-1)_{i,j} & \vdots \\
(-1)_{j',i} & & d(v_p) & (-1)_{p,p+1} & & & (-1)_{p,n-1} \\
\hline
(-1)_{p+1,1} & \cdots & (-1)_{p+1,p} & d(v_{p+1}) & & & \\
 & & & & \ddots & (-1)_{i',j'} & \\
 & (-1)_{j,i} & & & & d(v_j) & \\
 & & & & (-1)_{j',i'} & & \ddots \\
(-1)_{n-1,1} & \cdots & (-1)_{n-1,p} & & & & d(v_{n-1})
\end{array}\right], \qquad (1)$$

where $d(v_i)$, $1 \leq i \leq n-1$, is the degree of vertex v_i in graph G. The notation $(-1)_{i,j}$ for the off-diagonal (i,j)-elements means that the element is equal to -1 if the vertices v_i and v_j are adjacent in G and are 0 otherwise.

It is clear by Theorem 2.4 that $\tau(G) = \det(K')$; recall that $\tau(G)$ denotes the number of spanning trees of G. We associate each of the $n-1$ vertices with an *s-value* which is equal to the vertex's degree in G, i.e., $s(v_i) = d(v_i)$, where $d(v_i)$, $1 \leq i \leq n-1$, is the degree of vertex v_i in graph G. For the $p \times p$ submatrix of the first p rows and p columns of K' we let r be equal to -1 if vertices v_i and $v_{j'}$ are adjacent in G and 0 otherwise, $1 \leq i, j' \leq p$. Note also that the vertices v_i and $v_{j'}$ are adjacent in G iff they are adjacent in the representative graph G_t, where t is the parent of $v_1, v_2, \ldots v_p$ in $T(G)$. With respect to the values of $s(v_i)$ and r, the matrix K', next denoted as M_0, is formed as follows:

$$
M_0 = \left[\begin{array}{ccc|cccc}
s(v_1) & & (r)_{i,j'} & (c)_{1,p+1} & & & (c)_{1,n-1} \\
& \ddots & & \vdots & & (c)_{i,j} & \vdots \\
(r)_{j',i} & & s(v_p) & (c)_{p,p+1} & & & (c)_{p,n-1} \\
\hline
(c)_{p+1,1} & \cdots & (c)_{p+1,p} & s(v_{p+1}) & & & \\
& & & & \ddots & & (-1)_{i',j'} \\
& (c)_{j,i} & & & & s(v_j) & \\
& & & & (-1)_{j',i'} & & \ddots \\
(c)_{n-1,1} & \cdots & (c)_{n-1,p} & & & & s(v_{n-1})
\end{array}\right], \tag{2}
$$

where the values $(c)_{i,j}$ and their symmetric ones, $1 \leq i \leq p \leq j \leq n-1$, are equal to -1 if the vertices $v_i \in V(G_t)$ and $v_j \in V(G) - V(G_t)$ are adjacent in G and are 0 otherwise.

In order to compute the determinant of matrix M_0 we zero the off-diagonal elements formed by the $p \times p$ submatrix of the first p rows and p columns. Note that the form of this submatrix depends on the structure of the graph G_t (i.e., on the type of the internal node t of $T(G)$). This task is accomplished by standard techniques from linear algebra. Thus we transform the matrix M_0 into another matrix M_1 by making the $p \times p$ submatrix a diagonal matrix, such that $\det(M_0) = \det(M_1)$. For example, these transformations can be applied by the well-known Gauss-Jordan elimination on the $p \times p$ submatrix. We call this process *elimination* of the s-values of the vertices $v_1, v_2, \ldots v_p$ of G_t. It is important to note that the elimination of the s-values applied on the rows and columns of the $p \times p$ submatrix effects the diagonal elements of the positions (i,i) and the off-diagonal elements of the positions (i,j) and (j,i) of the matrix M_0, $1 \leq i \leq p \leq j \leq n-1$. Let $s_1(v_i)$ be the values of the elements of the positions (i,i) and $(c_1)_{i,j}$ and $(c_1)_{j,i}$ be the values of the elements of the positions (i,j) and (j,i) after applying the elimination on the p vertices. Then matrix M_1 is of size $(n-1) \times (n-1)$, similar to the matrix M_0, and has the following form:

$$
M_1 = \left[\begin{array}{ccc|cccc}
s_1(v_1) & & 0 & (c_1)_{1,p+1} & & & (c_1)_{1,n-1} \\
& \ddots & & \vdots & & (c_1)_{i,j} & \vdots \\
0 & & s_1(v_p) & (c_1)_{p,p+1} & & & (c_1)_{p,n-1} \\
\hline
(c_1)_{p+1,1} & \cdots & (c_1)_{p+1,p} & s(v_{p+1}) & & & \\
& & & & \ddots & & (-1)_{i',j'} \\
& (c_1)_{j,i} & & & & s(v_j) & \\
& & & & (-1)_{j',i'} & & \ddots \\
(c_1)_{n-1,1} & \cdots & (c_1)_{n-1,p} & & & & s(v_{n-1})
\end{array}\right]. \tag{3}
$$

Observation 3.1. *There exists a series of manipulations on matrix M_0 of Eq. (2) which transforms it into matrix M_1 of Eq. (3) such that* $\det(M_0) = \det(M_1)$.

The difference of matrix M_1 from matrix M_0 is that of having all the off-diagonal elements of the first p rows and p columns equal to zero. The values of c_1 of the corresponding rows and columns of matrix M_1 depend on the type of transformations that we apply on M_0. Due to the fact that the vertices $v_1, v_2, \ldots, v_p$ form a module in G, it follows that in row i all the elements $(c_1)_{i,j}$, for

$p \leq j \leq n-1$, have the same value, denoted by $c_1(i)$, and similarly in column i all the elements $(c_1)_{j,i}$, for $p \leq j \leq n-1$, have the same value which is equal to $c_1(i)$, since the initial values where both equal to -1. The following lemma proves the essential step of contracting a subtree into its highest indexed vertex v_p, which is applied after the elimination function.

Lemma 3.2. *For the determinant of matrix M_1 of Eq. (3) we have that*

$$\det(M_1) = \left(\prod_{i=1}^{p-2} s_1(v_i)\right) \cdot s_1(v_{p-1}) \cdot s_1(v_p) \cdot \theta \cdot \det(M'),$$

where $\theta = \sum_{i=1}^{p} \frac{c_1^2(i)}{s_1(v_i)}$ and M' is a $(n-p) \times (n-p)$ matrix of the form

$$M' = \begin{bmatrix} s'(v_p) & & & (-1)_{p,j} & & \\ & s(v_{p+1}) & & & & \\ & & \ddots & & (-1)_{i',j'} & \\ (-1)_{j,p} & & & s(v_j) & & \\ & & (-1)_{j',i'} & & \ddots & \\ & & & & & s(v_{n-1}) \end{bmatrix},$$

where $s'(v_p) = \frac{1}{\theta}$.

Proof. In order to compute the determinant of matrix M_1, we zero the off-diagonal elements of its first $p-1$ rows: for each $i = 1, 2, \ldots, p-1$, we multiply row p by $-c_1(i)/c_1(p)$ and we add it to row i, and subsequently, for each $i = 1, 2, \ldots, p-1$, we multiply column i by $\frac{c_1(i) \cdot s_1(v_p)}{c_1(p) \cdot s_1(v_i)}$ and add it to column p. As a result, only the diagonal elements of the first $p-1$ rows have non-zero values, which are equal to $s_1(v_i)$. Moreover, if θ is the value defined in the statement, all the non-zero elements in positions (i,p) of column p have now the same value $\frac{s_1(v_p)}{c_1(p)} \cdot \theta$ and all the non-zero elements in positions (p,i) of row p have now the same value $c_1(p)$, $p+1 \leq i \leq n-1$; these non-zero elements were equal to -1 in the original matrix M_0. In order to make them equal to -1, we divide the entries of row p by $c_1(p)$ and column p by $\frac{s_1(v_p)}{c_1(p)} \cdot \theta$, and then in order to maintain the value of the determinant unchanged, we multiply row $p-1$ by $s_1(v_p) \cdot \theta$. Thus, expanding in terms of the first $p-1$ rows, we find that the determinant of M_1 is

$$\begin{aligned} \det(M_1) &= \left(\prod_{i=1}^{p-1} s'(v_i)\right) \cdot \begin{vmatrix} s'(v_p) & & & (-1)_{p,j} & & \\ & s(v_{p+1}) & & & & \\ & & \ddots & & (-1)_{i',j'} & \\ (-1)_{j,p} & & & s(v_j) & & \\ & & (-1)_{j',i'} & & \ddots & \\ & & & & & s(v_{n-1}) \end{vmatrix} \\ &= \left(\prod_{i=1}^{p-1} s'(v_i)\right) \cdot \det(M'), \end{aligned}$$

where

$$\begin{aligned} s'(v_i) &= s_1(v_i), \qquad 1 \le i \le p-2 \\ s'(v_{p-1}) &= s_1(v_{p-1}) \cdot s_1(v_p) \cdot \theta \\ s'(v_p) &= s_1(v_p) \cdot \frac{1}{c_1(p)} \cdot \frac{c_1(p)}{s_1(v_p) \cdot \theta} = \frac{1}{\theta}. \end{aligned}$$

∎

We call this process *contraction* of the module formed by the vertices $v_1, v_2, \dots v_p$. We observe that the matrix M' is an $(n-p) \times (n-p)$ matrix similar to the original matrix M_0; in fact, it is identical to the submatrix of the original matrix M_0 formed by rows $p, p+1, \dots, n-1$ and columns $p, p+1, \dots, n-1$, with the only exception that the value $s'(v_p)$ is different from $s(v_p)$. Thus, if we assume (in an inductive fashion) that the determinant of the matrix M' can be expressed as the product of appropriate values $s'(v_p), s'(v_{p+1}), \dots, s'(v_{n-1})$, then the determinant of the original matrix M_0 is equal to the product of these values multiplied by the product of $s(v_1), s(v_2), \dots, s(v_{p-1})$. The recursive application of an elimination process of the s-values (see Eq. (3)) and the contraction process of the module (see Lemma 3.2) will be our main tools for presenting the following algorithm in the next section which computes the desired product of the s-values, i.e., the number of spanning trees.

3.1 The Algorithm

In order to compute the number of spanning trees of a graph G on vertices $v_1, v_2, \dots, v_n$, we make use of Theorem 2.4 and Lemma 3.2: we delete an arbitrary vertex $v_n \in V(G)$ and all the edges incident on v_n, we associate each of the remaining vertices with an *s-value* which is initialized to the vertex's degree in G, and we construct the modular decomposition tree of the graph $G - v_n$. Next, in a bottom-up fashion, we process each of the contractible subtrees of the tree and we update the s-values of its vertices/leaves by applying the following two processes:

▷ Elimination process: we eliminate the s-values in order to make diagonal the corresponding submatrix of Eq. (2). During that process we compute the values $s_1(v_i)$ and $c_1(i)$ of Eq. (3).

▷ Contraction process: we contract the subtree into the leaf-node corresponding to its highest-index vertex/leaf by updating the desired values according to Lemma 3.2.

Eventually, the entire tree becomes a single vertex/leaf, and the number of spanning trees of G is then equal to the product of the final values of the s-values of all the vertices in $V(G) - \{v_n\}$.

We note that the elimination process can be achieved by executing an appropriate Gauss-Jordan elimination. We will prove later that if the representative graph G_t of the subtree belongs to the family of basic graphs $\mathcal{F}$ then the elimination can be executed in a more efficient way than the straight forward Gauss-Jordan elimination. Thus before applying the elimination process we need to identify if G_t belongs to $\mathcal{F}$ or not. The algorithm for computing the number of spanning trees of a graph G is given in detail in Fig. 2; the input graph G is assumed to be connected (otherwise it has no spanning trees).

The elimination is applied on a contractible node t and is done by means of two functions, namely, *Handle-Basic*, in the case where G_t belongs to one of the graph classes of the family of

Spanning_Trees-Number

Input: A connected graph G on n vertices $v_1, v_2, \ldots, v_n$ and m edges.

Output: The number of spanning trees $\tau(G)$ of the graph G.

```
1. for every vertex $v_i$, $1 \leq i \leq n-1$, do
           compute its degree $d(v_i)$ in $G$;
       $s(v_i) \leftarrow d(v_i)$;

2. $T \longleftarrow$ the modular decomposition tree $T$ of the graph $G - v_n$;

3. Compute the node sets $L_0, L_1, \ldots, L_h$ of the levels $0, 1, \ldots, h$ of $T$;

4. for $i = h-1$ down to 0 do    {the subtree rooted at t is contractible}
       for every internal node $t \in L_i$ do    {update the s-values of the children of t}
           4.1 if the representative graph $G_t$ belongs to the family $\mathcal{F}$     } {elimination}
               then $T \leftarrow$ Handle-Basic$(t, T)$;                                  }
               else $T \leftarrow$ Handle-NonBasic$(t, T)$;                               }
           4.2 Compute the following value: $\theta \leftarrow \sum_{i=1}^{p} \frac{c(i)^2}{s(v_i)}$;   }
           4.3 Update the s-values of vertices $v_{p-1}$ and $v_p$ as follows:                      }
               $s(v_{p-1}) \leftarrow s(v_{p-1}) \cdot s(v_p) \cdot \theta$;   $s(v_p) \leftarrow \frac{1}{\theta}$;   } {contraction}
           4.4 Replace the subtree rooted at node $t$ by the leaf-node                               }
               associated with vertex $v_p$;                                                         }

5. $\tau(G) \leftarrow \prod_{i=1}^{n-1} s(v_i)$;
```

Figure 2: Algorithm *Spanning_Trees-Number.*

basic graphs, and *Handle-NonBasic*, otherwise. The first function handles basic graphs and is described in Section 4. The second function is basically a well-known Gauss-Jordan elimination applied on the Kirchhoff matrix of a graph that is not basic. Below we give in details function Handle-NonBasic.

In order to apply such a function we use an array $B[\,,\,]$ of size $p \times p$. We note also that during the elimination process we keep track of the values $c(i)$ as described in Eq. (3). At the end we update the proper s-values of the corresponding vertices and the values of $c(i)$ which are needed in the contraction process. The formal description of the function Handle-NonBasic is given below:

Function **Handle-NonBasic**(t, T)

```
1. Construct the $p \times p$ Kirchhoff matrix $B$ of the graph $G_t$, where $v_1, v_2, \ldots, v_p$ are the vertices/children of node $t$;
   for $i = 1, 2, \ldots, p$ do
       $B[i,i] \leftarrow s(v_i)$;
       $c(i) \leftarrow -1$;

2. for $i = 1, 2, \ldots, p-1$ do        {Gauss-Jordan elimination}
     2.1 for $j = i+1, i+2, \ldots, p$ do
```

```
        $r \leftarrow B[j,i]/B[i,i]$ ;
        $c(j) \leftarrow c(j) - r \cdot c(i)$ ;
        for  $k = i, i+1, \ldots, p$  do
            $B[j,k] \leftarrow B[j,k] - r \cdot B[i,k]$ ;
```

3. Update the s-values of vertices $v_1, v_2, \ldots, v_p$ and the values of $b(i)$ as follows:

```
    for  $i = 1, 2, \ldots, p$  do
        $s(v_i) \leftarrow B[i,i]$ ;
```

4 Processing Edgeless and Basic Graphs

As described in the previous section, in order to compute the determinant of matrix M_0 we zero the off-diagonal elements formed by the $p \times p$ submatrix of the first p rows and p columns. This task is accomplished by the elimination process of the s-values of $v_1, v_2, \ldots, v_p$. Note that, the structure of the given submatrix depends on the type of the graph G_t.

- Let G_t be an edgeless graph on p vertices (induced by the P-node t). Then the $p \times p$ submatrix is diagonal since there are no edges in G_t. We call such a function Eliminate_Edgeless which is responsible for assigning the p values of $c(i)$ equal to -1; note that the s-values do not need to be changed. The case of G_t associated with an S-node (complete graph) is handled in the case of a complement of a P-node.
- Let G_t be a tree. In [28], a determinant-based formula was shown in order to compute the number of spanning trees of the graph $K_n - G$, where G is a tree. Function Eliminate_Tree is based on a similar approach; for a detailed proof, see [28].
- Let G_t be a chordless cycle graph. In this case, the form of the $p \times p$ submatrix is similar to a tridiagonal symmetric matrix. A *tridiagonal* matrix is a square matrix with nonzero elements only on the main diagonal and on the first diagonals below and above the main diagonal. The inversion of a symmetric tridiagonal matrix requires only $O(n)$ transformations which are performed by a function which we call Eliminate-Cycle.
- Let G_t be a thin spider. This case is rather complicated, even though it is based on standard matrix operations. We establish these transformations through function Eliminate-Spider.
- Let G_t be the complement of a basic graph. In this case we refer to function Eliminate-Complement.

Let t be an internal node of $T(G)$ and $G_t \in \mathcal{F}$. First, we need to recognize, in which of the classes of the family $\mathcal{F}$ the graph G_t belongs. Function Handle-Basic settles this case and is responsible to apply the corresponding elimination process. More formally,

- if G_t is an edgeless graph then $T \leftarrow$ *Eliminate-Edgeless*$(t, T, -1)$;
- if G_t is a thin spider then $T \leftarrow$ *Eliminate-Spider*$(t, T, -1)$;
- if G_t is a prime tree then $T \leftarrow$ *Eliminate-Tree*$(t, T, -1)$;

- if G_t is a chordless cycle then $T \leftarrow$ *Eliminate-Cycle*$(t, T, -1)$;
- if G_t is the complement of an edgeless graph, a thin spider, a prime tree, or a chordless cycle then $T \leftarrow$ *Eliminate-Complement*(t, T).

Note that the parameter -1 of the functions for the elimination of an edgeless graph, a thin spider, a prime tree, or a chordless cycle is to signal that the input graph is edgeless, a thin spider, a prime tree, or a chordless cycle and not their complements. If these functions are applied on the complements of an edgeless graph, a thin spider, a prime tree, or a chordless cycle, then this parameter is 1 (see function Eliminate-Complement).

In what follows, we give in details the corresponding functions that handle edgeless graphs and basic graphs. We begin with the description of edgeless graphs and complements of basic graphs, since those functions are needed in a succeeding case.

4.1 Processing Edgeless Graphs and Complements of Basic Graphs

Let t be a contractible node and assume that the vertices/children of t are $v_1, v_2, \ldots, v_p$. As already explained if t is a P-node then G_t is an edgeless graph and the matrix M_0 in Eq. (2) is already of the form of matrix M_1 in Eq. (3). Thus the s-values of the p vertices are unaffected and we only need to set the values of $c(i)$ to -1. Function Eliminate-Edgeless is responsible for this simple computation, as shown below.

Function **Eliminate-Edgeless**(t, T, r)

```
1. for  $i = 1, 2, \ldots, p$  do
        $c(i) \leftarrow -1$;
```

Now let us assume that t is an S-node or an N-node such that $\overline{G_t}$ is an edgeless graph, a thin spider, a tree, or a chordless cycle. In this case, the matrix M_0 has the form of Eq. (2). We add a new first row and column to the matrix M_0. The entries of the first row $(1, i)$, $2 \le i \le p$, are equal to -1, except its diagonal element which has value 1, and all the other entries of the new row and column are equal to 0. This row and column augmentation preserves the value of the determinant of matrix M_0 which has the following form:

$$M_0 = \left[\begin{array}{cccc|cccc}
1 & r & \cdots & r & 0 & & \cdots & 0 \\
0 & s(v_1) & & (r)_{i,j'} & (c)_{1,p+1} & & & (c)_{1,n-1} \\
\vdots & & \ddots & & \vdots & & (c)_{i,j} & \vdots \\
0 & (r)_{j',i} & & s(v_p) & (c)_{p,p+1} & & & (c)_{p,n-1} \\
\hline
0 & (c)_{p+1,1} & \cdots & (c)_{p+1,p} & s(v_{p+1}) & & & \\
 & & & & & \ddots & & (-1)_{i',j'} \\
\vdots & & (c)_{j,i} & & & & s(v_j) & \\
 & & & & & (-1)_{j',i'} & & \ddots \\
0 & (c)_{n-1,1} & \cdots & (c)_{n-1,p} & & & & s(v_{n-1})
\end{array}\right],$$

where the values $(c)_{i,j}$ and their symmetric ones, $1 \leq i \leq p \leq j \leq n-1$, are equal to -1 if the vertices $v_i \in V(G_t)$ and $v_j \in V(G) - V(G_t)$ are adjacent in G and are 0 otherwise.

In order to compute the determinant $\det(M_0)$, we multiply the first row by -1 and add it to the next p rows. Now the $(r)_{i,j'}$ entries of the initial matrix, will have value 0, if they were -1 in the initial matrix, and additionally they will have value 1, if they were 0 in the initial matrix. Then, the determinant of matrix M_0 becomes as follows:

$$\det(M_0) = \left|\begin{array}{cccc:ccc}
1 & b(1) & \cdots & b(p) & 0 & \cdots & 0 \\
b(1) & s(v_1)-r & & (\overline{-r})_{i,j'} & (c)_{1,p+1} & & (c)_{1,n-1} \\
\vdots & & \ddots & & \vdots & (c)_{i,j} & \vdots \\
b(p) & (\overline{-r})_{j',i} & & s(v_p)-r & (c)_{p,p+1} & & (c)_{p,n-1} \\
\hdashline
0 & (c)_{p+1,1} & \cdots & (c)_{p+1,p} & s(v_{p+1}) & & \\
 & & & & \ddots & (-1)_{i',j'} & \\
\vdots & & (c)_{j,i} & & & s(v_j) & \\
 & & & & (-1)_{j',i'} & & \ddots \\
0 & (c)_{n-1,1} & \cdots & (c)_{n-1,p} & & & s(v_{n-1})
\end{array}\right|,$$

where the entries $(\overline{-r})_{i,j'}$ and $(\overline{-r})_{j',i}$ of the off-diagonal position (i',j) and (j,i') of the matrix M_0 are both 0 if the vertices $v_{i'}$ and v_j are adjacent in G_t (or else, are 1 if the vertices $v_{i'}$ and v_j are adjacent in $\overline{G_t}$) and are 1 otherwise, $1 \leq i,\ j' \leq p$. The non-zero values of the first row and column except the diagonal element are equal to $b(i)$ where $b(i) = -1$, for $1 \leq i \leq p$. We observe that the $p \times p$ submatrix formed by the $2, 3, \ldots, p+1$ row and column of M_0 is similar to the one corresponding to the basic graph $\overline{G_t}$. The basic graph $\overline{G_t}$ can be viewed as a basic graph with two additional differences:

- the initial s-values are $s'(v_i) = s(v_i) + 1$, $1 \leq i \leq p$;
- if two vertices v_i and v_j, $0 \leq i, j \leq p$ are adjacent in $\overline{G_t}$, then the corresponding entry of the $(p+1) \times (p+1)$ submatrix is equal to 1 (recall, that its initial value would be -1); thus, the corresponding functions are applied by the parameter r, which is -1 if G_t is a basic graph and which is 1 if the complement of G_t is a basic graph.

Keeping these constraints, we can apply the same operations to that we have applied for the basic graph $\overline{G_t}$. Such operations affect the s-values of the vertices $v_1, v_2, \ldots v_p$ and the s-values become $s'(v_i)$ while the corresponding values $c(i)$ and $b(i)$ become $c'(i)$ and $b'(i)$. Then the determinant $\det(M_0)$ is equal to:

$$
\det(M_0) = \left| \begin{array}{cccc|cccc}
1 & b'(1) & \cdots & b'(p) & 0 & & \cdots & 0 \\
b'(1) & s'(v_1) & & 0 & (c')_{1,p+1} & & & (c')_{1,n-1} \\
\vdots & & \ddots & & \vdots & & (c')_{i,j} & \vdots \\
b'(p) & 0 & & s'(v_p) & (c')_{p,p+1} & & & (c')_{p,n-1} \\
\hline
0 & (c')_{p+1,1} & \cdots & (c')_{p+1,p} & s(v_{p+1}) & & & \\
 & & & & & \ddots & (-1)_{i',j'} & \\
\vdots & & (c')_{j,i} & & & & s(v_j) & \\
 & & & & & (-1)_{j',i'} & & \ddots \\
0 & (c')_{n-1,1} & \cdots & (c')_{n-1,p} & & & & s(v_{n-1})
\end{array} \right| .
$$

In order to change the values of the off-diagonal elements of the first row into zero, we multiply rows $2, 3 \ldots p+1$ by $-b'(i)/s'(v_i)$ and add them to the first row. This operation will effect the values of the positions $(1,1)$ and $(1,p+2), (1,p+3), \ldots, (1,n-1)$. More specifically their corresponding values become: $1 - \sum_{i=1}^{p} \frac{c'(i)b'(i)}{s'(v_i)}$ and $-\sum_{i=1}^{p} \frac{c'(i)b'(i)}{s'(v_i)}$. Similarly in order to make zero the off-diagonal elements of the first column we multiply columns $2, 3 \ldots p+1$ by $-b'(i)/s'(v_i)$ and add them to the first row; note that the second operation will not effect the value of position $(1,1)$ but it will change the values of the positions $(p+2,1), (p+3,1), \ldots, (n-1,1)$. Finally the determinant of matrix M_0 becomes:

$$
\det(M_0) = \left| \begin{array}{cccc|cccc}
s'(v_0) & & & 0 & (c')_{0,p+1} & & & (c')_{0,n-1} \\
 & s'(v_1) & & & (c')_{1,p+1} & & & (c')_{1,n-1} \\
 & & \ddots & & \vdots & & (c')_{i,j} & \vdots \\
0 & & & s'(v_p) & (c')_{p,p+1} & & & (c')_{p,n-1} \\
\hline
(c')_{p+1,0} & (c')_{p+1,1} & \cdots & (c')_{p+1,p} & s(v_{p+1}) & & & \\
 & & & & & \ddots & (-1)_{i',j'} & \\
 & & (c')_{j,i} & & & & s(v_j) & \\
 & & & & & (-1)_{j',i'} & & \ddots \\
(c')_{p+1,0} & (c')_{n-1,1} & \cdots & (c')_{n-1,p} & & & & s(v_{n-1})
\end{array} \right| ,
$$

where the value $c'(0) = -\sum_{i=1}^{p} \frac{c'(i)b'(i)}{s'(v_i)}$, and the value of the position $(1,1)$ is equal to $s'(v_0) = 1 - \sum_{i=1}^{p} \frac{c'(i)b(i)}{s'(v_i)}$. Thus we transform the $p \times p$ submatrix into a diagonal $(p+1) \times (p+1)$ submatrix by applying the corresponding elimination function for $\overline{G_t}$. Note that in all cases we have $b'(i) = c'(i)$, since they are both initialized to -1. Below we describe in details Function Eliminate-Complement by keeping the values of $b'(i)$ because the elimination of a thin spider presented in the next section is based on these values.

Function **Eliminate-Complement**(t, T)

```
1. for every vertex $v_i$, $1 \leq i \leq p$, do
       $s(v_i) \leftarrow s(v_i) + 1$;
   for $i = 1, 2, \ldots, p$ do
       $b(i) \leftarrow -1$;
```

2. Depending on the type of the complement of the representative graph G_t call the appropriate function:
   $\overline{G}_t$ is edgeless: $T \leftarrow$ *Eliminate-Edgeless*$(t, T, 1)$;
   $\overline{G}_t$ is a thin spider: $T \leftarrow$ *Eliminate-Spider*$(t, T, 1)$;
   $\overline{G}_t$ is a tree: $T \leftarrow$ *Eliminate-Tree*$(t, T, 1)$;
   $\overline{G}_t$ is a chordless prime cycle: $T \leftarrow$ *Eliminate-Cycle*$(t, T, 1)$;

3. **for** $i = 1, 2, \ldots, p$ **do**
   $b(i) \leftarrow c(i)$;

4. Increase the number of vertices p by one and add a dummy vertex v_0 with the corresponding values:
   $$s(v_0) \leftarrow 1 - \sum_{i=1}^{p} \frac{c(i)b(i)}{s(v_i)}; \qquad c(0) \leftarrow - \sum_{i=1}^{p} \frac{c(i)b(i)}{s(v_i)};$$

4.2 Processing Spiders

Let t be an N-node such that the subtree rooted at t is contractible and the representative graph G_t is a thin spider. Then, if we assume without loss of generality that the vertices/children of t are $v_1, v_2, \ldots, v_p$, they can be partitioned into sets $S = \{v_1, v_2, \ldots, v_k\}$, $K = \{v_{k+1}, v_{k+2}, \ldots, v_{2k}\}$, and R which is either empty (in this case, $p = 2k$) or $R = \{v_{2k+1}\}$ (then, $p = 2k + 1$). Let us suppose for the time being that $R \neq \emptyset$; then, the matrix M_0 is:

$$M_0 = \left[\begin{array}{cccc:cccc:c:c}
s(v_1) & 0 & & 0 & r & & & & 0 & \\
0 & s(v_2) & & & & r & & & 0 & \\
 & & \ddots & & & & \ddots & & \vdots & \\
0 & & & s(v_k) & & & & r & 0 & \\
\hdashline
r & & & & s(v_{k+1}) & r & \cdots & r & r & Z_1 \\
 & r & & & r & s(v_{k+2}) & \cdots & r & r & \\
 & & \ddots & & \vdots & \vdots & \ddots & \vdots & \vdots & \\
 & & & r & r & r & \cdots & s(v_{2k}) & r & \\
\hdashline
0 & 0 & \cdots & 0 & r & r & \cdots & r & s(v_{2k+1}) & \\
\hdashline
\multicolumn{9}{c:}{Z_2} & Z_0
\end{array}\right],$$

where the parameter r is equal to -1 or 1 reflecting the adjacencies in a thin or thick spider, respectively; recall that the complement of a thin spider is a thick spider. The elements of the matrices Z_1 and Z_2 are equal to -1 and 0 depending on whether the corresponding vertices are adjacent in the graph or not, and the matrix Z_0 is an $(n-p-1) \times (n-p-1)$ submatrix of the form

$$Z_0 = \begin{bmatrix}
s(v_{p+1}) & & & & \\
 & \ddots & & (-1)_{j',i} & \\
 & & s(v_i) & & \\
 & (-1)_{i,j'} & & \ddots & \\
 & & & & s(v_{n-1})
\end{bmatrix}.$$

It is important to note that, because the spider is a module, the columns of the submatrix Z_1 are identical, and so are the rows of the submatrix Z_2. In order to compute the determinant $\det(M_0)$ of matrix M_0, we do the following: We zero the off-diagonal entries in the first k rows and columns of matrix M_0. In order to do that,

- ▷ we multiply column j, for $1 \leq j \leq k$, by $-r/s(v_j)$, and add it to column $k+j$, respectively;
- ▷ we multiply row j, for $1 \leq j \leq k$, by $-r/s(v_j)$, and add it to row $k+j$, respectively.

We note that after the above operations the diagonal entries (i,i), for $k+1 \leq i \leq 2k$, of the matrix M_0 have values $s(v_i) - \frac{r^2}{s(v_{i-k})}$. Since the value r is equal to -1 or 1, we have that $r^2 = 1$. All the off-diagonal elements of the (i,j) positions and their symmetric ones, for $k+1 \leq i \leq 2k$ and $p+1 \leq j \leq n-1$, have values

$$c'(i) = c(i) + \frac{r \cdot c(i-k)}{s(v_{i-k})}.$$

The above operations result on a new matrix Q such that $\det(Q) = \det(M_0)$ which has the following form:

$$Q = \left[\begin{array}{cccc:cccccc}
s(v_1) & & & & & & & & & (c)_{1,j} \\
& s(v_2) & & & & & & & & (c)_{2,j} \\
& & \ddots & & & & & & & \vdots \\
& & & s(v_k) & & & & & & (c)_{k,j} \\
\hdashline
& & & & s(v_{k+1}) - \frac{1}{s(v_1)} & r & \cdots & r & r & (c')_{k+1,j} \\
& & & & r & s(v_{k+2}) - \frac{1}{s(v_2)} & \cdots & r & r & (c')_{k+2,j} \\
& & & & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
& & & & r & r & \cdots & s(v_{2k}) - \frac{1}{s(v_k)} & r & (c')_{2k,j} \\
& & & & r & r & \cdots & r & s(v_{2k+1}) & (c)_{2k+1,j} \\
(c)_{j,1} & (c)_{j,2} & \cdots & (c)_{j,k} & (c')_{j,k+1} & (c')_{j,k+2} & \cdots & (c')_{j,2k} & (c)_{j,2k+1} & Z_0
\end{array}\right].$$

At that point we observe that the $(k+1) \times (k+1)$ submatrix formed by the $k+1, k+2, \ldots, 2k+1$ rows and columns of Q is obtained by a complete graph on $k+1$ vertices. Thus we apply the corresponding elimination process for an S-node (when G_t is a complete graph). Notice that any transformation applied on the specific submatrix in order to make it diagonal will not change the values of the elements of the positions (i,j) and their symmetric ones, for $1 \leq i \leq k$ and $k+1 \leq j \leq 2k+1$. This observation arises from the fact that these values are equal to 0 in the matrix Q. More specifically this elimination process changes the values of the diagonal positions, i.e., the s-values of $v_{k+1}, v_{k+2}, \ldots, v_{2k+1}$ and the values of the positions $(k+1,j), (k+2,j), \ldots, (2k+1,j)$ and their symmetric ones, for $2k+1 < j \leq n-1$, i.e., the values of $c'(i)$, $k+1 \leq i \leq 2k+1$.

Recall that the elimination process for a complete graph is achieved through the complement of an edgeless graph (see the previous section). Thus we apply the corresponding function for an edgeless graph, i.e., function Eliminate-Edgeless, with the appropriate values being updated as described in function Eliminate-Complement. We note also that function Eliminate-Edgeless does not change the s-values of the corresponding vertices, since the corresponding submatrix is diagonal.

The function Eliminate-Spider(t, T, r) first computes the spider partition (S, K, R) of the graph G_t if $r = -1$ (in this case G_t is a thin spider), otherwise, it computes the spider partition of the complement of the graph G_t (in this case G_t is a thick spider). Then it updates the s-values of the vertices of the set K and the corresponding values of $c(i)$. Next it assigns a value ℓ according to whether or not R is an empty set. Finally using the value ℓ it updates the s-values of the vertices of $K \cup R$ and the appropriate values of $c(i)$ according to the steps described in function Eliminate-Complement. More precisely, function Eliminate-Spider-N_node is applied as follows.

Function **Eliminate-Spider**(t, T, r)

```
1. If $r = -1$ then compute the sets $S$, $K$, and $R$ of the graph $G_t$;
   otherwise, if $r = 1$ compute these sets of the graph $\overline{G_t}$;
   let $S = \{v_1, v_2, \ldots, v_k\}$ and $K = \{v_{k+1}, \ldots, v_{2k}\}$;  (note that if $R \neq \emptyset$ then $R$ contains
   for $i = 1, 2, \ldots, 2k$ do                                         only one vertex, i.e., $R = \{v_{2k+1}\}$)
       $c(i) \leftarrow -1$;

2. Update the $s$-values of the vertices in $K$ as follows:
       for every vertex $v_i \in K$ do
           $s(v_i) \leftarrow s(v_i) - \frac{1}{s(v_{i-k})}$;      $c(i) \leftarrow c(i) + \frac{r \cdot c(i-k)}{s(v_{i-k})}$;

3. if $R \neq \emptyset$ then   {*R contains only one vertex, i.e., $R = \{v_{2k+1}\}$*}
       $\ell \leftarrow 2k + 1$;
   else  {$R = \emptyset$}
       $\ell \leftarrow 2k$;

4. for every vertex $v_i \in K \cup R$ do
       $s(v_i) \leftarrow s(v_i) + 1$;
       $b(i) \leftarrow -1$;

5. Increase by one the number of vertices $p$ and add a dummy vertex $v_0$ with the corresponding
   values:
       $s(v_0) \leftarrow 1 - \sum_{i=k+1}^{\ell} \frac{c(i)b(i)}{s(v_i)}$;   $c(0) \leftarrow - \sum_{i=k+1}^{\ell} \frac{c(i)b(i)}{s(v_i)}$;
```

4.3 Processing Trees

Let G_t be a prime tree. Based on the level sets of the prime tree we compute in a bottom-up fashion the s-values of the vertices. Each s-value of a vertex is computed according to the s-values of its children, while certain parameters are updated in similar way. We mention here that this technique is based on a recursive formula given in [28] (see Theorem 3.1 in [28] for more details).

Function **Eliminate-Tree**(t, T, r)

```
1. If $r = -1$ then compute the node sets $L_0, L_1, \ldots, L_h$ of the levels $0, 1, \ldots, h$ of the tree $G_t$,
   otherwise, if $r = 1$ compute these sets of the graph $\overline{G_t}$;
   for $i = 1, 2, \ldots, p$ do
       $c(i) \leftarrow -1$;
```

2. In a bottom-up fashion and according to the set of children of each node in the tree, update the s-values of vertices $v_1, v_2, \ldots, v_p$ as follows:

```
    for $i = 1, 2, \ldots, p$ do
        $s(v_i) \leftarrow s(v_i) - \sum_{j \in \mathrm{ch}(i)} \frac{1}{s(v_j)}$;
        $c(i) \leftarrow c(i) - r \cdot \sum_{j \in \mathrm{ch}(i)} \frac{1}{s(v_j) \cdot c(j)}$;
```

4.4 Processing Cycles

Let G_t be a chordless cycle and let $v_1 v_2 \cdots v_p$ be the sequence of the vertices of G_t. The computation of the s-values of the vertices is applied in a sequential fashion. Each s-value of a vertex is computed according to the s-value of the previous vertex in the sequence of the cycle. At the same time, the s-value of the highest-indexed vertex is computed.

Function **Eliminate-Cycle**(t, T, r)

```
1. If $r = -1$ then compute the sequence of the vertices such that $v_1 v_2 \cdots v_p$ forms a chordless
   cycle in the graph $G_t$, otherwise, if $r = 1$ compute such a sequence in the graph $\overline{G_t}$;
   $e \leftarrow r$;
   for $i = 1, 2, \ldots, p$ do
       $c(i) \leftarrow -1$;

2. Update the $s$-values of vertices $v_1, v_2, \ldots, v_{p-1}$ as follows:
       for $i = 2, 3 \ldots, p-1$ do        {elimination of a nearly tridiagonal matrix}
           $s(v_i) \leftarrow s(v_i) - \frac{r^2}{s(v_{i-1})}$;
           $c(i) \leftarrow c(i) - r \cdot \frac{c(i-1)}{s(v_{i-1})}$;
           $s(v_p) \leftarrow s(v_p) - \frac{e^2}{s(v_{i-1})}$;
           $c(p) \leftarrow c(p) - \frac{e \cdot c(i-1)}{s(v_{i-1})}$;
           $e \leftarrow \frac{e}{s(v_{i-1})}$;

3. Update the $s$-value of vertex $v_p$ as follows:
   $e \leftarrow e + r$;
   $s(v_p) \leftarrow s(v_p) - \frac{e^2}{s(v_{p-1})}$;
   $c(p) \leftarrow c(p) - \frac{e \cdot c(p-1)}{s(v_{p-1})}$;
```

The processing on Steps 2 and 3 can be viewed as an elimination of a symmetric nearly tridiagonal matrix of size $p \times p$. Recall that a tridiagonal matrix is a square matrix with nonzero elements only on the main diagonal and on the first diagonals below and above the main diagonal. A nearly tridiagonal matrix is a tridiagonal matrix with two additional nonzero elements: the diagonal elements of the matrix have value $s(v_i)$, the upper-diagonal and lower-diagonal elements have value r and in positions $(1, p)$ and $(p, 1)$ it has value e. Thus, Steps 2 and 3 transform a nearly tridiagonal matrix into a diagonal matrix, while certain parameters c_i being updated during the transformations.

5 Running Time

We next compute the time complexity of the algorithm.

Lemma 5.1. *The algorithm Spanning_Trees-Number runs in $O(n + m + \phi(G))$ time, where n is the number of vertices, m is the number of edges, and $\phi(G)$ is the non-basic cost of the input graph G.*

Proof. Step 1 of the algorithm Spanning_Trees-Number clearly takes $O(n+m)$ time and so does the construction of the modular decomposition tree $T(G)$ of the graph G [15, 25, 34]. The computation of the level sets $L_0, L_1, \ldots, L_{h-1}$ of the tree $T(G)$ in Step 3 can be performed in $O(n)$ time, since the tree $T(G)$ contains $O(n)$ nodes. Additionally, note that exactly one of the eliminate functions is applied on each of the nodes of $T(G)$. Observe that functions Eliminate-Tree and Eliminate-Cycle require $O(|\text{ch}(t)|)$ time, since the number of the nodes are $O(|\text{ch}(t)|)$. When the function Eliminate-Spider is applied on t with representative graph a prime spider G_t, we must compute the sets S, K, and R of G_t (this can be easily done in $O(|V(G_t)| + |E(G_t)|)$ time after having computed the degrees of the vertices of the spider; note that, if a graph is a spider with partition (S, K, R), then for every choice of v, u, and r in S, K, and R respectively, $d(v) < d(r) < d(u)$), update the s-values (this takes $O(|\text{ch}(t)|)$ time), and update the modular decomposition tree (this also takes $O(|\text{ch}(t)|)$ time). Lastly, when the function Handle-Non-Basic is applied on node t, it takes $O(|V(G_t)|^{2.376})$ time, so that the execution of this function on all the non-basic N-nodes of the tree $T(G)$ requires a total of $O(\phi(G))$ time, where $\phi(G)$ is the non-basic cost of G.

The construction of G_t on a node t takes $O(|V(G_t)|+|E(G_t)|)$ time, since node t is contractible, i.e. all the children of t are leaves in $T(G)$. We mention here that G_t and its complement are connected graphs, since node t is an N-node in $T(G)$. In Step 4.1 and in function Eliminate-Complement we need to recognize whether a graph G_t is a thin spider, a prime tree, a chordless cycle or the complement of such a graph. As mentioned, for a spider graph, we can compute its spider partition in $O(|V(G_t)| + |E(G_t)|)$ time. Then, checking the degrees of the vertices in S is sufficient to distinguish a thin spider from a thick spider. A graph G_t is a tree iff it is connected and has $|V(G_t)| - 1$ edges; if $\overline{G_t}$ is connected and G_t has $\frac{k(k-1)}{2} - k + 1$ edges, where $k = |V(G_t)|$, then G_t is the complement of a tree. The fact that a chordless cycle on k vertices and its complement are connected and all its vertices have degree 2 and $k - 3$, respectively, is sufficient to recognize them. Thus, the recognition takes time linear in the size of the input graph G_t.

Given that the number of nodes of the tree $T(G)$ is $O(n)$, the fact that the number of edges of G is no less than the total number of edges of all the representative graphs of $T(G)$, and the fact that checking whether a graph H is a basic graph takes $O(|V(H)| + |E(H)|)$ time, then Step 4 of the algorithm Spanning_Trees-Number requires $O(n + m + \phi(G))$ time. Finally, Step 5 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 ℓ numbers takes $O(\ell)$ time. Therefore, the algorithm Spanning_Trees-Number takes $O(n + m + \phi(G))$ time. ∎

Remark 1. If a single computer word can store an integer as large as n^{n-2}, where n is the number of vertices of the input graph, then the uniform-cost criterion is certainly realistic (recall that the number of spanning trees of a graph on n vertices is at most n^{n-2}, which is achieved by the complete graph K_n). If however this is not the case, then the uniform-cost criterion is not realistic; but, in such a case, even the logarithmic-cost criterion (which takes into account the limited size of a real memory word which is logarithmic in the number stored) is somewhat unrealistic as well, 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 [31]).

6 Counting Spanning Trees in Linear Time

Let G be a graph on n vertices and m edges and let $\phi(G)$ be its non-basic cost. From Lemma 5.1, it is clear that if $\phi(G)$ is linear in the size of G, then the algorithm Spanning_Trees-Number runs in linear time. More precisely, followed by the functions applied in each internal neighborhood node u of $T(G)$, we have the following theorem.

Theorem 6.1. *Let G be a graph on n vertices and m edges, and let $T(G)$ be its modular decomposition tree. If every prime graph in $T(G)$ is (i) a spider graph, or (ii) a tree, or (iii) a cycle, or (iv) the complement of a tree, or (v) the complement of a cycle, or (vi) a graph of restricted (fixed) size, then the number of spanning trees of G can be computed in $O(n+m)$ time and space.*

We next investigate classes of graphs which have linear or constant non-basic cost.

Tree-cographs: Recently, many researchers have devoted their work on generalizing cographs. Tinhofer in [35] introduced the tree-cographs where the recursion, instead with a single vertex, starts with any tree. It follows that tree-cographs contain all trees and forests. Thus, every prime graph on the modular decomposition tree of a tree-cograph induces a tree. Then by Definition 2.3 $\alpha(G) = \emptyset$ which implies that the non-basic cost of a tree-cograph G is $\phi(G) = 0$.

$(q, q-4)$-graphs: Babel and Olariu in [4] proposed the generalizing concept of (q,t)-graphs. In such a graph, no set of at most q vertices contains more than t distinct P_4s. In particular, the $(q, q-4)$-graphs possess important structural properties (they admit a unique tree representation; see Theorem 6.2) [3], are brittle[1] for $q \leq 8$ [4] but are not brittle (and not perfect) for $q \geq 9$. It turns out that the cographs are precisely the $(4,0)$-graphs, the P_4-sparse graphs are the $(5,1)$-graphs, and the C_5-free P_4-extendible graphs are the $(6,2)$-graphs. In our terminology, the structure of $(q, q-4)$-graphs can be described as follows.

Theorem 6.2 (Babel and Olariu [4])**.** *Let G be a $(q, q-4)$-graph. Then, every prime graph in the modular decomposition of G is either a prime spider or a graph with fewer than q vertices.*

[1]A graph G is called *brittle* if each induced subgraph H of G contains a vertex that is not a midpoint or not an endpoint of any P_4.

Based on the above result, many optimization and domination problems (such as the vertex ranking, the path cover, the list coloring, the domination clique problem, etc.) can be solved in linear time for the class of $(q, q-4)$-graphs for fixed q [3]. The number of nodes of $T(G)$ is $O(n)$ and, thus, according to Definition 2.3 $\alpha(G) = O(n)$. For a $(q, q-4)$-graph G on n vertices $\alpha(G)$ contains only nodes whose corresponding prime graphs have a fixed number of vertices. Since computing the number of spanning trees of a graph on a fixed number of vertices takes constant time, the non-basic cost of a $(q, q-4)$-graph G on n vertices is $\phi(G) = O(n)$, for every fixed $q \geq 4$.

P_4-tidy graphs: As mentioned in [18], the class of P_4-tidy graphs was introduced by I. Rusu in order to illustrate the notion of P_4-domination in perfect graphs. A graph G is *P_4-tidy* if for any induced P_4, say $abcd$, there exists at most one vertex $v \in V(G) - \{a, b, c, d\}$ such that the subgraph $G[\{a, b, c, d, v\}]$ has at least two P_4s (i.e., the P_4 has at most one *partner*). The P_4-tidy graphs strictly contain the cographs, P_4-reducible, P_4-sparse, P_4-extendible, and P_4-lite graphs. The P_4-lite graphs were defined by Jamison and Olariu in [22]: a graph G is *P_4-lite* if every induced subgraph H of G with at most six vertices either contains at most two P_4's, or is a 3-sun, or is the complement of a 3-sun (a *3-sun* is a thick spider on six vertices with $R = \emptyset$). They remark that every P_4-sparse graph is P_4-lite and prove that every P_4-lite graph is brittle and is thus perfect. We mention here that the P_4-lite graphs coincide with the C_5-free P_4-tidy graphs.

Theorem 6.3 (Giakoumakis *et al.* [18])**.** *Let G be a P_4-tidy graph. Then, every prime graph in the modular decomposition of G is a P_5, a $\overline{P_5}$, a C_5, an urchin, or a starfish*[2]*.*

In [18], Theorem 6.3 played a crucial role in the linear-time recognition of P_4-tidy graphs and in the linear-time solution of the problems of calculating the clique and stability number, the chromatic number, and the hamiltonian path. In connection to our work, it implies that according to Definition 2.3 we have $\alpha(G) = \emptyset$, since for every N-node of $T(G)$ the corresponding representative graph is a spider or a tree or a cycle or the complement of a tree. Therefore the non-basic cost of a P_4-tidy graph G on n vertices is $\phi(G) = 0$.

Concluding, the number of spanning trees of the classes of tree-cographs, $(q, q-4)$-graphs for any fixed $q \geq 4$, and of P_4-tidy graphs can be efficiently computed. Moreover, it is not difficult to see that for these graphs the space needed by the algorithm Spanning_Trees-Number is $O(n+m)$; recall that the modular decomposition tree of a graph and its construction require space linear in the size of the graph [15, 25, 34]. Thus, the results of this section are summarized in the following theorem.

Theorem 6.4. *The number of spanning trees of a tree-cograph, or of a $(q, q-4)$-graph for any fixed $q \geq 4$, or of a P_4-tidy graph can be computed in $O(n + m)$ time and space, where n and m are the number of vertices and edges of the input graph.*

7 Concluding Remarks

In this paper we have presented a general approach for computing the number of spanning trees of a graph using modular decomposition, which yields a linear-time algorithm for the problem on

[2] An *urchin* is a thin spider; a *starfish* is a thick spider.

P_4-tidy graphs, on $(q, q-4)$-graphs for any fixed $q \geq 4$, and on tree-cographs. We have taken advantage of the structural properties of the modular decomposition tree of these graphs and used the Kirchhoff matrix tree theorem as a tool for proving the correctness of the proposed algorithm.

We remark that several other subclasses of well-known graphs possess structural properties that allow a linear-time computation of their number of spanning trees. Examples include the classes of tree-perfect and forest-perfect graphs which were introduced in [9] and for which the following holds (Lemma 4.1 in [9]): Let G have an induced path P_6. Then G is tree-perfect if and only if G is obtained from a tree by replacing the leaves by cographs or G is obtained from a prime graph on eight vertices by replacing four of its vertices by cographs [9]. Based on the previous result, it is straightforward that the number of spanning trees of a tree-perfect graph with an induced P_6 can be computed in linear time and space. Therefore we expect that our approach applies to other types of prime graphs (besides spider graphs, prime trees, chordless cycles and their complements), which would allow the efficient computation of the number of spanning trees for additional classes as well, e.g., the *semi-P_4-sparse* graphs [16] and the $(P_5, \mathit{diamond})$*-free* graphs [8].

Other interesting problems involve counting other structures as well, e.g., the number of perfect matchings, Hamiltonian cycles and Euler cycles. More precisely, it is known [24] that the number of perfect matchings can be computed efficiently for graphs having a *Pfaffian orientation.* As in the Kirchhoff matrix tree theorem, this method involves the computation of a determinant followed by a square root calculation. Thus, given such an orientation, an algorithm implementing our contraction approach may lead to efficient solutions for other combinatorial enumeration problems. Further, as mentioned in the introduction, a uniformly-most reliable network (defined in [11, 27]) must maximize the number of spanning trees. Thus, it is interesting to determine the types of graphs which have the maximum number of spanning trees for fixed numbers of vertices and edges (see [29, 33]). The problem may be approached as an optimization question on the s-values of the vertices, which are calculated by the algorithm Spanning_Trees-Number. Work along these lines is currently in progress and some preliminary results suggest that almost regularity seems to be the key to the solution.

Acknowledgements

The authors would like to thank the anonymous referees whose valuable suggestions helped improve the presentation of the paper.

References

[1] T. Atajan, X. Yong, and H. Inaba, An efficient approach for counting the number of spanning trees in circulant and related graphs, *Discrete Math.* **310** (2010) 1210–1221.

[2] R. B. Bapat, A. K. Lal, and S. Pati, Laplacian spectrum of weakly quasi-threshold graphs, *Graphs and Combinatorics* **24** (2008) 273–290.

[3] L. Babel, T. Kloks, I. Kratochvil, D. Kratsch, H. Mueller, and S. Olariu, Efficient algorithms for graphs with few P_4's, *Discrete Math.* **235** (2001) 29–51.

[4] L. Babel and S. Olariu, On the structure of graphs with few P_4's, *Discrete Appl. Math.* **84** (1998) 1–13.

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

[6] H.L. Bodlaender and U. Rotics, Computing the treewidth and the minimum fill-in with the modular decomposition, *Algorithmica* **36** (2003) 375–408.

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

[8] A. Brandstädt, (P_5,Diamond)-free graphs revisited: structure and linear time optimization, *Discrete Appl. Math.* **138** (2004) 13–27.

[9] A. Brandstädt and V.B. Le, Tree- and forest-perfect graphs, *Discrete Appl. Math.* **95** (1999) 141–162.

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

[11] C.J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, Oxford, 1974.

[12] C.J. Colbourn, J.S. Provan, and D. Vertigan, A new approach to solving three combinatorial enumeration problems on planar graphs, *Discrete Appl. Math.* **60** (1995) 119–129.

[13] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, *Proc. 19th ACM Symposium on the Theory of Computing*, (1987) 1–6.

[14] B. Courcelle, J.A. Makowsky, and U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, *Theory Comput. Syst.* **33** (2000) 125–150.

[15] A. Cournier and M. Habib, A new linear algorithm for modular decomposition, *Proc. 19th Int'l Colloquium on Trees in Algebra and Programming (CAAP'94)*, LNCS **787** (1994) 68–84.

[16] J.-L. Fouquet and V. Giakoumakis, On semi-P_4-sparse graphs, *Discrete Math.* **165-166** (1997) 277–300.

[17] J. Gagneur, R. Krause, T. Bouwmeester, and G. Casari, Modular decomposition of protein-protein interaction networks, *Genome Biology* **5**:R57 (2004).

[18] V. Giakoumakis, F. Roussel, and H. Thuillier, On P_4-tidy graphs, *Discrete Math. and Theoret. Comput. Science* **1** (1997) 17–41.

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

[20] M. Habib and C. Paul, A survey of the algorithmic aspects of modular decomposition, *Computer Science Review* **4** (2010) 41–59.

[21] F. Harary, *Graph Theory*, Addison-Wesley, 1969.

[22] B. Jamison and S. Olariu, A new class of brittle graphs, *Studies Appl. Math.* **81** (1989) 89–92.

[23] R.J. Lipton, D. Rose, and R.E. Tarjan, Generalized nested dissection, *SIAM J. Numerical Anal.* **16** (1979) 346–358.

[24] L. Lovasz and M.D. Plummer, *Matching Theory*, North-Holland, Amsterdam, 1986.

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

[26] R.H. Möhring and F.J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, *Annals of Discrete Mathematics* **19** (1984) 257–356.

[27] 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.

[28] 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.

[29] S.D. Nikolopoulos, L. Palios, and C. Papadopoulos, Maximizing the number of spanning trees in K_n-complements of asteroidal graphs, *Discrete Math.* **309** (2009) 3049–3060.

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

[31] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

[32] C. Papadopoulos and C. Voglis, Drawing graphs using modular decomposition, *Journal of Graph Algorithms and Applications* **11** (2007) 481–511.

[33] 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.

[34] M. Tedder, D. Corneil, M. Habib, and C. Paul, Simpler linear-time modular decomposition via recursive factorizing permutations, *Proc. 35th Int'l Colloquium on Automata, Languages and Programming (ICALP'08)*, LNCS **5125** (2008) 634–645.

[35] G. Tinhofer, Strong tree-cographs are Birkhoff graphs, *Discrete Appl. Math.* **22** (1988) 275–288.

[36] 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.

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