Counting Spanning Trees in Graphs Using Modular Decomposition

Stavros D. Nikolopoulos[1], Leonidas Palios[1], and Charis Papadopoulos[2]

[1] Department of Computer Science, University of Ioannina, GR-45110, Greece
{stavros,palios}@cs.uoi.gr
[2] Department of Mathematics, University of Ioannina, GR-45110, Greece
charis@cs.uoi.gr

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.

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 [5], is an important, well-studied quantity in graph theory, and appears in a number of applications. Most notable application fields are network reliability [19], enumerating certain chemical isomers [7], and counting the number of Eulerian circuits in a graph [5]. In particular, counting spanning trees is an essential step in many methods for computing, bounding, and approximating network reliability [8]; 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 or 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 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

N. Katoh and A. Kumar (Eds.): WALCOM 2011, LNCS 6552, pp. 202–213, 2011.

examined depending on the choice of G, such as when G is a labeled molecular graph [7], a circulant graph [2,27], a multi-star related graph [22], and a quasi-threshold graph [3,20].

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* [5], 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 [3,15,22]), but it requires $\Theta(n^{2.376})$ arithmetic operations on matrix entries and $\Theta(n^2)$ space [10]; 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 [16]. 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 [9] 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 *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. 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 [11,12]. Other application areas of modular decomposition arise in graph drawing [24] and in biological strategies of proteins [13].

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 the 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 [1,23]. Our algorithm is easy to implement; its correctness is established by means of the Kirchhoff 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 [18], 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.

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 $y \in V_j$.

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*. A non-trivial module is also called a *homogeneous* set. A prime spider is a prime graph, since it does not contain any non-trivial module. 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 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 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. 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.

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 $E(G_t) = \{u_iu_j \mid v_iv_j \in E(G),\ v_i \in M_{u_i}$ 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 $T(G)$.

The modular decomposition tree $T(G)$ of a graph G is constructed recursively as follows: modules corresponding to P-nodes are decomposed into their connected components, modules corresponding to S-nodes are decomposed into their co-connected components, and modules corresponding to N-nodes are decomposed 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 [18].

Definition 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.*

Let $T(G)$ be the modular decomposition of a graph G and let t be an N-node. We can 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; this implies that if all the prime graphs of a graph G belong to $\mathcal{F}$ then the number of spanning trees of G can be computed in time linear in the size of G.

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). 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 either for each vertex $v \in S$, $N(v) \cap K = \{f(v)\}$ or 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 to distinguish the two different bijections of (S3), 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. *The family $\mathcal{F}$ contains thin spiders, prime trees, chordless cycles of length greater than four, and their complements; 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 [10].

Definition 3. *Let $\mathcal{F}$ be the family of basic graphs, G be a graph, $T(G)$ be its modular decomposition tree, and let $\alpha(G) = \{t_1, t_2, \ldots, t_{s'}\}$ be the set of the N-nodes of $T(G)$ such that the representative graphs $G_{t_1}, G_{t_2}, \ldots, G_{t_{s'}}$ are not in $\mathcal{F}$. We define the* non-basic cost *of G as the value $\phi(G) = \sum_{t\in\alpha(G)} |V(G_t)|^{2.376} = \sum_{t\in\alpha(G)} |ch(t)|^{2.376}$, where $ch(t)$ denotes the set of children of node t in $T(G)$.*

For an $n \times n$ matrix M, the $(n-1)$-st order *minor* μ^i_j is the determinant of the $(n-1)\times(n-1)$ matrix obtained from M 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 1 (Kirchhoff Matrix Tree Theorem [5]). *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 a graph G on n vertices and let the p vertices $v_1, v_2, \ldots v_p$ define a module of G. In other words the vertices $v_1, v_2, \ldots v_p$ form the set of children of an internal node t of $T(G)$ and thus the subtree rooted at t is a contractible subtree. The Kirchhoff matrix tree theorem (Theorem 1) 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 M 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 M is as follows:

$$M = \left[\begin{array}{ccc:ccc} d(v_1) & & (-1)_{i,j} & & & \\ & \ddots & & & (-1)_{i,j'} & \\ (-1)_{i,j} & & d(v_p) & & & \\ \hdashline & & & d(v_{p+1}) & & (-1)_{i',j'} \\ & (-1)_{j',i} & & & \ddots & \\ & & & (-1)_{j',i'} & & d(v_{n-1}) \end{array}\right]. \qquad (1)$$

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, for $1 \leq i,j \leq p$. Similarly, the entries (i,j') and (j',i) (resp. the entries (i',j) and (j,i')) are both -1 if the vertices v_i and $v_{j'}$ (resp. $v_{i'}$ and v_j) are adjacent in G and are equal to 0 otherwise, $1 \leq i \leq p$ (resp. $p+1 \leq j' \leq n-1$). We note that, the off-diagonal elements in the first p rows and p columns of matrix M depend on the type of module formed by the vertices $v_1, v_2, \ldots, v_p$ (whether it is formed by a P-node, an S-node or an N-node). Additionally, since the first p vertices induce a module of G, the rows of M's submatrix formed by rows $1, 2, \ldots, p$ and columns $p+1, p+2, \ldots, n-1$ are identical and similarly, the columns of M's submatrix formed by rows $p+1, p+2, \ldots, n-1$ and columns $1, 2, \ldots, p$ are identical.

It is clear by Theorem 1 that $\tau(G) = \det(M)$; 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. In order to compute the determinant of matrix M 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 into another matrix M_1, by making the $p \times p$ submatrix a diagonal matrix, such that $\det(M) = \det(M_1)$. For example, these transformations can be applied by a 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, $1 \leq i \leq p \leq j' \leq n-1$. 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, and has the following form:

$$M_1 = \left[\begin{array}{ccc:ccc} s_1(v_1) & & 0 & & & \\ & \ddots & & & (c_1)_{i,j'} & \\ 0 & & s_1(v_p) & & & \\ \hdashline & & & s(v_{p+1}) & & (-1)_{i',j'} \\ & (c_1)_{j',i} & & & \ddots & \\ & & & (-1)_{j',i'} & & s(v_{n-1}) \end{array}\right]. \tag{2}$$

Lemma 1. *There exists a series of manipulations on matrix M of Eq. (1) which transforms it into matrix M_1 of Eq. (2) such that* $\det(M) = \det(M_1)$.

The difference of matrix M_1 from matrix M 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. 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 2. *For the determinant of matrix M_1 of Eq. (2) 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*

$$M' = \begin{bmatrix} s'(v_p) & \vdots & (-1)_{p,j'} & \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ & s(v_{p+1}) & & (-1)_{i',j'} \\ & & \ddots & \\ (-1)_{j',p} & & & \\ & (-1)_{j',i'} & & s(v_{n-1}) \end{bmatrix}, \quad \text{where } s'(v_p) = \tfrac{1}{\theta}.$$

We call this process *contraction* of the module formed by the vertices $v_1, v_2, \ldots v_p$. We observe that the matrix M' is an $(n-p) \times (n-p)$ matrix similar to the original matrix M; in fact, it is identical to the submatrix of the original matrix M formed 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 $s(v_p)$. Thus, if we assume (in an inductive fashion) that the determinant of the matrix M' can be expressed as the product of appropriate values $s'(v_p), s'(v_{p+1}), \ldots, s'(v_{n-1})$, then the determinant of the original matrix M is equal to the product of these values multiplied by the product of $s(v_1), s(v_2), \ldots, s(v_{p-1})$. The recursive application of an elimination process of the s-values (see Eq. (2)) and the contraction process of the module (see Lemma 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 $v_1, v_2, \ldots, v_n$, we make use of Theorem 1 and Lemma 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:

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. $s(v_i) \leftarrow d(v_i)$, $1 \leq i \leq n-1$;
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** {*let* $v_1, \ldots, v_p$ *be the children of* t}
   - 4.1 **if** the representative graph $G_t \in \mathcal{F}$ **then** $T \leftarrow$ *Handle-Basic*(t, T); **else** $T \leftarrow$ *Handle-NonBasic*(t, T); } {*elimination*}
   - 4.2 Compute the value: $\theta \leftarrow \sum_{i=1}^{p} \frac{c(i)^2}{s(v_i)}$;
   - 4.3 Update the s-values of v_{p-1}, 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. $\tau(G) \leftarrow \prod_{i=1}^{n-1} s(v_i)$;

Fig. 1. Algorithm *Spanning_Trees-Number*

- ▷ Elimination process: we eliminate the s-values in order to make diagonal the corresponding submatrix of Eq. (1). During that process we compute the values $s_1(v_i)$ and $c_1(i)$ of Eq. (2).
- ▷ 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 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. 1; 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 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.

3.2 Processing Basic Graphs

As described in the previous section, in order to compute the determinant of matrix M 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. Due to space restrictions we briefly explain each of the corresponding case.

- 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 graph. In [20], 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 graph. Function Eliminate_Tree is based on a similar approach; for a detailed proof, see [20].
- 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. It is easy to see that 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. By applying standard matrix operations and using one of the above functions on its complement, we transform the given matrix into the desired form. In this case we use a function that we call Eliminate-Complement. For technical reasons, throughout the transformation we add a dummy vertex that preserves a certain value that corresponds to the addition of a row and column in the given matrix.

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).

4 Counting Spanning Trees in Linear Time

We first describe the running time of our algorithm and next we investigate classes of graphs which have linear or constant non-basic cost.

Lemma 3. *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.*

Let G be a graph on n vertices and m edges and let $\phi(G)$ be its non-basic cost. From Lemma 3, 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 N-node u of $T(G)$, we have the following theorem.

Theorem 2. *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) their complements, or (v) a graph of restricted (fixed) size, then the number of spanning trees of G can be computed in $O(n + m)$ time and space.*

Recently, many researchers have devoted their work on generalizing cographs. Tinhofer in [26] 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 graph. Therefore by Theorem 2 the non-basic cost of a tree-cograph G is $\phi(G) = 0$.

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 our terminology, the structure of $(q, q-4)$-graphs can be described as follows: 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 [4]. Since computing the number of spanning trees of a graph on a fixed number of vertices takes constant time, Theorem 2 implies that the non-basic cost of a $(q, q-4)$-graph G on n vertices is $\phi(G) = O(n)$, for every fixed $q \geq 4$.

As mentioned in [14], the class of P_4-tidy graphs was introduced by 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*). 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, a thin spider, or a thick spider [14]. Thus in connection to our work, the previous result implies that the non-basic cost $\phi(G)$ of a P_4-tidy graph G is linear in the number of vertices of G. Therefore the non-basic cost of a P_4-tidy graph G on n vertices is $\phi(G) = 0$.

Concluding, it is not difficult to see that for the mentioned graph classes 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 [18]. Thus, the results of this section are summarized in the following theorem.

Theorem 3. *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.*

5 Concluding Remarks

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 [17] 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 [8,19]) 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 [21,25]). 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.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Atajan, T., Yong, X., Inaba, H.: An efficient approach for counting the number of spanning trees in circulant and related graphs. Discrete Math. 310, 1210–1221 (2010)
3. Bapat, R.B., Lal, A.K., Pati, S.: Laplacian spectrum of weakly quasi-threshold graphs. Graphs and Combinatorics 24, 273–290 (2008)
4. Babel, L., Olariu, S.: On the structure of graphs with few P_4's. Discrete Appl. Math. 84, 1–13 (1998)
5. Biggs, N.: Algebraic Graph Theory. Cambridge University Press, London (1974)

6. Bodlaender, H.L., Rotics, U.: Computing the treewidth and the minimum fill-In with the modular decomposition. Algorithmica 36, 375–408 (2003)
7. Brown, T.J.N., Mallion, R.B., Pollak, P., Roth, A.: Some methods for counting the spanning trees in labeled molecular graphs, examined in relation to certain fullerenes. Discrete Appl. Math. 67, 51–66 (1996)
8. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press, Oxford (1974)
9. Colbourn, C.J., Provan, J.S., Vertigan, D.: A new approach to solving three combinatorial enumeration problems on planar graphs. Discrete Appl. Math. 60, 119–129 (1995)
10. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. In: Proc. 19th ACM Symposium on the Theory of Computing, pp. 1–6 (1987)
11. Courcelle, B., Delhommé, C.: The modular decomposition of countable graphs: Constructions in monadic second-order logic. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 325–338. Springer, Heidelberg (2005)
12. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33, 125–150 (2000)
13. Gagneur, J., Krause, R., Bouwmeester, T., Casari, G.: Modular decomposition of protein-protein interaction networks. Genome Biology 5, R57 (2004)
14. Giakoumakis, V., Roussel, F., Thuillier, H.: On P_4-tidy graphs. Discrete Math. and Theoret. Comput. Science 1, 17–41 (1997)
15. Golin, M.J., Yong, X., Zhang, Y.: Chebyshev polynomials and spanning tree formulas for circulant and related graphs. Discrete Math 298, 334–364 (2005)
16. Lipton, R.J., Rose, D., Tarjan, R.E.: Generalized nested dissection. SIAM J. Numerical Anal. 16, 346–358 (1979)
17. Lovasz, L., Plummer, M.D.: Matching Theory. North-Holland, Amsterdam (1986)
18. McConnell, R.M., Spinrad, J.: Modular decomposition and transitive orientation. Discrete Math. 201, 189–241 (1999)
19. Myrvold, W., Cheung, K.H., Page, L.B., Perry, J.E.: Uniformly-most reliable networks do not always exist. Networks 21, 417–419 (1991)
20. Nikolopoulos, S.D., Papadopoulos, C.: The number of spanning trees in K_n-complements of quasi-threshold graphs. Graphs and Combinatorics 20, 383–397 (2004)
21. Nikolopoulos, S.D., Palios, L., Papadopoulos, C.: Maximizing the number of spanning trees in K_n-complements of asteroidal graphs. Discrete Math. 309, 3049–3060 (2009)
22. Nikolopoulos, S.D., Rondogiannis, P.: On the number of spanning trees of multi-star related graphs. Inform. Process. Lett. 65, 183–188 (1998)
23. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
24. Papadopoulos, C., Voglis, C.: Drawing graphs using modular decomposition. Journal of Graph Algorithms and Applications 11, 481–511 (2007)
25. Petingi, L., Rodriguez, J.: A new technique for the characterization of graphs with a maximum number of spanning trees. Discrete Math. 244, 351–373 (2002)
26. Tinhofer, G.: Strong tree-cographs are Birkhoff graphs. Discrete Appl. Math. 22, 275–288 (1988)
27. Zhang, Y., Yong, X., Golin, M.J.: The number of spanning trees in circulant graphs. Discrete Math. 223, 337–350 (2000)