Counting minimum weight spanning trees

Andrei Z. Broder* Ernst W. Mayr†

We present an algorithm for counting the number of minimum weight spanning trees, based on the fact that the generating function for the number of spanning trees of a given graph, by weight, can be expressed as a simple determinant. For a graph with n vertices and m edges, our algorithm requires $O(\mathcal{M}(n))$ elementary operations, where $\mathcal{M}(n)$ is the number of elementary operations needed to multiply $n \times n$ matrices. The previous best algorithm for this problem, due to Gavril [3], required $O(n\mathcal{M}(n))$ operations. (Since the number of trees in a complete graph is n^{n-2}, our algorithm, as well as Gavril's, might involve operations on numbers of this magnitude. Such operations are accounted as elementary operations.)

Theorem 1 *Let G=(V,E) be a graph, with vertex set $V = \{1, \ldots, n\}$, edge set $E = \{e_1, \ldots, e_m\}$, and edge weights $w_{i,j}$. Arbitrarily orient the edges of G. Let $\mathcal{A}(x)$ be the n by m matrix defined by*

$$a_{i,j}(x) = \begin{cases} x^{w_{i,k}} & \text{if } e_j = (i,k); \\ -x^{w_{i,k}} & \text{if } e_j = (k,i); \\ 0 & \text{otherwise.} \end{cases}$$

Then the generating function for the number of spanning trees by weight is the determinant of the matrix

$$D_G = \tilde{\mathcal{A}}(x) \times \tilde{\mathcal{A}}^T(1)$$

where $\tilde{\mathcal{A}}$ is obtained from $\mathcal{A}$ by the deletion of an arbitrary row.

*Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA. E-mail: `broder@pa.dec.com`

†Institut für Informatik, Tech. Univ. München, 80290 München, Germany. E-mail: `mayr@informatik.tu-muenchen.de`

Proof: Assuming that we have deleted the n-th row from $\mathcal{A}(x)$, the matrix D_G is given by

$$d_{i,j} = \begin{cases} \sum_{\{i,k\}\in E} x^{w_{i,k}} & \text{if } i = j; \\ -x^{w_{i,j}} & \text{if } i \neq j \text{ and } \{i,j\} \in E; \\ 0 & \text{otherwise,} \end{cases}$$

where $1 \leq i, j \leq n-1$.

The claim now follows analogous to the proof of the matrix tree theorem (see e.g. Theorem 2.10 in [2]). $\square$

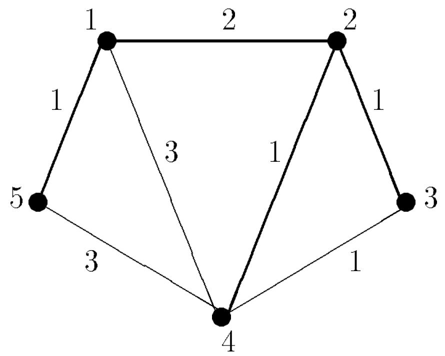

Figure 1: Example

For instance for the graph in Figure 1, the determinant $|D_G(x)|$ is

$$|D_G| = \begin{vmatrix} x^3+x^2+x & -x^2 & 0 & -x^3 \\ -x^2 & x^2+2x & -x & -x \\ 0 & -x & 2x & -x \\ -x^3 & -x & -x & 2x^3+2x \end{vmatrix} = 2x^9+3x^8+7x^7+6x^6+3x^5.$$

Obviously, if $w_{\min}$ is the weight of a minimum spanning tree of G then, by Theorem 1, $x^{w_{\min}}$ divides $|D_G(x)|$. However, as our example shows, the product of the gcd's of each column might be only a strict divisor of $x^{w_{\min}}$. The gist of our algorithm is that it is possible to use column operations on $D_G(x)$ that preserve the value of $|D_G(x)|$, such that eventually the product of the gcd's of each column is exactly $x^{w_{\min}}$, after which counting the

```
begin Algorithm A.
    Let T be an arbitrary spanning tree of G.
    while T≠{n} do
        choose a leaf i≠n of T; let p(i) be its parent in T;
        if p(i)≠n then add column i to column p(i) fi;
        delete i from T
    od
end
```

Figure 2: Listing of Algorithm **A**.

number of minimum spanning trees reduces to the evaluation of the factored determinant at $x = 0$.

To this end, fix an arbitrary minimum spanning tree T of G. We associate to it a sequence of operations on D_G as given by Algorithm **A** depicted in Figure 2.

For example, for the graph in Figure 1, the subgraph drawn in bold lines is a minimum spanning tree. To this tree, Algorithm **A** might associate the following sequence of operations (we show below that the final result does not depend on the processing order chosen by **A**):

add column 4 to column 2;
add column 3 to column 2;
add column 2 to column 1;

after which, the determinant $|D_G|$ has the form

$$\begin{vmatrix} x & -x^2-x^3 & 0 & -x^3 \\ 0 & x^2 & -x & -x \\ 0 & 0 & 2x & -x \\ x^3 & 2x^3 & -x & 2x^3+2x \end{vmatrix} = x^5 \begin{vmatrix} 1 & -x-1 & 0 & -x^2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 2 & -1 \\ x^2 & 2x & -1 & 2x^2+2 \end{vmatrix}.$$

(We factored x from each of columns 1, 3, and 4, and x^2 from column 2.)

Lemma 2

1. *The operations performed by Algorithm* **A** *preserve the value of* $|D_G|$.
2. *The final result* D'_G *does not depend on the processing order in* **A**.
3. *The product of the highest powers of* x *that can be factored from each column of* D'_G *is exactly* $x^{w_{\min}}$.

Proof: The first claim follows from the fact that Algorithm **A** only performs elementary column operations on the matrix. To prove the other claims, assume that the tree T is rooted at node n. Let S_j be the set of nodes in T that hang from node j, including j. Clearly column j in D'_G is precisely the sum of those columns in D_G that correspond to the nodes in S_j, thus establishing the second claim. Furthermore, it is easy to check that the entry $d'_{i,j}$ of matrix D'_G in row i and column j is

$$d'_{i,j} = \begin{cases} \displaystyle\sum_{k \notin S_j;\ \{i,k\} \in E} x^{w_{i,k}} & \text{if } i \in S_j\,; \\ \displaystyle -\sum_{k \in S_j;\ \{i,k\} \in E} x^{w_{i,k}} & \text{if } i \notin S_j\,. \end{cases}$$

The reason is that as noted above, column j in D'_G is the sum of those columns in D_G that correspond to the nodes in S_j, and, if $i \in S_j$ then this sum consists of $d_{i,i}$ with those terms belonging to edges with both endpoints in S_j cancelled, whereas, if $i \notin S_j$ then the sum is simply the sum of those elements in row i and columns in S_j of D_G.

Therefore, the entries of column j contain only terms corresponding to edges between S_j and $V \setminus S_j$. The only edge from T in the cut $(S_j, V \setminus S_j)$ is the edge $\{j, p(j)\}$. Any edge between a vertex of S_j and a vertex of $V \setminus S_j$ has weight at least $w_{j,p(j)}$, because otherwise the edge $\{j, p(j)\}$ could be replaced in T by an edge of lower weight, contradicting the minimality of T. Thus, all edges in this cut have weight at least $w_{j,p(j)}$. Hence we can factor $x^{w_{j,p(j)}}$ from column j. But $\prod_{j=1,\ldots,n-1} x^{w_{j,p(j)}} = x^{w_{\min}}$. $\square$

A naive implementation of Algorithm **A** could require as many as nm operations; to implement it more efficiently we start by sorting the vertices of T in reverse topological order, and renumbering them, so that

$$j < p(j), \qquad \text{for } j = 1, \ldots, n-1\,.$$

```
begin Algorithm A'.
    Let T be an arbitrary spanning tree of G.
    for j ← 1 to n − 1 do
        S_j ← {j} ∪ {the descendants of j in T}
        for i ∉ S_j do
            if j is a leaf of T then
                d''_{i,j} ← −δ_{{i,j}∈E} x^{w_{i,j}}
            else /* j is an internal node of T */
                let j_1, ..., j_r be the children of j in T;
                d''_{i,j} ←
                    min deg monomial of (−δ_{{i,j}∈E} x^{w_{i,j}} +
                    Σ_{s=1}^{r} d''_{i,j_s})
            fi
        od
    od ;
    for i ← 1 to n − 1 do
        for j ancestor of i in T, j in increasing order do
            if l_i ⊆ S_j then d''_{i,j} ← 0
            else
                let w be the weight in the first weight class
                    in l_i containing a vertex ∉ S_j;
                let r be the number of elements in this
                    weight class which are not in S_j;
                d''_{i,j} ← r · x^w
            fi
        od
    od
end
```

Figure 3: Listing of Algorithm $\mathbf{A}'$.

For our purposes it suffices to compute $d''_{i,j}$, defined as the minimum degree monomial of every $d'_{i,j}$. The algorithm $\mathbf{A}'$ depicted in Figure 3 does this. For this algorithm, we precompute for $i = 1, \ldots, n-1$, the lists l_i of edges incident to vertex i, and then sort each list, first in increasing weight order, and then within each weight class we sort the list such that the ancestors of i (if any) come first and in increasing order. (Within a weight class of l_i the order of the nodes that are not ancestors of i is irrelevant.) Furthermore for each element in the list we store the number of elements in the same weight class that follow it in the list.

The Algorithm $\mathbf{A}'$ as given below first determines, for all j and all nodes $i \notin S_j$, the minimum degree monomial $d''_{i,j}$ of the (i,j)-entry of D'_G. This part uses the fact that the parent of a node has a larger number than the node itself, and that the computations for different rows are independent. In the second part of the algorithm, the entries for all j and all nodes $i \in S_j$, i.e., node j an ancestor of node i, are computed. In this case, it is advantageous to have the outer loop run over all nodes i in topological order from the leaves towards the root, and to consider, for each node i, the sequence of its ancestors towards the root.

In the listing of Algorithm $\mathbf{A}'$, as given in Figure 3, the variable $\delta_{\{i,j\}\in E}$ is 1 if $\{i,j\} \in E$ and 0 otherwise.

The cost for the topological sort and for initially constructing the lists l_i is $O(m \log n)$.

The first loop in Algorithm $\mathbf{A}'$ determines the minimum degree monomial of $d'_{i,j}$, for all i and j such that $i \notin S_j$. Its running time is $O(n^2)$ since each child of j contributes at most $O(n)$ operations, one for each i, and there are at most $O(n)$ children in total.

The second loop computes the minimum degree monomial of $d'_{i,j}$, for all i and j such that $i \in S_j$, or, in other words, for j being an ancestor of i. Since we are considering the ancestors of i in increasing order $j_1 < j_2 < j_3 < \cdots$, the corresponding sets form a tower $S_{j_1} \subset S_{j_2} \subset S_{j_3} \subset \cdots$. This fact and our presorting of the lists l_i ensure that we can implement the inner loop such that each element in the list l_i is scanned only once. Thus the total cost of the second loop is $O(m)$.

Hence, the total running time, including the final computation of the determinant is $O(m \log n + n^2 + \mathcal{M}(n)) = O(\mathcal{M}(n))$ operations and we conclude with

Theorem 3 *For a graph with n vertices and m edges and nonnegative integral edge weights, the number of minimum spanning trees can be computed using $O(\mathcal{M}(n))$ elementary operations, where $\mathcal{M}(n)$ is the number of elementary operations needed to multiply two $n \times n$ matrices.*

Acknowledgment. We wish to thank an anonymous referee for several useful comments and corrections.

References

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

[2] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[3] F. Gavril. Generating the maximum spanning-trees of a weighted graph. *Journal of Algorithms*, 8:592–597, 1987.