Complexity of Computation of a Spanning Tree Enumeration Algorithm

R. JAYAKUMAR, K. THULASIRAMAN, AND M. N. S. SWAMY, FELLOW, IEEE

Abstract—**In 1968, Char [4] presented an algorithm to enumerate all the spanning trees of an undirected graph G. This algorithm starts with a known initial spanning tree of G, and generates all the other spanning trees along with certain spanning non-tree subgraphs of G. In this paper a detailed complexity analysis of Char's algorithm and methods to speed up the algorithm are discussed. Two heuristics for the selection of the initial spanning tree are suggested. These heuristics result in a considerable reduction in the number of spanning non-tree subgraphs generated. A technique called path compression, aimed at reducing the actual number of comparisons, is described. Computational results on several randomly generated graphs are presented to illustrate the improvement achieved.**

I. Introduction

ENUMERATING all the spanning trees of a graph without duplication is one of the widely studied graph problems in Electrical Engineering and Computer Science literature. Since the number of spanning trees of a graph increases rapidly with the size of the graph, a highly efficient algorithm is desired to enumerate all the spanning trees of a graph. Several algorithms of varying efficiency have been proposed in the literature. One of the well-known algorithms is due to Minty [1], which has been shown [2] to be of complexity $O(m+n+mt)$, where m and n are the number of edges and the number of vertices of the graph, respectively, and t is the number of spanning trees of the graph. Another efficient algorithm of complexity $O(m+n+nt)$ is due to Gabow and Myers [3].

In 1968, Char [4] had presented a conceptually simple and elegant algorithm to enumerate all the spanning trees of a graph. However, Char had not presented a complexity analysis of his algorithm. The recent analysis of Char's algorithm presented in [5] suggests that this algorithm might be the best of all the algorithms available so far for the spanning tree enumeration problem. In this paper, we not only present a more detailed complexity analysis of Char's algorithm but also discuss different methods to further improve the speed of the algorithm.

In Section II we give a formal description of Char's algorithm and summarise some of its interesting properties reported in [5]. In Section III we give a detailed complexity analysis of this algorithm and present several of its quantitative and qualitative properties. In Section IV we develop two heuristic procedures which help speed up Char's algorithm by minimizing the number of non-tree subgraphs generated by the algorithm. Finally, in Section V, we discuss a general technique called path compression, which can be used for an efficient implementation of Char's algorithm to reduce the actual number of comparisons made by the algorithm.

For graph theory terms and notation not defined here, see [6]. Without any loss of generality we also assume that the graphs considered in this paper are simple biconnected undirected graphs.

Manuscript received March 31, 1983; revised September 6, 1983 and November 25, 1983. This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A-7739 and under Grant A-4680.

The authors are with the Faculty of Engineering and Computer Science, Concordia University, Montreal, P.Q., H3G 1M8 Canada.

II. Char's Algorithm to Enumerate All the Spanning Trees

Consider a connected undirected graph $G=(V,E)$ with $n=|V|$ vertices and $m=|E|$ edges. Let the vertices of G be denoted as $1,2,\cdots,n$. Let $\lambda=(\text{DIGIT}(1), \text{DIGIT}(2),\cdots, \text{DIGIT}(n-1))$ denote a $(n-1)$-digit sequence of vertices such that DIGIT(i) is a vertex adjacent to vertex i in G. With each such sequence λ we can associate a subgraph $G_\lambda=(V_\lambda, E_\lambda)$ of G such that

$$V_\lambda=\{1,2,\cdots,n\}$$

and

$$E_\lambda=\{(1,\text{DIGIT}(1)),(2,\text{DIGIT}(2)),\cdots,(n-1,\text{DIGIT}(n-1))\}.$$

Char's algorithm first performs a Breadth-First Search [6] on G and finds a spanning tree called the initial spanning tree. During this search, the vertices of G are also renumbered as $n, n-1,\cdots,1$ in the order in which they are visited. Let $\lambda_0=(\text{REF}(1), \text{REF}(2),\cdots, \text{REF}(n-1))$ be the sequence corresponding to this spanning tree. Starting with λ_0, the algorithm enumerates all the other spanning trees of G by generating the sequences corresponding to the spanning trees of G. During this enumeration, the algorithm also generates certain sequences which correspond to spanning non-tree subgraphs of G. The sequences corresponding to spanning trees are referred to as *tree sequences* and those corresponding to spanning non-tree subgraphs are referred to as *non-tree sequences*. The sequence λ_0 is called the *initial tree sequence*.

Char's algorithm classifies a generated sequence as a tree sequence if it satisfies the following.

Tree Compatibility Property

The sequence (DIGIT(1), DIGIT(2),$\cdots$, DIGIT($n-1$)) represents a spanning tree of graph G if and only if

0098-4094/84/1000-0853$01.00

for each $j \leqslant n-1$ there exists, in G, a sequence of edges (chosen from among the edges $(1, \text{DIGIT}(1))$, $(2, \text{DIGIT}(2)), \cdots, (n-1, \text{DIGIT}(n-1))$) with $(j, \text{DIGIT}(j))$ as the starting edge, which leads to a vertex $k > j$. □

If a sequence does not have the above property, then it is a non-tree sequence.

Char's algorithm can be presented in ALGOL-like notation as follows.

Char's Algorithm to Enumerate all the Spanning Trees

```
procedure CHAR;
comment The graph G is represented by the adjacency lists
        of its vertices. SUCC(DIGIT(i)) is the entry next
        to DIGIT(i) in the adjacency list of vertex i.
begin
  find the initial spanning tree and obtain the initial tree
  sequence λ0 = (REF(1), REF(2), ···, REF(n-1));
  renumber the vertices of the graph using the initial
  spanning tree;
  initialize DIGIT(i) := REF(i), 1 ≤ i ≤ n-1;
  output the initial spanning tree;
  k := n-1;
  while k ≠ 0 do begin
    if SUCC(DIGIT(k)) ≠ nil
      then begin
        DIGIT(k) := SUCC(DIGIT(k));
        if DIGIT(i), 1 ≤ i ≤ n-1, is a tree sequence
          then begin
            output the tree sequence;
            k := n-1
          end
      end
      else begin
        DIGIT(k) := REF(k);
        k := k-1
      end
    end
end CHAR;
```

Given any sequence $\lambda = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(n-1))$, Char's algorithm obtains the next sequence by changing DIGIT(k) in λ. In the new sequence $\text{DIGIT}(i) = \text{REF}(i) > i$, $k+1 \leqslant i \leqslant n-1$, and DIGIT(1), DIGIT(2), $\cdots$, DIGIT($k-1$) have the same values as in the previous sequence. Hence the new sequence is to be tested for tree compatibility property only at position k and this test, in the worst case, involves $k-1$ comparisons. Hence at most n computational steps are required to generate and test a sequence. So, if t_0 is the number of non-tree sequences and t is the number of tree sequences generated by Char's algorithm, in the worst case $n(t+t_0)$ computational steps are required to enumerate all the spanning trees of the given graph and hence, Char's algorithm is of time complexity $O(m+n+n(t+t_0))$, which also includes the complexity of finding the initial spanning tree.

The following are two of the interesting properties of Char's algorithm. For other properties, see [5].

Theorem 1

For a complete graph, the number t_0 of spanning non-tree subgraphs generated by Char's algorithm is independent of the initial spanning tree. □

Theorem 2

Let $G^{(n-1)}$ be the set of all connected n-vertex graphs having at least one vertex of degree $n-1$. For any graph $G \in G^{(n-1)}$, $t_0 \leqslant t$, if the initial spanning tree is a star tree. □

A characterization of each spanning non-tree subgraph generated by Char's algorithm is also given in [5].

III. Computational Complexity of Char's Algorithm

Since the computational complexity of Char's algorithm is $O(m+n+n(t+t_0))$, any complexity analysis of this algorithm would require a study of the number $(t+t_0)$. With this objective in view, we first obtain an expression for $(t+t_0)$.

Let

$$T = \bigcup_{i=0}^{n-1} T_i$$

be the set of all the tree sequences such that

(i) $T_0 = \{\lambda_0\}$, and

(ii) T_i, $1 \leqslant i \leqslant n-1$, is the set of all the tree sequences of the form (DIGIT(1), DIGIT(2), $\cdots$, DIGIT(i), REF($i+1$), REF($i+2$), $\cdots$, REF($n-1$)) with $\text{DIGIT}(i) \neq \text{REF}(i)$.

Also let

$$T' = \bigcup_{i=1}^{n-1} T_i'$$

be the set of all the non-tree sequences such that T_i' is the set of all the non-tree sequences of the form (DIGIT(1), DIGIT(2), $\cdots$, DIGIT(i), REF($i+1$), REF($i+2$), $\cdots$, REF($n-1$)) with $\text{DIGIT}(i) \neq \text{REF}(i)$ for $1 \leqslant i \leqslant n-1$. Note that $|T| = t$. Further, it follows from the characterisation of the non-tree subgraphs given in [5] that $|T'| = t_0$.

Theorem 3

Let G be a connected n-vertex undirected graph with its vertices numbered as in Char's algorithm. Let $G_k^{(s)}$, $1 \leqslant k \leqslant n-1$, be the graph obtained from G by coalescing the vertices $k, k+1, \cdots, n$ and let $t(k)$ be the number of spanning trees of $G_k^{(s)}$. If t is the number of tree sequences and t_0 is the number of non-tree sequences generated by the algorithm, then

$$t + t_0 = 1 + \sum_{k=1}^{n-1} (\deg(k) - 1) t(k)$$

where $\deg(k)$, $1 \leqslant k \leqslant n$, is the degree of vertex k in G.

Proof:

Consider a tree sequence $\lambda_k = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1), \text{REF}(k), \text{REF}(k+1), \cdots,$

REF$(n-1)$) generated by Char's algorithm. Let $G_k = (V_k, E_k)$ be the spanning tree corresponding to λ_k and $G'_k = (V'_k, E'_k)$ be the spanning 2-tree obtained from G_k by deleting the edge $(k, \text{REF}(k))$. Since $\text{REF}(i) > i$, $1 \leqslant i \leqslant n-1$, it follows that in G'_k the edges $(k+1, \text{REF}(k+1))$, $(k+2, \text{REF}(k+2)), \cdots, (n-1, \text{REF}(n-1))$ are in one component, say the component $G'_{k,1} = (V'_{k,1}, E'_{k,1})$, and the vertex k is in the other component, say the component $G'_{k,2} = (V'_{k,2}, E'_{k,2})$. Note that in $G'_{k,1}$ and in $G'_{k,2}$ there exists an unique path between every pair of vertices.

Consider any vertex $v \neq \text{REF}(k)$, adjacent to vertex k. Let $\lambda^*_k = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1), v, \text{REF}(k+1), \text{REF}(k+2), \cdots, \text{REF}(n-1))$ and $G^*_k = (V'_k, E'_k \cup (k, v))$ be the subgraph corresponding to λ^*_k. Now the following two cases arise.

(i) If $v \in V'_{k,1}$, then G^*_k is a spanning tree of G. Thus the sequence λ^*_k with $v \in V'_{k,1}$ is a tree sequence passing the tree compatibility test at position k.

(ii) If $v \in V'_{k,2}$, then in G^*_k the edge (k, v), along with the unique path in $G'_{k,2}$ between the vertices k and v, forms a circuit passing through the vertex k, and so G^*_k is a non-tree subgraph of G. Thus the sequence λ^*_k with $v \in V'_{k,2}$ is a non-tree sequence failing the tree compatibility test at position k.

Since vertex k is adjacent to $\deg(k)-1$ vertices other than $\text{REF}(k)$, there are $\deg(k)-1$ distinct λ^*_k's which have the same DIGIT(1), DIGIT(2), $\cdots$, DIGIT$(k-1)$ as λ_k. Each one of these sequences is either a tree sequence or a non-tree sequence and so all these sequences belong to $T_k \cup T'_k$. Thus if $t(k)$ is the number of all the tree sequences of the form $\lambda_k = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1), \text{REF}(k), \text{REF}(k+1), \cdots, \text{REF}(n-1))$, then

$$|T_k \cup T'_k| = (\deg(k)-1)t(k). \tag{1}$$

Since in the spanning tree corresponding to λ_k, the edges $(k, \text{REF}(k))$, $(k+1, \text{REF}(k+1)), \cdots, (n-1, \text{REF}(n-1))$ are present, it follows that $t(k)$ is the number of spanning trees of G in which the edges $(k, \text{REF}(k))$, $(k+1, \text{REF}(k+1)), \cdots, (n-1, \text{REF}(n-1))$ are present. Thus $t(k)$ is the number of spanning trees of the graph obtained from G by coalescing the vertices $k, k+1, \cdots, n-1$, REF(k), REF$(k+1), \cdots,$ REF$(n-1)$. But $\{k, k+1, \cdots, n-1, \text{REF}(k), \text{REF}(k+1), \cdots, \text{REF}(n-1)\} = \{k, k+1, \cdots, n\}$ because $\text{REF}(i) > i$, $1 \leqslant i \leqslant n-1$, and so $t(k)$ is the number of spanning trees of $G^{(s)}_k$, the graph obtained from G by coalescing the vertices $k, k+1, \cdots, n$. Also the total number of sequences generated by Char's algorithm is

$$t + t_0 = |T_0| + \sum_{k=1}^{n-1} |T_k \cup T'_k| = 1 + \sum_{k=1}^{n-1} |T_k \cup T'_k|.$$

From these observations and equation (1) the theorem follows. □

From Theorem 3 we can easily prove Theorem 1 stated in Section II.

Now we develop a systematic procedure to compute $t(k)$. Let $G(w)$ be a weighted undirected graph in which $w(i, j)$ denotes the weight of the edge (i, j). Let i be any vertex of $G(w)$ and let $\Gamma(i)$ be the set of vertices adjacent to vertex i in $G(w)$. Let

$$d_i = \sum_{j \in \Gamma(i)} w(i, j).$$

By *pivotal condensation* at vertex i in $G(w)$ we mean the following operation: For each pair of vertices $j_1, j_2 \in \Gamma(i)$, if the edge (j_1, j_2) is already present in $G(w)$, then increase its weight by $w(i, j_1)w(i, j_2)/d_i$; otherwise add to $G(w)$ the edge (j_1, j_2) with the weight $w(i, j_1)w(i, j_2)/d_i$. After all possible pairs of neighbors of the vertex i are considered, delete from $G(w)$ the vertex i and all the edges incident on it.

Let N be a resistive network consisting of one Siemens admittances and $G(N)$ be the graph of N in which all the edges are of unit weight. Let A be a subset of the vertex set $V = \{1, 2, \cdots, n\}$ of N. Let the networks N_A and N^0_A be defined as

N_A the network that results after coalescing all the vertices of N which do not belong to A,

N^0_A the network that results after suppressing all the vertices of N which belong to A.

If $T(N)$, $T(N_A)$, and $T(N^0_A)$ denote the sum of tree-admittance products of the networks N, N_A, N^0_A, respectively, then it has been shown in [7] that

$$T(N) = T(N_A)T(N^0_A). \tag{2}$$

Note that the graph $G(N^0_A)$ of the network N^0_A can be obtained from the graph $G(N)$ by performing pivotal condensation, in $G(N)$, at all the vertices in A. Let $A = \{1, 2, \cdots, k-1\}$, $G_1(N) = G(N)$ and the graph $G_i(N)$, $2 \leqslant i \leqslant k-1$, be obtained from $G_{i-1}(N)$ by performing a pivotal condensation at vertex $i-1$ in $G_{i-1}(N)$. If d_i, $1 \leqslant i \leqslant k-1$, is the sum of admittances of all the edges incident on vertex k in $G_i(N)$, then as shown in [7]

$$T(N) = d_1 d_2 \cdots d_{k-1} T(N^0_A). \tag{3}$$

Comparing (2) and (3) we get

$$T(N_A) = d_1 d_2 \cdots d_{k-1}$$

when $A = \{1, 2, \cdots, k-1\}$. Note that the graph of the network N_A is obtained from G by coalescing the vertices $k, k+1, \cdots, n$ and hence it is $G^{(s)}_k$. Since each element of N is of admittance one Siemens, the admittance product of each spanning tree is one and so $T(N_A)$ is the number of spanning trees of the graph $G^{(s)}_k$. Thus we get the following theorem.

Theorem 4

The number of spanning trees $t(k)$ of the graph $G^{(s)}_k$ is given by

$$t(k) = d_1 d_2 \cdots d_{k-1}. \qquad \square$$

If $A = \{1, 2, \cdots, n-1\}$, then the above theorem reduces to the following corollary.

Corollary 4.1

The number of spanning trees of G is given by

$$t = d_1 d_2 \cdots d_{n-1}. \qquad \square$$

Using the above corollary and Theorem 4 in Theorem 3, we get the following.

Theorem 5

The number of sequences generated by Char's algorithm is

$$t + t_0 = 1 + t \sum_{k=1}^{n-1} \frac{\deg(k) - 1}{d_{n-1} d_{n-2} \cdots d_k}. \qquad \square$$

Now we illustrate the above procedure to compute $(t + t_0)$ for the graph G in Fig. 1(a). This graph has 8 spanning trees and Char's algorithm generates 11 sequences for the vertex numbering shown. The graphs G_1, G_2, and G_3 are shown in Fig. 1(a)–(c), respectively. Note that $d_1 = 2$, $d_2 = 5/2$, and $d_3 = 8/5$. Thus

$$t = d_1 d_2 d_3 = 8$$

and

$$t + t_0 = 1 + t \sum_{k=1}^{3} \frac{\deg(k) - 1}{d_{n-1} d_{n-2} \cdots d_k} = 11.$$

Using Theorem 5 we can easily prove the following.

Theorem 6

For an n-vertex complete graph

$$t_0 = n^{n-2} - \left[\frac{n^{n-1} - 1}{(n-1)^2} \right]. \qquad \square$$

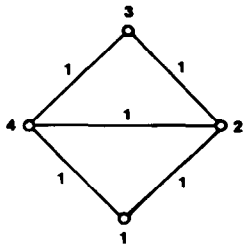
Corollary 6.1

For a complete graph $t_0 < t$, for any choice of the initial spanning tree. $\square$

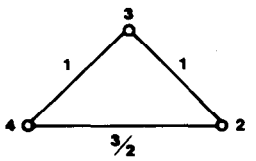
The value of $(t + t_0)$ given in Theorem 3 depends on the number of spanning trees of $G_k^{(s)}$, which is obtained from G by coalescing the vertices $k, k+1, \cdots, n$. So, for two different initial spanning trees, the values of $t(k)$ for a given k will be the same if the set of vertices which receive the numbers $k, k+1, \cdots, n$ as given by Char's algorithm is identical in both cases. In other words, the value of $t(k)$ depends on the set of vertices which are assigned the numbers $k, k+1, \cdots, n$ and not on the edges connecting these vertices. Since this statement is true for all values of k, we get the following result which is more general than Theorem 1.

Theorem 7

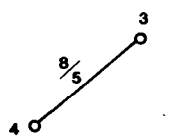
Let G be a connected undirected graph. Let the vertices of G be numbered as $v_1^1, v_2^1, \cdots, v_n^1$ according to one initial spanning tree and as $v_1^2, v_2^2, \cdots, v_n^2$ according to another initial spanning tree. Let $V_i^1 = \{v_i^1, v_{i+1}^1, \cdots, v_n^1\}$ and $V_i^2 = \{v_i^2, v_{i+1}^2, \cdots, v_n^2\}$, for $2 \leqslant i \leqslant n$. If $V_i^1 = V_i^2$ for all i, $2 \leqslant i \leqslant n$, then the number of non-tree sequences generated by Char's algorithm will be the same for both initial spanning trees. In other words, $t + t_0$ will be the same for all choices of initial spanning trees which have identical V_i^1 and V_i^2 for every i, $2 \leqslant i \leqslant n$. $\square$



(a)



(b)



(c)

Fig. 1. (a) Graph $G = G_1$. (b) Graph G_2. (c) Graph G_3.

Consider a graph $G \in G^{(n-1)}$ (defined in Theorem 2) in which vertex x is of degree $n-1$. Let G_1 be any arbitrary spanning tree of G and G_2 be a star tree having the vertex x as the star vertex. Suppose we assign the number n to vertex x and number the other vertices using G_1 and satisfying the tree compatibility property; then an identical numbering (satisfying Theorem 7) of the vertices of G using G_2 and satisfying the tree compatibility property is possible. Thus if t_0 and t_0' are the numbers of non-tree sequences generated with respect to G_1 and G_2, then $t_0 = t_0'$. By Theorem 2 $t_0' \leqslant t$, where t is the number of spanning trees of G and so $t_0 \leqslant t$. Since the above arguments are valid for any arbitrary G_1 chosen as the initial spanning tree, we get the following theorem which is more general than Corollary 6.1.

Theorem 8

For an n-vertex connected graph with maximum degree $n-1$, $t_0 \leqslant t$ for any choice of the initial spanning tree in which a vertex with degree $n-1$ is assigned the number n. $\square$

Using Theorems 3 and 5 we can give simpler proofs of the following two interesting results which have been reported in [5].

Theorem 9

If L_n is the number of spanning trees of an n-vertex ladder[1] and L_n^0 is the number of non-tree sequences generated by Char's algorithm when a star tree is chosen as the initial spanning tree, then

$$L_n^0 = L_{n-1}$$

and

$$\lim_{n\to\infty} \frac{L_n^0}{L_n} \approx 0.382. \qquad \square$$

Theorem 10

If W_n is the number of spanning trees of an n-vertex wheel and W_n^0 is the number of non-tree sequences generated by Char's algorithm when a star tree is chosen as the initial spanning tree, then

$$W_n^0 = 1 + L_n$$

and

$$\lim_{n\to\infty} \frac{W_n^0}{W_n} \approx 0.4472. \qquad \square$$

Next we consider the question of computing the total number of computational steps required in the execution of Char's algorithm. Consider a sequence $\lambda = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1), x, \text{REF}(k+1), \cdots, \text{REF}(n-1))$, with $x \neq \text{REF}(k)$, generated by the algorithm. This sequence belongs to $T_k \cup T_k'$. To generate this sequence the algorithm explicitly requires setting $\text{DIGIT}(i) = \text{REF}(i)$ for each i, $k+1 \leqslant i \leqslant n-1$, in addition to setting $\text{DIGIT}(k) = x$. Then λ is tested for the tree compatibility property at position k. Thus generating and testing λ involves the following two types of computational steps.

Type 1: $(n-k-1)$ steps to set $\text{DIGIT}(i) = \text{REF}(i)$, $k+1 \leqslant i \leqslant n-1$.

Type 2: C_k steps to set $\text{DIGIT}(k) = x$ and to test λ for the tree compatibility property.

Suppose λ is a tree sequence. Then the cost of Type 1 computation required to generate λ can be associated with λ. If, on the other hand, λ is a non-tree sequence, then the algorithm generates a new sequence λ' by setting $\text{DIGIT}(k)$ to the vertex next to x in the adjacency list of k. Note that generating λ' does not require Type 1 computation. If λ' also fails the test, the algorithm continues to generate sequences (without using Type 1 computation) until a tree sequence λ'' is generated. The cost of the Type 1 computation required in generating λ'' can therefore be charged to the tree sequence λ. Thus the cost of each Type 1 computation can be charged to a tree sequence. Clearly the cost of Type 1 computations (in terms of computational steps) for generating all the tree sequences in T_k is given by $|T_k|(n-k-1)$. If we denote by COST1 the total cost of Type 1 computations required in generating all the tree sequences, then

$$\text{COST1} = \sum_{k=1}^{n-1} |T_k|(n-k-1) = \sum_{k=1}^{n-2} |T_k|(n-k-1).$$

But $|T_k| = t(k+1) - t(k)$ for all k, $1 \leqslant k \leqslant n-2$. So

$$\text{COST1} = \sum_{k=1}^{n-1} [t(k+1) - t(k)](n-k-1)$$

$$= t \sum_{k=2}^{n-1} \left[\frac{1}{d_{n-1} d_{n-2} \cdots d_k} \right] - (n-2). \qquad (4)$$

As regards Type 2 computation, it is required for each sequence in $T_k \cup T_k'$, $1 \leqslant k \leqslant n-1$. If C_k^m denotes the maximum number of computational steps required to perform Type 2 computation for any sequence in $T_k \cup T_k'$, and COST2 denotes the cost of performing all the Type 2 computations, then

$$\text{COST2} \leqslant \sum_{k=1}^{n-1} C_k^m |T_k \cup T_k'| = t \sum_{k=1}^{n-1} \left[\frac{\deg(k)-1}{d_{n-1} d_{n-2} \cdots d_k} \right] C_k^m. \qquad (5)$$

From (4) and (5), it is clear that $\text{COST1} \leqslant nt$ and $\text{COST2} \leqslant n^3 t$. So the total cost of execution of Char's algorithm is $O(n^3 t)$. A better bound for COST2 does not appear to be possible, even though it has been found in a large number of cases that $\text{COST2} \leqslant nt$. For example, for all the graphs in $G^{(n-1)}$, COST2 is $O(nt)$ and hence the total cost is $O(nt)$ in such cases.

IV. Heuristics for Choice of Initial Spanning Tree

Since t_0 and the complexity of Char's algorithm depends on the initial spanning tree, we now consider the problem of choosing the initial spanning tree which leads to a minimum t_0. The initial spanning tree can be obtained by performing a Breadth-First Search (BFS) or a Depth-First Search (DFS) on the given graph. The implementation given in [5] selects the initial spanning tree by performing a BFS starting at a vertex of maximum degree. In this section we consider the question of using DFS for selecting the initial spanning tree, with the objective of minimizing t_0. For results relating to DFS, see [6].

Let T_{DFS} denote a DFS tree of the given graph G. Starting at the root of T_{DFS}, let the vertices of G be numbered as $n, n-1, \cdots, 1$, in the order in which they are visited during the DFS. With such a numbering, T_{DFS} will clearly satisfy the tree compatibility property. It should be noted that each ancestor of k in T_{DFS} will have a number greater than k and each descendant of k will have a number less than k. Furthermore, there are no cross edges in G [6]. In other words, if x and y are two vertices such that neither of them is a descendant of the other in T_{DFS}, then the edge (x, y) is not in G. Using these observations, we can prove the following.

Theorem 11

If vertex k is a leaf in T_{DFS}, then $|T_k'| = 0$. $\square$

[1]A ladder is also known as a fan [8].

Theorem 12

If δ_k is the number of descendants of vertex k in T_{DFS}, then

$$C_k^m \leqslant \delta_k. \qquad \square$$

From Theorem 5, it is clear that if the vertices of the graph G could be numbered in such a way that $\deg(n-1), \deg(n-2), \cdots, \deg(1)$ are in the ascending order and $d_{n-1}, d_{n-2}, \cdots, d_1$ are in the descending order, then $(t+t_0)$ will be reduced considerably. Since $\deg(n)$ does not appear in the expression for $(t+t_0)$, we can number the vertex having the maximum degree in G as n. In other words, we can start the DFS to find the initial spanning tree at a vertex of maximum degree.

Let $\Gamma'(i)$ be the set of ancestors of vertex i in T_{DFS} which are adjacent to i in G and let $d_i' = |\Gamma'(i)|$. To find the numbers $d_1, d_2, \cdots, d_{n-1}$, we start with the graph G_1 obtained from G by assigning unit weight to each edge of G. Recall that d_i is the sum of the weights of the edges incident on i in the graph G_i which is obtained from G_1 by performing pivotal condensation at the vertices $1, 2, \cdots, i-1$. Since pivotal condensation does not reduce the weight of any edge connecting i to any vertex in $\Gamma'(i)$, and since each such edge has a weight of value at least one, it follows that

$$d_i \geqslant d_i', \qquad 1 \leqslant i \leqslant n-1.$$

It is evident from Theorems 11 and 12 and the above discussions that t_0 could be reduced considerably if we do the following.

1) Maximize the number of leaves in T_{DFS}.
2) Maximize the number of ancestors of each vertex during the DFS.
3) Minimize the number of descendants δ_k, for each k.

To achieve the above objectives, we suggest the following two heuristics for selecting the initial spanning tree using DFS.

Heuristic 1: Start the DFS at a vertex of maximum degree. During the search, when we are at vertex i, choose, from among the neighbors of i, the one having the maximum number of ancestors in the tree developed so far. If more than one vertex has this property, then choose, from among these vertices, the one having minimum degree in G.

Heuristic 2: Start the DFS at a vertex of maximum degree. During the search, when we are at vertex i, choose, from among the neighbors of i, the one having minimum degree in G. If more than one vertex has this property, then choose, from among these vertices, the one having the maximum number of ancestors in the tree developed so far.

We have implemented Char's algorithm using each one of the above two heuristics as well as BFS. In Table I we give the number of non-tree sequences generated in these cases for ten randomly generated graphs. From Table I it is clear that the heuristics considerably reduce the number of non-tree sequences generated by the algorithm and that the two heuristics result in approximately the same number of non-tree sequences.

TABLE I
NUMBER OF NON-TREE SEQUENCES GENERATED

Graph	Number of Spanning Trees	Number of Non-Tree Sequences		
		BFS	Heuristic 1	Heuristic 2
G_1	24672	20738	14412	14438
G_2	13931	8778	5308	5310
G_3	151662	120259	66079	67036
G_4	151719	90831	65657	65950
G_5	1360710	1112223	504279	481506
G_6	12897990	7559568	7136979	6971221
G_7	1592512	871944	528193	528128
G_8	1820488	1151321	634183	635357
G_9	14689650	11998877	6179924	6207721
G_{10}	26520950	20921468	9096476	8941338

V. PATH COMPRESSION

The heuristics for the selection of the initial spanning tree discussed in the previous section are aimed at reducing both COST1 and COST2 (defined in Section III). Though the number of comparisons required in a straightforward implementation is influenced by the initial spanning tree chosen, the actual number of comparisons done during the execution of the algorithm can be reduced considerably by an appropriate choice of a data structure for maintaining the information relating to a tree sequence. In this section we discuss a method to achieve this.

Consider a sequence $\lambda = (\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1), x, \text{REF}(k+1), \cdots, \text{REF}(n-1))$, $x \neq \text{REF}(k)$, generated by Char's algorithm. Let G_λ be the corresponding subgraph of the given graph G. Let G' be the subgraph obtained by removing from G_λ the edge (k, x). To test whether λ is a tree sequence or not, Char's algorithm traverses the sequence of vertices k, x, $\text{DIGIT}(x)$, $\text{DIGIT}(\text{DIGIT}(x)), \cdots$ until the vertex k or a vertex $j > k$ is encountered. In the latter case, λ is a tree sequence. Suppose λ is a tree sequence. Let P denote the path k, x, $\text{DIGIT}(x)$, $\text{DIGIT}(\text{DIGIT}(x)), \cdots, j$.

After generating and identifying the tree sequence λ, the algorithm proceeds to generate sequences in which $\text{DIGIT}(1), \text{DIGIT}(2), \cdots, \text{DIGIT}(k-1)$, and $\text{DIGIT}(k)$ are the same as in λ. So the path P will be present in all the subgraphs corresponding to such sequences. Consider now one such sequence λ' which is to be tested for the tree compatibility property at position i. Clearly $i > k$. Let $\text{DIGIT}(i) = \alpha$ in λ'. Then to test λ' for the tree compatibility property, we need to traverse the sequence P' of vertices i, α, $\text{DIGIT}(\alpha)$, $\text{DIGIT}(\text{DIGIT}(\alpha)), \cdots$ until vertex i or a vertex greater than i is encountered. If k lies on P', then the sequence of vertices k, x, $\text{DIGIT}(x)$, $\text{DIGIT}(\text{DIGIT}(x)), \cdots, j$ representing P will be a subsequence of P'. Thus while traversing P', we can proceed to j directly from k using the path P. In other words, we can effectively compress P' if we keep track of the information relating to the path P. This technique, called path compression [9], will considerably reduce the actual number of

TABLE II
NUMBER OF COMPARISONS MADE

Graph	Number of Spanning Trees	Number of Non-Tree Sequences	Number of Comparisons	
			Heuristic 1	Heuristic 1 With Path Compr.
G_1	24672	14412	110374	83342
G_2	13931	5308	40711	33811
G_3	151662	66079	593753	442127
G_4	151719	65657	565449	434929
G_5	1360710	504279	5323910	3841745
G_6	12897990	7136979	64931380	48970813
G_7	1592512	528193	5599546	4080411
G_8	1820488	634183	6749935	4808779
G_9	14689650	6179924	66484647	45516982
G_{10}	26520950	9096476	102984126	69897903

TABLE III
EXECUTION TIME

Graph	Number of Spanning Trees	Execution Time in Seconds		
		BFS	Heuristic 1	Heuristic 1 With Path Compr.
G_1	24672	2.037	1.924	1.767
G_2	13931	0.970	0.944	0.928
G_3	151662	12.495	10.167	9.462
G_4	151719	10.661	10.489	10.205
G_5	1360710	102.660	87.835	81.612
G_6	12897990	994.735	966.608	894.635
G_7	1592512	113.124	105.868	99.491
G_8	1820488	129.747	124.721	115.582
G_9	14689650	1193.974	1026.815	946.918
G_{10}	26520950	2264.015	1822.345	1662.247

comparisons made during the execution of Char's algorithm.

To implement Char's algorithm with path compression, we use a new array NEXTVERTEX. Whereas the DIGIT array keeps the adjacency information of each sequence, the NEXTVERTEX array, for a tree, is defined as NEXTVERTEX(i) = j, where $j > i$ is the first vertex reachable from vertex i as we traverse the tree from vertex i to vertex n. We create and maintain the NEXTVERTEX array as follows. Since for the initial tree sequence, DIGIT(i) = REF(i) > i, $1 \leqslant i \leqslant n-1$, we initialize NEXTVERTEX(i) = REF(i), $1 \leqslant i \leqslant n-1$. Whenever a tree sequence λ = (DIGIT(1), DIGIT(2), $\cdots$, DIGIT($k-1$), x, REF($k+1$), $\cdots$, REF($n-1$)) is generated by changing the value of DIGIT(k) of the previous tree sequence, the NEXTVERTEX array is updated as follows.

Update 1: NEXTVERTEX(i) = REF(i), $k+1 \leqslant i \leqslant n-1$.

Update 2: NEXTVERTEX(k) = j, where $j > k$ is the first vertex reachable from vertex k in the tree path from vertex k to vertex n.

Note that j will be known when the tree compatibility test for λ is completed.

We have implemented Char's algorithm with path compression using NEXTVERTEX array. In Table II we give the total number of comparisons made by the implementation of Char's algorithm with Heuristic 1 and the implementation with Heuristic 1 and path compression. From Table II it is clear that the use of path compression considerably reduces the total number of comparisons.

Next we compute the number of computational steps required to create and update the NEXTVERTEX array. Note that initially NEXTVERTEX($n-1$) = REF($n-1$) = n. Since Update 2 sets NEXTVERTEX($n-1$) to the first vertex greater than $n-1$ in the tree path from vertex $n-1$ to vertex n, NEXTVERTEX($n-1$) is always equal to n and so we need to update only NEXTVERTEX(i), $1 \leqslant i \leqslant n-2$. For each tree sequence of the form λ_k = (DIGIT(1), DIGIT(2), $\cdots$, DIGIT($k-1$), x, REF($k+1$), $\cdots$, REF($n-1$)) with $x \neq$ REF(k), Update 1 requires $(n-k-1)$ assignments and Update 2 requires exactly one assignment. Thus Update 1 and 2 together require $(n-k)$ computational steps for each tree sequence of the form λ_k. The number of tree sequences of the form λ_k is given by $t(k+1)-t(k)$, where $t(i)$, $1 \leqslant i \leqslant n-2$, is the number of spanning trees of the graph $G_k^{(s)}$ defined in Section III. Thus the total number of computational steps required to create and update the NEXTVERTEX array is given by

$$n-1+\sum_{k=1}^{n-2}[t(k+1)-t(k)](n-k)$$

$$=2t(n-1)+\sum_{k=2}^{n-2}t(k), \qquad \text{since } t(1)=1.$$

$$=t\left[\frac{1}{d_{n-1}}+\sum_{k=2}^{n-1}\frac{1}{d_{n-1}d_{n-2}\cdots d_k}\right]$$

which is of order $O(nt)$. Thus employing path compression in the implementation of Char's algorithm does not change the complexity of the algorithm.

Since the total number of comparisons are reduced when Char's algorithm is implemented with path compression, the execution time of the algorithm with path compression should also be less than the execution time of the algorithm without path compression. This can be verified from Table III where we have tabulated the execution times (on a CDC Cyber 835 computer) for three implementations of Char's algorithm—Char's implementation where Breadth-First Search (BFS) is used to select the initial spanning tree, implementation using Heuristic 1, and implementation using Heuristic 1 and path compression.

VI. SUMMARY AND CONCLUSION

In this paper we have presented a detailed computational complexity analysis—both theoretical and experimental—of Char's algorithm to enumerate all the spanning trees of a graph and have also presented several quantitative and qualitative properties of the algorithm.

We have described methods to speed up the algorithm. In particular, we have discussed two heuristics to choose the initial spanning tree which lead to a minimum number of spanning non-tree subgraphs. We have also described a

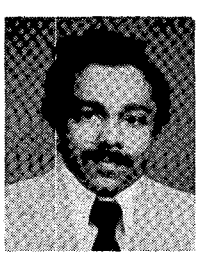
technique called path compression to reduce the number of comparisons. Theoretical analysis and experimental results presented in this paper establish the superiority of Char's algorithm when implemented using the heuristics and path compression.

ACKNOWLEDGMENT

The authors thank the reviewers for their suggestions which have resulted in considerable improvement in the presentation of the paper.

REFERENCES

[1] G. J. Minty, "A simple algorithm for listing all the spanning trees of a graph," *IEEE Trans. Circuit Theory*, vol. CT-12, p. 120, 1965.
[2] R. C. Read and R. E. Tarjan, "Bounds on backtrack algorithms for listing cycles, paths and spanning trees," *Networks*, vol. 5, pp. 237–252, 1975.
[3] H. N. Gabow and E. W. Myers, "Finding all spanning trees of directed and undirected graphs," *SIAM J. Comput.*, vol. 7, pp. 280–287, 1978.
[4] J. P. Char, "Generation of trees, two-trees and storage of master forests," *IEEE Trans. Circuit Theory*, vol. 15, pp. 128–138, 1968.
[5] R. Jayakumar, "Analysis and study of a spanning tree enumeration algorithm," M.S. thesis, Dep. of Computer Science, Indian Inst. of Technol., Madras, India, 1980. Also reported in "Combinatorics and graph theory," Springer-Verlag Lecture Notes on Mathematics, (S. B. Rao, Ed.) no. 885, 1981, pp. 284–289.
[6] M. N. S. Swamy and K. Thulasiraman, *Graphs, Networks, and Algorithms*. New York: Wiley-Interscience, 1981.
[7] M. N. S. Swamy and K. Thulasiraman, "A Theorem in the Theory of Determinants and the Number of Spanning Trees of a Graph," *Canadian Elect. Eng. J.* vol. 8, no. 4, pp. 147–152, 1983.
[8] A. J. W. Hilton, "The number of spanning trees of labelled wheels, fans, and baskets," in *Combinatorics*, published by *Inst. Math. Appl.*, 1972, pp. 203–206.
[9] R. E. Tarjan, "Applications of path compression on balanced trees," *J. Ass. Comput. Mach.*, vol. 26, pp. 690–715, 1979.

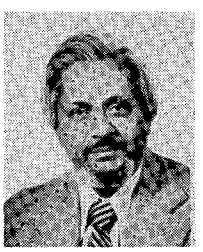


R. Jayakumar was born in Tamilnadu, India, on February 10, 1955. He received the Bachelor's degree in electrical engineering from the University of Madras, Madras, India, in 1977, and the M.S. degree in computer science from the Indian Institute of Technology, Madras, India, in 1980. He is now working towards his Ph.D. degree at Concordia University, Montreal, P.Q., Canada. His research interests are in graph theory, analysis of algorithms, VLSI layout, and data structures.

K. Thulasiraman was born in Ammayappan, Tamilnadu, India, on June 9, 1942. He received the Bachelor's and Master's degrees in electrical engineering from the University of Madras, Madras, India, in 1963 and 1965, respectively, and the Ph.D. degree in electrical engineering from the Indian Institute of Technology, Madras, in 1968.

He joined the Indian Institute of Technology, Madras, in 1965, where he was associated with the Department of Electrical Engineering from 1965 to 1973 and with the Department of Computer Science from 1973 to 1981. He was promoted to the rank of Professor in January 1977. After serving for a year (1981–1982) at the Department of Electrical Engineering, Technical University of Nova Scotia, Halifax, Canada, he joined Concordia University, Montreal, as Professor at the Department of Mechanical Engineering where he is now involved in the development of programs in Industrial Engineering at the undergraduate and graduate levels. Earlier, he had held visiting positions at Concordia University during the periods of 1970–1972, 1975–1976, and 1979–1980. He has published over 50 technical papers on different aspects of Electrical Network Theory, Graph Theory and Design and Analysis of Graph Algorithms. He has also coauthored the book *Graphs, Networks and Algorithms* (New York: Wiley-Interscience, 1981). His current research interests are in Network Theory, Analysis of Algorithms, Computer Networks and VLSI Layout.

Dr. Thulasiraman is a Fellow of the Institution of Telecommunication Engineers, India.



M. N. S. Swamy (S'59–M'62–SM'74–F'80) was born on April 7, 1935. He received the B.Sc. (honors) degree in mathematics from Mysore University, Mysore, India, in 1954, the Diploma in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1957, and the M.Sc and Ph.D. degrees in electrical engineering from the University of Saskatchewan, Saskatoon, Sask., Canada, in 1960 and 1963, respectively.

He worked as a Senior Research Assistant at the Indian Institute of Science until 1959, when he began graduate study at the University of Saskatchewan. In 1963 he returned to India to work at the Indian Institute of Technology, Madras. From 1964 to 1965 he was an Assistant Professor of Mathmematics at the University of Saskatchewan. He has also taught as a Professor of Electrical Engineering at the Technical University of Nova Scotia, Halifax, N.S., Canada, and the University of Calgary, Calgary, Alta., Canada. He was chairman of the Department of Electrical Engineering, Concordia University (formerly Sir George Williams University), Montreal, P.Q., Canada, From July 1970 until August 1977, when he became the Dean of Engineering and Computer Science of the same university. He has published a number of papers on number theory, semiconductor circuits, control systems, and network theory. He is a coauthor of the book *Graphs, Networks and Algorithms* (New York: Wiley-Interscience, 1981).

Dr. Swamy is a Fellow of several professional societies including the Engineering Institute of Canada, the Institution of Engineers (India), the Institution of Electrical Engineers (U.K.), and the Institution of Electronics and Telecommunications Engineering (India). He is Associate Editor of the *Fibonacii Quarterly* and the new journal, *Circuits, Systems and Signal Processing*. He was Vice-President of the IEEE Circuits and Systems Society in 1976, and General Chairman of the International Symposium on Circuits and Systems, 1984.