Counting Trees

George Havas[1]
School of Information Technology,
The University of Queensland.

Dean G. Hoffman[2]
Department of Discrete and Statistical Sciences,
Auburn University.

Colin Ramsay[3]
School of Information Technology and Department of Mathematics,
The University of Queensland.

This is an updated version of the paper which appeared in *Research on Combinatorial Algorithms* (Proc. Eighth Australasian Workshop on Combinatorial Algorithms), Queensland University of Technology (1997) 1–10, (correcting an error in one of the tables and updating the bibliography).

Abstract: We consider the problem of enumerating particular kinds of spanning trees of a graph. The problem is interesting in its own right and has applications in the analysis of graph algorithms. The general problem seems hard to resolve. We present an algorithm to list depth-first trees for moderate sized graphs. We use this algorithm to count the number of depth-first spanning trees in a variety of graphs, and we give the results for hypercubes, meshes and tori, and complete bipartite graphs. We also give a proof of a theoretical result for complete multipartite graphs.

1 Introduction

Depth-first search is a fundamental operation in graph traversal algorithms. There is active research on these basic operations in the context of distributed systems, see [8, 9, 10] for example. To properly analyse the more recent algorithms we need to count the number of depth-first spanning trees. This leads us to the following problems.

Suppose that we are given a graph G, a root node r, and a list of all the spanning trees of G rooted at r. How many of these trees have a traversal that is a depth-first spanning tree of G? Given all the depth-first spanning trees of G, how many distinct traversals are there?

[1]Research supported by the Australian Research Council
[2]Research supported by UQ Quality Grant
[3]Research supported by an APA Scholarship

We consider these problems for some families of graphs. We present an algorithm for counting depth-first spanning trees and provide some results for moderate sized graphs. We provide a theoretical result for an infinite class of graphs.

2 The problems

In order to analyse distributed graph algorithms we wish to properly understand the nature of depth-first spanning trees. In [8, 9, 10] we see analyses which involve a parameter which ranges from zero to one. More accurate determination of this parameter is done empirically in some cases. Improved knowledge about these trees would help in better estimating the parameter. The first step in this direction is to enumerate the particular types of trees. Given our initial motivation we are particularly interested in solving these problems for the types of graphs which are used in distributed systems, but the problems have independent theoretical interest.

We use $G = (V, E)$ to denote a generic graph, and set $p = |V|$ and $q = |E|$. All the graphs we consider have $p > 2$, and have no self-loops or multiple-edges. The nodes are labelled $0 \cdots p-1$, with some $r \in V$ distinguished as the root node. The edges may be directed or undirected, and all nodes must be reachable from the root node.

We are interested in counting, or listing, the following.

$ST(G, r)$**:** The number of spanning trees. If G is undirected, the choice of root is immaterial. If G is a digraph, then this is the number of directed spanning trees rooted at r.

$DT(G, r)$**:** The number of spanning trees rooted at r that have at least one depth-first traversal.

$DF(G, r)$**:** Given the spanning trees that have at least one depth-first traversal rooted at r, this is the total number of distinct traversals.

$HP(G, r)$**:** The number of Hamiltonian paths starting at r. These are simply the number of depth-first spanning trees which are paths.

$HC(G, r)$**:** The number of Hamiltonian cycles. This is independent of r. Easy obtained from the Hamiltonian paths, by checking if we can return to r from the final node. Note that if G is undirected this count includes each Hamiltonian cycle twice; once in each direction.

Where r or G is clear from the context, we may abbreviate our notation to, for example, $ST(G)$ or simply ST.

3 Background

Since a tree is acyclic, the spanning trees of a graph can be generated by considering all ways of breaking all cycles. This idea was introduced by Kirchoff in 1847 (see the translation by O'Toole [5]), and is discussed in any standard text (see, for example, Gibbons [4] or Even [1]).

Given an undirected graph $G = (V, E)$, define the $p \times p$ *degree matrix* $D = [d_{i,j}]$ by

$$d_{i,j} = \begin{cases} \text{degree}(i) & \text{if } i = j, \\ -1 & \text{if } ij \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Then $ST(G, r)$ is given by the value of the minor of entry $d_{i,i}$, for any $1 \leq i \leq p$; that is, delete the ith row and column of D then take the determinant.

If G is a digraph, define the $p \times p$ *in-degree matrix* $D = [d_{i,j}]$ by

$$d_{i,j} = \begin{cases} \text{in-degree}(i) & \text{if } i = j, \\ -1 & \text{if } ij \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Then $ST(G, r)$ is given by the value of the minor of entry $d_{r,r}$.

It is a standard result, due to Cayley [2, 3], that the number of spanning trees of K_n, the complete undirected graph on n vertices, is n^{n-2}. Given a spanning tree T of K_n and a choice of root, there is a unique orientation of T to obtain a spanning tree of the complete directed graph on n nodes, K_n^d. So K_n^d has a total of n^{n-1} spanning trees, and for each choice of r there are n^{n-2} spanning trees.

Each depth-first traversal of G induces a permutation of $\{0, \ldots, p-1\}$, and any permutation represents at most one depth-first traversal, so there are at most $(p-1)!$ depth-first traversals rooted at r.

So, noting that our restrictions on G and r ensure that $DT \geq 1$, it must be the case that

$$p^{p-2} \geq ST \geq DT \geq HP \geq HC \geq 0 \quad \text{and} \quad (p-1)! \geq DF \geq DT \geq 1$$

for all G, r. Note also that ST and DF are incomparable. All of these inequalities are best possible, as our results will illustrate.

4 An Algorithm

Spanning trees can be counted using the degree or in-degree matrix method of the previous section, while all spanning trees can be listed using, for example, an algorithm of Tarjan [12]. Hamilton paths and cycles can be counted and listed by a simple backtrack algorithm.

We could obtain DT by generating spanning trees and testing the edges of G not in the tree (see Tarjan [11]). However, it is more efficient to generate depth-first trees directly. To obtain DT and DF we introduce, into a standard backtracking algorithm, the concept of a *backcall*.

The algorithm is given, using a C-like notation, in Figure 1. The global data used by the algorithm is: `p`, the number of nodes; `r`, the index of the root node; `am[][]`, the graph's adjacency matrix; `tick[]`, whether or not a node has been visited yet; and `dft[]`, the current depth-first traversal. After the data structures have been initialised and the graph read-in, we set `dft[0] = r` and `tick[r] = 1`, and then start

```
 1  void df(int curr, int size)
 2  { if (size == p)
 3    { ... process another depth-first spanning tree ... }
 4    else if (... curr has any unvisited children ...)
 5    { for (... each unvisited child i ...)
 6      { dft[size] = i;
 7        tick[i] = 1;
 8        df(i,size+1);
 9        tick[i] = 0;
10    } }
11    else
12    { ... i = most recently visited node with unvisited children ...
13      df(i,size);
14  } }
```

Figure 1: The backcalling algorithm for DF.

the algorithm with the call `df(r,1)`. The arguments to `df()` are the current node `curr` and the current tree size `size`.

Lines 2–3 handle the case where the current node is the final leaf node of a depth-first tree and lines 4–10 the case where we must go deeper. If neither of these two cases holds then we would normally backtrack, rescinding the current node. However, for a depth-first search, we retain the current partial tree and find the most recently visited node from which we can still go deeper. Note that such a node always exists, and is unique. In line 13 we backcall to this node; this retains the current tree (and its size) but resets the current node. When we return from the backcall, we backtrack normally.

This algorithm gives all ways of traversing those spanning trees that have depth-first traversals (that is, it yields DF), and works for both undirected and directed graphs.

Now, if G in undirected, the subtrees of a node of a depth-first spanning tree are disjoint. This allows us to enumerate DT via a simple modification to the algorithm. For each node j we maintain a list of the children of j which have been visited from j. In line 5 we only iterate over those children that have a higher label than all the currently visited children; that is, we impose an ordering on the visiting of children. Note that this strategy is *not* valid for digraphs.

These algorithms are based on the following characterisation of depth-first trees. Let T be a spanning tree of G. Then T is a depth-first search tree of G rooted at r if and only if it has the following property: For every edge uv of G, either u is on the unique path in T from r to v, or v is on the unique path in T from r to u.

It is easy to see that a depth-first search tree must have this property since, during a depth-first traversal, we do not backtrack from a vertex until all its neighbors have already been visited. Conversely, given a tree T with the indicated property, for every vertex of T, order its children arbitrarily, and traverse T by always choosing the smallest child available whenever there is a choice. The resulting depth-first

G, r	ST	DT	HP	HC	DF
P_n, 0	1	1	1	0	1
P_n, 1	1	1	0	0	2
C_n, -	n	2	2	2	2
S_n, 0	1	1	0	0	$(n-1)!$
S_n, 1	1	1	0	0	$(n-2)!$
K_n, -	n^{n-2}	$(n-1)!$	$(n-1)!$	$(n-1)!$	$(n-1)!$
$T_{n,d}$, 0	1	1	0	0	$(n!)^{1+n+\cdots+n^{d-1}}$

Table 1: Some test cases.

traversal of T will be a legal depth-first search of G.

In particular, the number of depth-first traversals of T is the number of such orderings, namely the product over all vertices v of

$$(\text{the number of children of } v \text{ in } T)! \; .$$

5 Results

To check that the programme functioned correctly, we tested it against some cases where the values of ST et al. are readily calculated. The test cases are summarised in Table 1.

P_n, $n \geq 3$, is the path of length $n-1$ on n vertices, with the vertices labelled in order $0, \ldots, n-1$; note the two choices for r. C_n, $n \geq 3$, is the cycle of length n on n vertices; the choice of root is irrelevant. S_n, $n \geq 4$, is the star on n vertices, with the vertex of degree $n-1$ labelled 0; note the two choices for r. It is obvious that

$$DF(S_n, 0) = (n-1)! \quad \text{and} \quad DF(S_n, 1) = (n-2)!$$

K_n, $n \geq 3$, is the complete graph on n vertices, with $q = n(n-1)/2$; the choice of root is irrelevant. Recall that $ST(K_n) = n^{n-2}$, and note that

$$DT(K_n) = HP = HC = DF = (n-1)!$$

The complete n-ary tree of depth d (that is, all paths from the root to a leaf have length d) is denoted by $T_{n,d}$, with the root node labelled 0. Now $DF(T_{n,1}, 0) = n!$ and

$$DF(T_{n,d+1}, 0) = n!(DF(T_{n,d}, 0))^n.$$

This yields

$$DF(T_{n,d}, 0) = (n!)^{1+n+\cdots+n^{d-1}}.$$

In the following subsections we present some of the results obtained using our algorithm and a theoretical result.

5.1 Hypercubes

Here Q_n is the n-dimensional hypercube. Each node has degree n, so $p = 2^n$ and $q = n2^{n-1}$. The nodes are labelled with n-bit binary strings, and two nodes are

G	ST	DT	HP	HC	DF
Q_2	4	2	2	2	2
Q_3	384	24	18	12	30
Q_4	42467328	15984	5712	2688	32592
Q_5	$\approx 2.08 \times 10^{19}$	80395896960	5859364320	1813091520	423889362240
Q_2^+	2	2	0	0	2
Q_3^+	24	12	0	0	12
Q_4^+	20736	576	0	0	576
Q_5^+	309586821120	1658880	0	0	1658880
Q_6^+	$\approx 1.15 \times 10^{28}$	$\approx 1.65 \times 10^{13}$	0	0	$\approx 1.65 \times 10^{13}$
Q_3^c	32400000	6144	84	12	37732

Table 2: Results for hypercubes.

adjacent if and only if they differ in exactly one binary-digit. We set $r = 0$, although all nodes are equivalent. Note that when counting DT, HP, HC and DF we can fix the first two nodes after r (but no more), and then multiply the count by $n(n-1)$. The results for $2 \leq n \leq 5$ are given in Table 2.

If we now direct each edge from the lower to the higher of its node labels, we obtain Q_n^+, the directed hypercube. This has precisely one source (node 0) and precisely one sink (node $p-1$), and we set $r = 0$. Note that Q_n^+ is isomorphic to the poset of subsets of an n-set.

Now the in-degree matrix of Q_n^+ is upper triangular, and each value of i, $0 \leq i \leq n$, appears $\binom{n}{i}$ times on the main diagonal. Deleting the row and column for zero, and calculating the determinant, we obtain

$$ST(Q_n^+, 0) = \Pi_{i=1}^n i^{\binom{n}{i}}.$$

Let P_2^+ be the directed path of length 1. Then $Q_{n+1}^+ = P_2^+ \times Q_n^+$, so we see that

$$DT(Q_n^+) = DF(Q_n^+) \quad \text{and} \quad DF(Q_n^+) = n(DF(Q_{n-1}^+))^2.$$

Since $DF(Q_2^+) = 2$, this yields

$$DF(Q_n^+) = \Pi_{i=0}^{n-2}(n-i)^{2^i}.$$

These values are illustrated for $2 \leq n \leq 6$ in Table 2.

The cycle-connected hypercube Q_n^c is obtained from Q_n by replacing each vertex with an n-cycle. So $p = n2^n$ and $q = 3n2^{n-1}$. Note that $q/p = 3/2$ in Q_n^c, while in Q_n this ratio is $n/2$. The results for Q_3^c are given in Table 2.

5.2 Meshes and Tori

The $m \times n$ mesh is $P_m \times P_n$. We consider this a Cartesian grid with $(0,0)$ at the lower left and $(n-1, m-1)$ at the upper right. Up to symmetry, we try all choices for the root. The results for $2 \leq m \leq n \leq 4$ are given in Table 3.

The $m \times n$ torus is $C_m \times C_n$. Up to symmetry, there is only one choice for the root. Note that $2 \times m$ tori are somewhat of an abuse, since they contain multiple edges. So

G, r	ST	DT	HP	HC	DF
$P_2 \times P_2$, $(0,0)$	4	2	2	2	2
$P_2 \times P_3$, $(0,0)$	15	4	3	2	5
$(1,0)$	15	3	2	2	4
$P_2 \times P_4$, $(0,0)$	56	8	4	2	12
$(1,0)$	56	6	3	2	9
$P_3 \times P_3$, $(0,0)$	192	14	8	0	20
$(0,1)$	192	11	0	0	22
$(1,1)$	192	8	8	0	8
$P_3 \times P_4$, $(0,0)$	2415	50	17	4	93
$(0,1)$	2415	39	4	4	106
$(1,0)$	2415	39	6	4	94
$(1,1)$	2415	30	12	4	48
$P_4 \times P_4$, $(0,0)$	100352	322	52	12	844
$(0,1)$	100352	251	25	12	869
$(1,1)$	100352	200	36	12	544

Table 3: Results for meshes.

G	ST	DT	HP	HC	DF
$P_2 \times P_2$	4	2	2	2	2
$P_2 \times C_3$	75	12	10	6	14
$P_2 \times C_4$	384	24	18	12	30
$P_2 \times C_5$	1805	48	26	10	70
$C_3 \times C_3$	11664	264	168	96	376
$C_3 \times C_4$	367500	1504	688	252	2496
$C_3 \times C_5$	10609215	8544	2432	780	18624
$C_4 \times C_4$	42467328	15984	5712	2688	32592
$C_4 \times C_5$	4381392020	168076	28716	5860	463764
$C_5 \times C_5$	1562500000000	3226368	236456	47160	13099528

Table 4: Results for tori.

we collapse multiple edges into single edges, and use $P_2 \times C_n$ instead. This does not affect our results, since we are concerned with trees. The results for $2 \leq m \leq n \leq 5$ are given in Table 4. Note that $P_2 \times P_2$, $P_2 \times C_4$ and $C_4 \times C_4$ are Q_2, Q_3 and Q_4, respectively.

5.3 Complete Multipartite Graphs

In this subsection, we determine the number of spanning trees of the complete multipartite graph

$$G = K(p_1, p_2, \ldots, p_m).$$

Here $m, p_1, p_2, \ldots, p_m$ are positive integers, and G is the graph on

$$n = p_1 + p_2 + \cdots + p_m$$

vertices, partitioned into m parts of sizes $p_1, p_2, \ldots, p_m$ respectively, and two vertices are adjacent if and only if they are in different parts. We claim the number of spanning

trees of G is

$$n^{m-2}\prod_{i=1}^{m}(n-p_i)^{p_i-1}.$$

We sketch the proof, which uses the degree matrix D defined in Section 3, but omit the ugly calculations. (This result was given in [6], without proof.)

Lemma : *Let R be a diagonal matrix with entries $d_1, d_2, \ldots$, and let J be the all 1's matrix of the same size. Then*

$$\det(R+J) = \prod_i d_i + \sum_i \prod_{j\neq i} d_j.$$

Proof : We use elementary row and column operations to reduce $R+J$ to an upper triangular matrix T.

First subtract the first row from every other row. Then, for each $i > 1$, add d_1/d_i times the ith column to the first column. The resulting matrix T is upper triangular, and $\det(R+J) = \det(T) =$ the product of the diagonal entries of T.

This proof works if no $d_i = 0$. Otherwise, we may assume $d_1 = 0$, and the proof above, with the column operations omitted, still works. □

Now we calculate $ST(G)$. We may assume $m > 1$. If each $p_i = 1$, then $n = m$, $G = K_n$, and the proposed formula agrees with Cayley's formula from Section 3. So we may assume that, say, $p_m > 1$. Form the degree matrix D, and delete the last row and column to give D'. Then $ST(G) = \det(D')$; we indicate how to calculate this.

First, factor out of each row of D' its diagonal entry. The result is a matrix of the form $M + I$, where M is a matrix of rank at most m. In particular,

$$M = AB$$

for some $(n-1)\times m$ matrix A, and some $m\times(n-1)$ matrix B. (The entries of A are mostly 0, with a few equal to $-d_i^{-1}$; the entries of B are mostly 1, with a few equal to 0.)

It is well known (and easy to prove) that AB and BA have the same spectrum, except for the multiplicity of 0 as an eigenvalue. Thus $AB + I$ and $BA + I$ have the same spectrum, except for the multiplicity of 1 as an eigenvalue. In particular, the product of the eigenvalues of $AB + I$ equals the product of the eigenvalues of $BA + I$, so $\det(AB + I) = \det(BA + I)$.

It remains to evaluate the determinant of the $m \times m$ matrix $BA + I$. To this end, factor the appropriate scalar out of each column of $BA + I$ to produce the matrix $R + J$, where R is the diagonal matrix with diagonal entries

$$-np_1^{-1}, -np_2^{-1}, \ldots, -np_{m-1}^{-1}, -(n-1)(p_m-1)^{-1}.$$

Now apply the Lemma above to complete the proof.

5.4 Complete Bipartite Graphs

The complete bipartite graph, with partite set sizes m and n, is denoted by $K_{m,n}$. We require $m, n \geq 2$, and let the root be any node in the part of size m. Note that

G	ST	DT	HP	HC	DF
$K_{2,2}$	4	2	2	2	2
$K_{2,3}$	12	3	0	0	6
$K_{2,4}$	32	4	0	0	24
$K_{2,5}$	80	5	0	0	120
$K_{3,2}$	12	4	4	0	4
$K_{3,3}$	81	12	12	12	12
$K_{3,4}$	432	24	0	0	48
$K_{3,5}$	2025	40	0	0	240
$K_{4,2}$	32	6	0	0	12
$K_{4,3}$	432	36	36	0	36
$K_{4,4}$	4096	144	144	144	144
$K_{4,5}$	32000	360	0	0	720
$K_{5,2}$	80	8	0	0	48
$K_{5,3}$	2025	72	0	0	144
$K_{5,4}$	32000	576	576	0	576
$K_{5,5}$	390625	2880	2880	2880	2880

Table 5: Results for complete bipartite graphs.

$ST(K_{m,n}) = ST(K_{n,m})$, and our computed results agree with the theoretical results of the previous subsection. It is easy to see that $DT(K_{2,n}) = n$ and $DF(K_{2,n}) = n!$ A moments thought shows that

$$DT(K_{m,m-1}) = HP = DF = ((m-1)!)^2,$$

with $HC = 0$; and that

$$DT(K_{m,m}) = HP = HC = DF = m((m-1)!)^2.$$

In all other cases, $HP = HC = 0$. The results for $m, n \leq 5$ are given in Table 5.

6 Conclusions

We raise the apparently difficult problem of enumerating particular kinds of spanning trees of a graph. We present an algorithm to list depth-first trees for moderate sized graphs. We use it to count the number of depth-first spanning trees in a variety of graphs, and we give results for some graphs in common classes. We give a theoretical answer to one specific question for an infinite class of graphs which is consistent with our practical results. Similar questions are of interest for breadth-first search spanning trees, with applications to the analysis of distributed breadth-first search algorithms [7].

References

[1] Shimon Even, Graph algorithms, Computer Science Press, 1979.

[2] A. Cayley, *On the theory of analytical forms called trees*, Phil. Mag., 13:172–176, 1857.

[3] A. Cayley, *A theorem on trees*, Quart. J. of Math., 23:376–387, 1889.

[4] A. Gibbons, Algorithmic graph theory, Cambridge University Press, 1985.

[5] G. Kirchoff, *On the solution of the equations obtained from the investigation of the linear distribution of galvanic currents*, 1847. English translation by J.B. O'Toole, IRE Trans. on Circuit Theory, CT-5(1):4–7, March 1958.

[6] C.J. Liu, *Enumeration of Hamiltonian cycles and paths in a graph*, Proc. of the AMS, 111(1):289–296, 1991.

[7] S.A.M. Makki, *Efficient distributed breadth-first search algorithm*, Computer Communications, 19:628-636, 1996.

[8] S.A.M. Makki and George Havas, *Empirical analysis of distributed depth-first search algorithms*, Proc. Eighth IASTED Internat. Conf. Parallel and Distributed Computing and Systems, Acta Press (1996) 292-296.

[9] S.A.M. Makki and George Havas, *Distributed algorithms for depth-first search*, Information Processing Letters, 60:7-12, 1996.

[10] S.A.M. Makki and George Havas, *An efficient method for constructing a distributed depth-first search tree*, Proc. Internat. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'97), CSREA Press (1997) 660–666.

[11] R.E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1(2):146–160, 1972.

[12] R.E. Tarjan, *Bounds on backtrack algorithms for listing cycles, paths and spanning trees*, Networks, 5:237–252, 1975.