The latter representation of $S(D)$ is shown in Fig. 1 with its digits placed over the transition paths of the trellis.

Now proceed briefly through the trellis. The syndrome is set to 0 prior to stage 0 so that at stage 0, $S^{(1)}(D) = 0$ and $S(D) = 1$. Likewise it is assumed initially that the shift register is cleared so that $t^{(1)}(D) = t^{(2)}(D) = 0$; this puts the algorithm at stage 0 at state $a = 0\ 0$. The label on the branch for $t(D) = 0$ is found by substituting these values in eqn. 7, i.e. $E_1(D) = 0$ and $E_2(D) = 1$. Hence the label on the branch for $t(D) = 0$ at stage zero is $[E_1(D), E_2(D)] = [0\ 1]$. By the same substitution, but with $t(D) = 1$, the label on the alternate branch is $[E_1(D), E_2(D)] = [1\ 0]$, the componentwise complement of the previous branch. The Hamming weight of $[E_1(D), E_2(D)]$ for both these branches is 1. Note that this weight of 1 is placed immediately above states a and c at stage 1.

To illustrate the Viterbi or dynamic-programming technique for computing survivors in the trellis suppose that the algorithm is either at state a or state b at stage 2. Note that there are only two ways on to state a, by a transition from b to a or a transition from a to a. At state a or b at stage 2, $S(D) = 0$ and $S^{(1)}(D) = 1$. At stage 2 at state a, $t^{(1)}(D) = 0$ and $t^{(2)}(D) = 0$, so that at the branch for $t(D) = 0$, by eqn. 7 $E_1(D) = 1$ and $E_2(D) = 1$. The total weight of $[E_1(D), E_2(D)]$ for the minimum weight path, going through state a at stage 2 is thus $2 + 2 = 4$. However, at stage 2 at state b, $t^{(1)}(D) = 0$ and $t^{(2)}(D) = 1$, so that at the branch for $t(D) = 0$, by eqn. 7 $E_1(D) = 0$ and $E_2(D) = 0$. Thus the total weight of $[E_1(D), E_2(D)]$ for the minimum weight path, going through state b at stage 2, is thus $3 + 0 = 3$. Since this weight is smaller than the previous weight only the path going through state b to a at stage 2 survives. The segment of path from state a to a is deleted as shown in Fig. 1. Similarly in Fig. 1 some paths lead to equal weights or a 'tie'. In such a case either segment can be chosen as part of the survivor path.

The entire trellis diagram shown in Fig. 1 is completed by the rules illustrated above. At stage 9 the minimum weight path in the trellis diagram of Fig. 1 is $a\ c\ d\ b\ c\ b\ c\ d\ b\ a$. The branches of this path yield $\hat{E}(D) = [1\ 0\ 0\ 0\ 0\ 1\ 0\ 0, 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0] = [1 + D^5, D^2] = [\hat{E}_1(D), \hat{E}_2(D)]$ as the estimate of the error vector. Subtracting these estimates of the error from $Z(D)$ produces $\hat{Y}_1(D) = [0\ 1\ 0\ 1\ 1\ 0\ 1\ 0]$ and $\hat{Y}_2(D) = [0\ 1\ 1\ 1\ 1\ 1\ 1\ 0]$ as estimates of the transmitted coded message. Finally the inverse linear sequential circuit of Massey and Sain,[4] with equation $\hat{X}(D) = (1 + D)\hat{Y}_1(D) + D\hat{Y}_2(D)$ is used to find $\hat{X}(D) = [0\ 1\ 0\ 0\ 1\ 0]$ as an estimate of the original message.

Acknowledgment: The authors wish to thank Charles Wang of JPL for his helpful suggestions made during the preparation of this letter. This work was supported in part by the National Aeronautics & Space Administration under contract NAS 7-100, and also in part by the US Air Force Office of Scientific Research under grant AFOSR-80-0151, in part by the US Army Research Office under grant DAAG 29-82-K-0142 and also in part by the Naval Air Systems Command under contract N00019-81-C-0541.

I. S. REED

Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-0272, USA

T. K. TRUONG

Communication System Research
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109, USA

24th February 1983

References

1 SCHALKWIJK, J. P. M., and VINCK, A. J.: 'Syndrome decoding of binary rate—½ convolutional codes', *IEEE Trans.*, 1976, **COM-24**, pp. 977–985
2 MCELIECE, R. J.: 'The theory of information and coding' (Addison-Wesley Publishing Co., Reading, MA, USA, 1974)
3 FORNEY, G. D.: 'Convolutional codes I: Algebraic structure', *IEEE Trans.*, 1970, **IT-16**, pp. 720–738
4 MASSEY, J. L., and SAIN, M. K.: 'Inverses of linear circuits', *ibid.*, 1968, **C-17**, pp. 330–337
5 NAGELL, T.: 'Introduction to number theory' (John Wiley & Sons, New York, 1951)

0013-5194/83/090344-03$1.50/0

EFFICIENT ALGORITHM FOR COMMON SPANNING TREE PROBLEM

Indexing terms: Circuit theory and design, Graphs and graph theory, Algorithms

Enumerating the spanning trees common to a pair of graphs is a problem which arises in symbolic circuit analysis. The algorithm presented here is between 350% and 26 000% faster than the best algorithm previously published, and hence the computer analysis of circuits of a reasonable size is now feasible.

Introduction: The tree enumeration method of symbolic circuit analysis is applicable not only to passive circuits[1] and active circuits with active elements in the form of four-terminal current sources (VCCS),[2] but has recently been extended[3] to active circuits containing four-terminal infinite gain amplifiers (FTOA). Since any active device can be modelled by a combination of FTOAs and passive components, any active circuit can be analysed in this way. The method consists of constructing voltage and current graphs as described in Reference 3; thereafter the problem reduces to finding the common spanning trees of the two graphs.

Grimbleby[4] has published an algorithm for enumerating the common spanning trees of two graphs G1 and G2, each with N edges and M vertices. It has recently been pointed out[5] that the CPU time for Grimbleby's algorithm can be reduced by up to 50% simply by using a doubly linked data structure for storing the trees. Further meaningful reductions in running time seem unlikely, as Grimbleby's work is based on a relatively inefficient algorithm[6] for finding the spanning trees of a single graph. As a consequence, in the worst case his algorithm generates all $\binom{N}{M-1}$ subsets of $M - 1$ edges of the graphs, and hence in the worst case the number of steps required is proportional to N^c, where c = minimum $(M - 1, N - M + 1)$.

In this letter a new algorithm for the common spanning tree problem is presented which is based on a relatively efficient algorithm for the single spanning tree problem, namely Minty's algorithm[7] as formalised by Read and Tarjan.[8] This new algorithm is between 350% and 26 000% faster than the modified form of Grimbleby's algorithm,[5] thus allowing Grimbleby's method of symbolic circuit analysis to be applied to larger circuits than are currently feasible.

Algorithm MRT: Algorithm MRT is similar to Grimbleby's algorithm in that spanning trees are built up one edge at a time in the form of partial spanning forests. All possible combinations of $(M - 1)$ edges are potentially generated. However, whenever an edge j of G1 (or G2) is encountered which is such that adding edge j to the current partial spanning forest for G1 (or G2) would induce a loop (or cycle), then edge j is immediately rejected. This reduces the number of possibilities to be investigated, as no spanning trees containing edge j and the current spanning forest are considered. Edge j is termed 'cycle-inducing'.

The key differences between the two algorithms are: (*a*) that algorithm MRT makes use of bridges, and (*b*) the indices $\{j\}$ of the edges of the current partial forest are stored as a list L; there is no explicit tree- or forest-building (with the resulting overheads).

Before defining a bridge, the concept of connectedness must be considered. A graph G is connected if there is a path between every pair of vertices v and w in G. A bridge of a graph G is an edge b such that if b were removed from the graph then G would no longer be connected. If a bridge is removed, then G is split into connected components such that

Table 1 CPU TIMES FOR THREE TYPICAL GRAPHS

M	N	y_1	y_2	y_c	CPU time original Grimbleby	CPU time modified Grimbleby	CPU time MRT algorithm
					s	s	s
10	15	405	187	9	1·565	1·083	0·112
10	20	1632	11565	93	10·758	6·248	1·769
20	25	792	12	2	26·654	11·464	0·044

M denotes number of vertices and N of edges. The number of spanning trees in G1 and G2 are denoted by y_1 and y_2, respectively; y_c is the number of common spanning trees

there is still a path between every pair of vertices within a connected component (but not within the graph as a whole).

Algorithm MRT uses bridges in two ways. First, any bridge clearly must be included in all spanning trees containing the current partial spanning forest. Second, if the jth edge of G1 is a bridge of G1, bu the jth edge of G2 connects two vertices in the same tree of the partial spanning forest for G2 (and is therefore cycle-inducing), then backtracking must commence immediately, as it is obviously impossible to satisfy the requirements of both including and excluding the jth edge.

Algorithm MRT is now given in a Pascal-like notation (unfortunately neither Basic nor Fortran support the recursive calls of steps 5 and 11).

```
procedure MRT;
 procedure REC_MRT;
 begin (*REC_MRT*)
1: if L has (M − 1) edges then output list L
    else
     begin
2:    if an edge j can be found which is not cycle-inducing
      then
       begin
3:      add j to L;
4:      mark as deleted from G1 and G2 all cycle-inducing
        edges;
5:      REC_MRT;
6:      replace all edges deleted at step 4 in G1 and G2;
7:      delete j from L; delete edge j from G1 and from G2;
8:      mark all edges j which are bridges of G1 or of G2;
9:      if no edge marked at step 8 as a bridge with respect
        to one graph is cycle-inducing with respect to the
        other graph then
         begin
10:       add the indices of all edges marked at step 8 to L;
11:       REC_MRT;
12:       remove from L the indices of all edges marked at
          step 8
         end;
13:     replace edge j in G1 and in G2
 end end end; (*REC_MRT*)
begin (*MRT*)
14: initialise L to contain the indices of all edges j which are
    bridges of G1 or of G2;
15: if both partial spanning forests are cycle-free then
     begin
16:   delete all cycle-inducing edges from G1 and G2;
17:   if G1 and G2 are both connected then
18:    REC_MRT
end end; (*MRT*)
```

The easiest way of implementing procedures MRT and REC_MRT is to make use of the standard algorithms of graph theory. The standard procedure for finding the bridges of a graph[9] is used at steps 8 and 14. Determining whether a graph (and hence a partial spanning forest) is cycle-free (step 15) can be done by looking for back edges in a depth-first search.[9] In order to determine which edges are cycle-inducing we need to know the connected components of the partial spanning forests. Two arrays of length M are maintained. The first contains the number of the connected component into which each vertex of G1 falls with respect to the partial spanning forest of G1, and similarly for G2 with respect to its partial spanning tree. (The numbers of the connected components correspond to the root numbers of the trees of Grimbleby's partial spanning forest[4]). An edge is cycle-inducing if the two vertices joined by that edge are in the same connected component; this is used to achieve steps 2, 4, 9 and 16. Step 17 uses the standard algorithm to test for connectedness.[9] All other steps either delete or add edges, or print out a spanning tree (step 1) and are thus easy to implement. It is possible to optimise procedure REC_MRT in a number of places, including computing the connectivity at steps 4 and 8 in one operation. The reader who is interested in the technical details should consult Reference 10.

The implementation requires two vectors of length N for storing the edges of G1 and G2, respectively, and the two vectors of length M mentioned above for storing the connected component number of each vertex. In addition, three further vectors of length N are required, namely IN_L, DELETED and BRIDGE. Vector IN_L is used to store the elements of L, DELETED is used to mark the edges which are to be deleted at step 4 and then replaced at step 6, while BRIDGE is used at steps 8 to 12 to indicate which edges are bridges of G1 or of G2.

Results: Algorithm MRT was coded in Pascal, as was Grimbleby's algorithm in both its original[4] and modified[5] form, and run on the University of Cape Town UNIVAC 1100/80 computer. The observed CPU times (in seconds) for three typical graphs are shown in Table 1.

In all three cases in Table 1 the relations $y_c \ll y_1$ and $y_c \ll y_2$ hold. They hold not only for randomly generated graphs, but also for graphs to which the tree enumeration method of symbolic analysis is applied in practice; Grimbleby[4] cites figures of $y_1 = 5000$ and $y_c = 2$ for a typical active-filter element.

From Table 1 it can be seen that algorithm MRT is between 600% and 60 000% faster than Grimbleby's original algorithm, and 350% to 26 000% faster than the modified algorithm. This new algorithm can therefore be used for the symbolic analysis of reasonably larger circuits than are feasible at present.

S. R. SCHACH

Computer Science Department
University of Cape Town
Private Bag, Rondebosch 7700, South Africa

21st February 1983

References

1 PERCIVAL, W. S.: 'The solution of passive electrical networks by means of mathematical trees', *Proc. IEE*, 1953, **100**, Pt. III, p. 143
2 PERCIVAL, W. S.: 'The graphs of active networks', *ibid.*, 1955, **102C**, pp. 270–278
3 GRIMBLEBY, J. B.: 'Symbolic analysis of circuits containing active elements', *Electron. Lett.*, 1981, **17**, pp. 754–756
4 GRIMBLEBY, J. B.: 'Algorithm for finding the common spanning trees of two graphs', *Electron. Lett.*, 1981, **17**, pp. 470–471
5 SCHACH, S. R.: 'Comment on "Algorithm for finding the common spanning tree of two graphs"', *ibid.*, 1982, **18**, pp. 988–989
6 CHRISTOFIDES, N.: 'Graph theory: an algorithmic approach' (Academic Press, London, 1975)
7 MINTY, G. J.: 'A simple algorithm for listing all the trees of a graph', *IEEE Trans.*, 1965, **CT-12**, p. 120
8 READ, R. C., and TARJAN, R. E.: 'Bounds on backtrack algorithms for listing cycles, paths and spanning trees', *Networks*, 1975, **5**, pp. 237–252
9 REINGOLD, E. M., NIEVERGELT, J., and DEO, N.: 'Combinational algorithms: theory and practice' (Prentice-Hall, New Jersey, 1977)
10 SCHACH, S. R.: 'On the common spanning tree problem'. Technical Report CS82-2, Computer Science Department, University of Cape Town, 1982

0013-5194/83/090346-02$1.50/0