last term of eqn. 2 is negligible. Therefore the series resistance is the slope of the I/V curve at the open-circuit point. This method gives a value of the series resistance of the flash lamp I/V curves of Fig. 3 equal to 7·3 $m\Omega cm^2$ that is in good agreement with the 7·8 $m\Omega cm^2$ value derived from the outdoor curve by Handy's method.

Finally, if the short circuit of the cell at a standard illumination (for example AM1 solar power) is known, the efficiency of the cell and the fill factor can be readily calculated for each concentration ratio.

In conclusion, the flash lamp testing technique reported herein is a simple, reliable and quick method for the assessment of the performance of concentrator GaAs solar cells, series resistance and efficiency, at concentration over 1000 suns in particular.

Acknowledgment: The author wishes to thank G. P. Donzelli and F. Faletta for valuable discussions and A. Tosoni for the technical support. This work was supported by CISE-ENEL/DSR contract 1980.

E. FANETTI *15th May 1981*

CISE SpA, PO Box 12081, 20100 Milan, Italy

References

1 CHAFFIN, R. J., and OSBOURN, G. C.: 'Measurement of concentrator solar cell series resistance by flash testing', *Appl. Phys. Lett.*, 1980, **37**, pp. 637–639

2 LARUE, R. A., BORDEN, P. G., and GREGORY, P. E.: 'A distributed resistance model of an AlGaAs/GaAs concentrator solar cell illuminated with a curved groove Fresnel lens', *IEEE Electron. Device Lett.*, 1981, **2**, pp. 41–43

3 FANETTI, E., FLORES, C., GUARINI, G., and PALETTA, F.: 'High concentration P.V. 100 Wp module making use of spectrum splitting and Si-GaAlAs coupled cells'. 1981 EC solar energy programme, project C, Coordination Meeting of Contractors, Sorrento, Italy

4 HANDY, R. J.: 'Theoretical analysis of the series resistance of a solar cell', Solid-State Electron., 1967, **10**, pp. 765–775

5 BARNES, P. A., and PAOLI, T. L.: 'Derivative measurements of the current-voltage characteristics of double-heterostructure injection lasers', *IEEE J. Quantum Electron.*, 1976, **10**, pp. 633–639

0013-5194/81/130469-02$1.50/0

ALGORITHM FOR FINDING THE COMMON SPANNING TREES OF TWO GRAPHS

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

A new algorithm is described which finds the common spanning trees of two graphs efficiently and without duplication. The algorithm can be incorporated in a symbolic circuit analysis program where its speed and storage requirements are such that circuits of considerable complexity can be analysed on small computers.

Introduction: The two-graph tree enumeration approach to the symbolic analysis of active circuits, which was originally formulated by Percival,[1] involves generation of the following:

{set of spanning trees of Gv} ∩ {set of spanning trees of Gi}

{set of (i, r)2-trees of Gv} ∩ {set of (j, r)2-trees of Gi}

where Gv and Gi are the voltage and current graphs derived from the original network, and i, j, r are nodes of the graphs. Since the (x, y)2-trees of a graph are simply the spanning trees of a new graph derived by coalescing nodes x and y, the symbolic circuit analysis problem reduces to that of finding the common spanning trees of two graphs.

One approach to this problem is to find the sets of spanning trees of the two graphs, and then to generate the intersection set. Unfortunately, however, this requires an indeterminate amount of workspace and is impracticable on most small computers for other than simple networks.

An alternative approach is to generate the spanning trees of one graph and to test each of these to determine whether it is also a spanning tree of the other graph. While this procedure is satisfactory for passive networks (because the graphs are similar and a large proportion of the trees of one graph are also trees of the other graph) it becomes inefficient for networks containing many active elements. In one typical active-filter circuit investigated there are 5000 spanning trees of one graph, only 2 of which are common.

This algorithm is based on a method proposed for finding the spanning trees of a single graph,[2] but which is probably less efficient for this purpose than alternative methods, Reference 3, for example. The two graphs for which the common spanning trees are required have N nodes and M branches, and will be termed G1 and G2. Two additional graphs, H1 and H2, are used by the algorithm. H1 and H2 have the same number of nodes as G1 and G2. Also required for the operation of the algorithm is a stack S of capacity $(N - 1)$ and a branch pointer K. A description of the algorithm follows below.

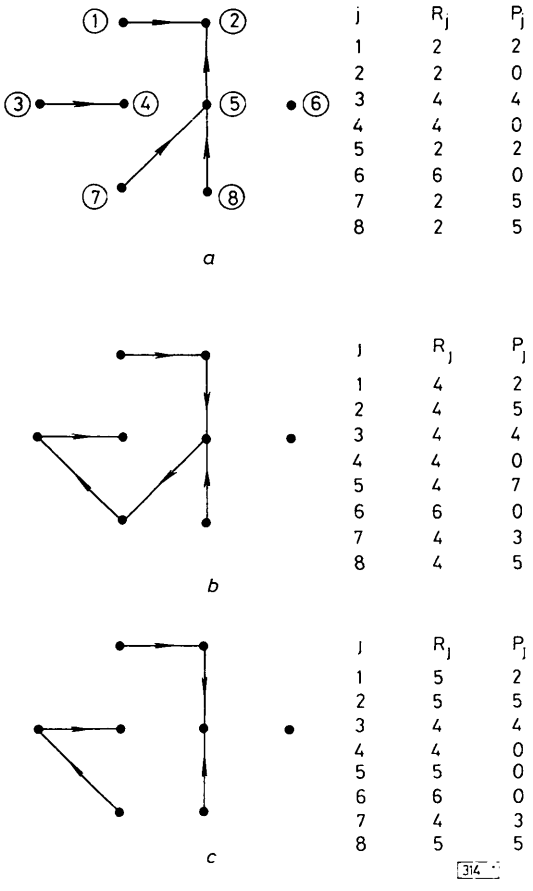
1 Initialise: H1 and H2 initially contain no branches. They are each considered to be a forest of N trees (for the purpose of this algorithm an unconnected node is taken to be a tree). The stack S is empty and the brach pointer $K = 0$.

2 Test for a common spanning tree: If there are $(N - 1)$ branches in the stack then a common spanning tree has been found; print the stack and go to 7.

3 Select a new branch: Increment K $(K = K + 1)$. If there are insufficient branches left to complete a spanning tree (i.e. if [number of branches on stack] + $M - K < N - 2$) then go to 6.

4 Does the branch K generate loops?: If branch K from G1 generates a loop when added to H1 then go to 3; if branch K from G2 generates a loop when added to H2 then go to 3.

5 Include branch K: Put branch K on top of stack S. Include branch K from G1 in H1 and branch K from G2 in H2. In both H1 and H2 this will merge two trees into a single tree. Go to 2.



j	R_j	P_j
1	2	2
2	2	0
3	4	4
4	4	0
5	2	2
6	6	0
7	2	5
8	2	5

j	R_j	P_j
1	4	2
2	4	5
3	4	4
4	4	0
5	4	7
6	6	0
7	4	3
8	4	5

j	R_j	P_j
1	5	2
2	5	5
3	4	4
4	4	0
5	5	0
6	6	0
7	4	3
8	5	5

Fig. 1 *Illustration of insertion and deletion of branches from a graph*

6 *Test for completion:* If the stack S is empty then exit from the algorithm: all common spanning trees have been found.

7 *Backtrack:* Remove branch from top of stack S. Let K = number of this branch. Remove branch K of G1 from H1 and branch K of G2 from H2. In both H1 and H2 this will split a tree into two distinct trees. Go to 3.

Graphs G1 and G2 can be stored in any convenient form, for example as two lists of M pairs of terminal nodes. H1 and H2, however, must be stored in a form that facilitates the following operations required by the algorithm:

(i) Test whether a particular branch, when added to H1 or H2, generates a loop. This is equivalent to testing whether the terminal nodes of the branch are in the same tree.
(ii) Include a branch connecting two trees, thus merging the trees into a single tree.
(iii) Remove a branch, thus splitting a tree into two distinct trees.

For each graph a list is maintained in which each node j ($j = 1$ to N) is assigned a pair of node pointers (R_j, P_j). One node in each tree of the graph is arbitrarily chosen as the root node; R_j is the root node of the tree to which node j belongs. P_j is the predecessor node of j and points to a node of the tree connected to j by a branch of the tree. Root nodes are an exception and have no predecessor. The list of N pairs of root and predecessor nodes completely defines a graph; however, a graph does not have a unique representation because the root nodes may be chosen arbitrarily. Fig. 1*a* shows a graph together with a possible representation. (The arrows on the branches point from a node to its predecessor and do not indicate that the branches are directed.)

Initially H1 and H2 contain no branches and each node is considered to be a tree. In this situation each node is its own root and has no predecessor, i.e. $R_j = j$, $P_j = 0$ for $j = 1$ to N.

A simple test can be applied to determine whether the addition of a branch (i, j) to the graph generates a loop. If the root nodes R_i and R_j of the branch terminal nodes are the same, then i and j belong to the same tree and the branch generates a loop. For example, in Fig. 1*a* nodes 1 and 7 have the same root and a branch (1, 7) would generate a loop.

To connect two trees by a branch (i, j) to form a single tree the following modifications to the Rs and Ps are made:

(i) All nodes with root R_j have their root changed to R_i.
(ii) The predecessors on the path between j and R_j are reversed, thus making j the root of this tree.
(iii) P_j is changed to i.

In Fig. 1*b* is shown the result of adding a branch (3, 7) to the graph of Fig. 1*a*. A branch (i, j) is removed from a tree, thus splitting it into two distinct trees, by making the following modifications to the Rs and Ps:

(i) If $P_j \neq i$ then swop i and j. Now $P_j = i$.
(ii) Make $P_j = 0$ and $R_j = j$, thus setting up j as the root of the separated tree.
(iii) Considering each of the nodes l from 1 to N, except node j, if $R_l \neq j$ and $R_{P_l} = j$, then change R_l to j. Repeat this sweep through the nodes until no further root modifications are made.

Fig. 1*c* shows the result of applying this procedure to the removal of branch (5, 7) from the graph of Fig. 1*b*.

Performance of algorithm: The common spanning tree algorithm can be programmed in about 85 basic statements and requires the storage of a total of $(10 + 5N + 4M)$ variables (integers). This includes storage of the original graphs and the stack. A symbolic analysis program incorporating the new algorithm was used to analyse an active-filter circuit of 20 nodes containing 24 passive components and 9 ideal operational amplifiers. On a small microprocessor-based computer the time taken to analyse this circuit was 96 min (interpreted) and 140 s (compiled).

J. B. GRIMBLEBY
Department of Engineering, University of Reading
Whiteknights, Reading, RG6 2AY, England

12th May 1981

References

1 PERCIVAL, W. S.: 'The graphs of active networks', *Proc. IEE*, 1955, **102**, Monograph 129R, pp. 270–278
2 CHRISTOFIDES, N.: 'Graph theory: an algorithmic approach' (Academic Press, 1975), pp. 125–132
3 CHAN, S. G., and CHANG, W. T.: 'A modification of Minty's method of tree listing'. Conference record of second Asilomar conference on circuits and systems, IEEE, 1968, pp. 608–611

0013-5194/81/130470-02$1.50/0

GaAs READ-TYPE IMPATT DIODE FOR 130 GHz CW OPERATION

Indexing terms: Semiconductor devices and materials, Impatts, Gallium arsenide

A Schottky-barrier GaAs read-type impatt diode has been fabricated from the vapour-phase epitaxial material. CW oscillation at 130 GHz with an output of 5 mW and 0·5% efficiency has been achieved.

Introduction: Below 40 GHz, GaAs read-type impatt diodes have offered both high power and efficiency. However, it has been speculated that, at the higher frequencies, the performance of GaAs devices is limited by the long intrinsic response time.[1] This effect might explain why GaAs impatt mode devices have had little experimental success in achieving oscillation above 50 GHz. The highest frequency of operation has been 96 GHz with about 1 mW of peak output power.[2] Recently, Elta[3] reported a GaAs mittatt source with 3 mW of output power at 150 GHz. This letter presents the results of a GaAs read-type impatt diode fabricated by use of the vapour-phase epitaxy (VPE) technique, from which an output of 5 mW at 130 GHz was achieved.

Device fabrication and packaging: The GaAs material used for device fabrication was grown in a Ga-$AsCl_3$-H_2 VPE reactor. The sulphur derived from H_2S was used as the n-type dopant. The substrate material was a silicon-doped GaAs wafer with a doping concentration of 2×10^{18} cm^{-3}. Prior to epitaxial growth, an *in situ* etch was used to remove a few micrometres of material from the top surface of the substrate wafer to ensure a clean surface for the subsequent growth cycles.

All epitaxial layers were grown *in situ*. A 2 to 3 μm-thick n^+ buffer layer with a doping concentration of 3×10^{17} cm^{-3} was grown first. Then a second layer of n-type material with a doping concentration of $1{\cdot}6 \times 10^{16}$ cm^{-3} was grown, followed by another n^+ layer of $2{\cdot}3 \times 10^{17}$ cm^{-3}. Fig. 1 shows the measured doping concentration against depth for this wafer. No attempt has been made to optimise the drift region thickness.

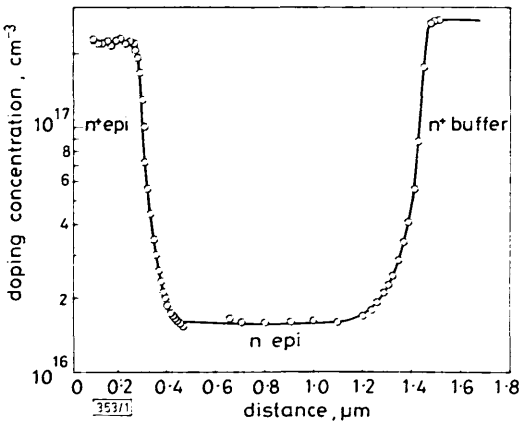


Fig. 1 *Measured doping profile*