Quaestiones Informaticae

Vol. 2 No. 3 September, 1983

Quaestiones Informaticae

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

An Improved Implementation of Grimbleby's Algorithm

Stephen R. Schach
Computer Science Department, University of Cape Town

Abstract
Details are given of an implementation of Grimbleby's algorithm for the common spanning tree problem with running times up to 50% less than for the original implementation. An explanation is given as to why implementations with even lower running times are unlikely.

1. Introduction

The common spanning tree problem may be formulated as follows:
Given a set V of M vertices, and given two sets E_1, E_2 each of N (undirected) edges such that there exists a bijection $f : E_1 \rightarrow E_2$, find all spanning trees T_1 of $G_1 = (V, E_1)$ such that $T_2 = f(T_1)$ is a spanning tree of $G_2 = (V, E_2)$. (If T_1 denotes the tree $\{e_1, e_2, \ldots, e_{M-1} | e_i \text{ in } E_1\}$, then $f(T_1)$ denotes the graph $\{f(e_1), f(e_2), \ldots, f(e_{M-1})\}$).

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 FTOA and passive components, any active circuit can be analysed in this way. The method consists of constructing voltage and current graphs as described in [3]; thereafter the problem reduces to finding the common spanning trees of the two graphs.

Grimbleby [4] has published an algorithm for enumerating these common spanning trees. Running times for his algorithm are non-trivial; for an implementation in Basic running on a microcomputer he cites execution times of 104 seconds (compiled) and 96 minutes (interpreted) for an active filter circuit of 20 nodes containing 24 passive components and 9 ideal operational amplifiers.

It has recently been pointed out [5] that the CPU time for Grimbleby's algorithm can be reduced by between a third and a half simply by using a doubly linked data structure for storing the trees. This improvement certainly constitutes a meaningful saving in computer time.

In this paper we account for the observed improvement by means of a detailed implementation as well as a complexity analysis. Finally, we explain why we do not consider it likely that implementations with significantly lower running times can be found.

In what follows, the term "original" will refer to Grimbleby's implementation, [4] while "modified" will be used to mean the implementation described in this paper.

2. Grimbleby's Algorithm

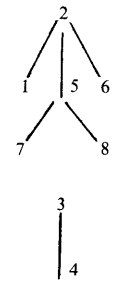
Grimbleby's original algorithm appears in reference [4]. In this paper we use the word "edge" in place of "branch", and "vertex" in place of "node", in order to conform with generally accepted graph theoretical nomenclature; otherwise, Grimbleby's notation is followed.

An inefficiency in the original implementation arises from Step 7 (Backtrack), which has complexity $O(N^2)$. The storage of the edges of the forests as an array of pairs of vertex pointers ($root_j$, $father_j$) requires a sweep through the edges to be repeated "until no further root modification is made" [4]. Consider now the case where the tree is a chain (N, N-1, . . ., 1), and edge (N, N-1) is to be removed. N-2 sweeps are required, and on each sweep O(N) operations are performed. Thus Step 7 has worst case complexity $O(N^2)$. In fact, the average complexity is also $O(N^2)$; this can be shown by a method similar to the complexity analysis of bubblesort [6].

The overall complexity is then at least $O(N^3)$, because a complete spanning tree has N-1 edges each of which has to be unstacked at Step 7 during the running of the algorithm.

3. The Modified Implementation

Instead of storing each edge as a pair ($root_j$, $father_j$) as in the original implementation, we now represent each edge by a quadruple of vertex pointers ($root_j$, $father_j$, son_j, $brother_j$). $Root_j$ is the root of the tree to which vertex j currently belongs. $Father_j$ is the next vertex on the path from vertex j to $root_j$. At any stage during the execution of the algorithm vertex j may have more than one son; son_j is defined to be the leftmost one. Finally, if vertex k = $father_j$ has more than one son vertex, then $brother_j$ denotes the son immediately to the right of vertex j. These definitions are illustrated in Figure 1.



j	$root_j$	$father_j$	son_j	$brother_j$
1	2	2	0	5
2	2	0	1	0
3	3	0	4	0
4	3	3	0	0
5	2	2	7	6
6	2	2	0	0
7	2	5	0	8
8	2	5	0	0

FIGURE 1. Example of nomenclature.

Consider vertex 5. We see that $root_5 = 2$, $father_5 = 2$, $son_5 = 7$, $brother_5 = 6$.

The three basic operations performed on a forest are 'addson' which makes vertex A a son of vertex B, 'removeson' which removes a vertex from its tree thus yielding two trees, and 'changerootpointer' which sets the root pointer of a vertex F and all its descendents in its tree to F. These procedures are implemented as follows:

```
PROCEDURE addson (A, B);
(* this procedure makes vertex A a son of vertex B *)
BEGIN addson
  IF B has no son
    THEN make B the son of A
    ELSE make B the brother of the rightmost son of B;
  father_A := B
END addson;

PROCEDURE removeson (D);
(* vertex D is a son of vertex C; this procedure inlinks D *)
BEGIN removeson
  C := father_D;
  IF D is the only son of C
    THEN set son_C := 0
    ELSE IF D is leftmost son
        THEN make brother_D the son of C
        ELSE scan the sons of C from left to right until a
             node E is found whose brother is D, and set
             brother_E := brother_D;
  father_D := 0
END removeson;

PROCEDURE changerootpointer (F);
(* this procedure sets root pointer of F and all its
   descendents in its tree to F *)
```

```
PROCEDURE recursivechange (G, rootnumber);
(* this procedure sets root pointer of vertex G, its
   son and its brother to rootnumber *)
BEGIN recursivechange
  IF G is non-null
    THEN BEGIN
         root_G := rootnumber;
         recursivechange (son_G, rootnumber);
         recursivechange (brother_G, rootnumber)
           END
END recursivehange;

BEGIN changerootpointer
  root_F := rootnumber;
  recursivechange (son_F, F)
END changerootpointer;
```

Procedures 'addson', 'removeson' and 'changerootpointer' have complexity O(≠S(A)), O(≠S(C)) and O(≠S(F)) respectively, where ≠S(v) denotes the number of sons of vertex v. Clearly 'addson (A,B)' cannot be performed unless A is the root of its tree; in this case procedure 'makeroot' must first be called.

```
  PROCEDURE makeroot (H);
  (* this procedure makes H the root of its tree *)
  PROCEDURE reverselink (J, K);
  (* this procedure makes J a son of K; before the call K is
     a son of J *)
  BEGIN reverselink
    IF J is non-null
    THEN BEGIN
           removeson (K); addson (J, K);
           reverselink (father_J, J)
           END
  END reverselink;
BEGIN makeroot
  reverselink (father_H, H);
  changerootpointer (H)
END makeroot;
```

The complexity of 'makeroot' is $\text{sum}_i|O(\neq S(i))| + O(N)$ where the sum is taken over all nodes i on the path from father_H to root_H in the original tree. The sum is clearly bounded by O(N) since the total number of sons of vertices along any path in a tree cannot exceed the number of vertices in the tree itself. Thus the complexity of 'makeroot' is also O(N).

Procedures 'linkedge' (Step 5 of the algorithm) and 'unlinkedge' (Step 7) can now be given.

```
PROCEDURE linkedge (e);
(* this procedure adds edge e = (P, Q) to the forest *)
BEGIN linkedge
  makeroot (P);
  addson (P, Q);
  changerootpointer (Q)
END linkedge;
```

```
PROCEDURE unlinkedge (e);
(* this procedure removes edge e = (P, Q) from the forest *)
BEGIN unlinkedge
  IF P is the son of Q
    THEN interchange P and Q;
  removeson (Q);
  root_Q := Q;
  changerootpointer (Q)
END unlinkedge;
```

From the complexity analysis of the routines called by 'linkedge' and 'unlinkedge' it is clear that both these procedures have complexity O(N); in the original implementation the corresponding procedures had complexity O(N) and $O(N^2)$ respectively.

The original and modified algorithms have been implemented in Pascal on a Sharp MZ-80B microcomputer. The modified algorithm is some 25% longer, and requires 2M more storage locations (for storing the edges). However, as predicted by the above analysis, the modified algorithm is more efficient. Comparative CPU times are given in Figure 2; savings from 33% to over 50% have been obtained.

Further Improvements

The question might well be asked: are further improvements in the running time of Grimbleby's algorithm possible? The answer is probably not, because in the worst case the main loop of the algorithm considers all possible combinations of M-1 edges from N edges. The algorithm as a whole therefore has worst case complexity $O(N^b)$ where b = min (M-1, N-M+1), and it is therefore unlikely that running times for Grimbleby's algorithm can be further significantly reduced.

M number of nodes	N number of branches	number of trees	CPU time original algorithm	CPU time modified algorithm	percent CPU time decrease
10	14	9	27 s	18 s	33%
10	20	93	255 s	147 s	42%
20	25	2	539 s	234 s	56%

FIGURE 2 : Comparative CPU times.

References

[1.] PERCIVAL, W.S.: The solution of passive electrical networks by means of mathematical trees, Proc. IEE, Vol. 100, Part III, Paper 1492R, 1953, p. 143.

[2.] PERCIVAL, W.S.: The graphs of active networks, Proc. IEE, 102C, Monograph 129R, 1955, pp. 470-471.

[3.] GRIMBLEBY, J.B.: Symbolic analysis of circuits containing active elements, Electron. Lett., Vol. 17, 1981, pp. 754-756.

[4.] GRIMBLEBY, J.B.: Algorithm for finding the common spanning trees of two graphs, Electron. Lett., Vol. 17, 1981, pp. 470-471.

[5.] SCHACH, S.R.: Comment on 'Algorithm for finding the common spanning trees of two graphs', Electron. Lett., Vol. 18, 1982, pp. 988-989.

[6.] KNUTH, D.E.: The Art of Computer Programming. Volume 3: Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM*, **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages*, McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

*Presented at the second South African Computer Symposium held on 28th and 29th October, 1981.

MILITHO