An algorithm to generate all topological sorting arrangements

Y. L. Varol* and D. Rotem†

A new algorithm to generate all topological sorting arrangements is presented. It is compared with other such algorithms and is found to be more efficient.

(Received April 1978; revised April 1979)

Topological sorting is the problem of finding a permutation of N objects $\{a_i\}_{i=1}^{N}$ which conforms to a partial order defined by a binary relation R having the properties of assymetry and transitivity. We shall take a_iRa_j to imply that a_i must precede a_j in the sorted sequence to be derived. For finding one permutation consistent with the partial order, Knuth (1968, p. 258) has given an efficient version of an algorithm first published by Kahn (1962). More recently a new method based on constructing a ternary tree and then traversing it has been proposed (Szwarcfiter and Wilson, 1978). The number of topological sortings is a decreasing function of the number of relations in the partial order, and even for large N, it may be practical to generate all of them and search for a particular one. It would be grossly inefficient to generate and test each of the $N!$ permutations in order to obtain all topological sortings. One could use backtracking techniques in conjunction with Knuth's algorithm (Knuth and Szwarcfiter, 1974). Wells (1971, p. 140) has given another algorithm which dynamically constructs subsets of marks that can occupy the i^{th} position.

The procedure presented below is similar to an algorithm attributed to M. Hall Jr. if it were used to generate all $N!$ permutations (see Ord-Smith, 1970). It assumes that initially P contains a starting topological sorting arrangement of the N objects plus an end marker as its $(N + 1)$'st entry. The boolean matrix M, the incidence matrix of the renamed partial order where objects are referenced by their position in the starting permutation, is defined as

$$M(i,j) = \begin{cases} \text{true if } iRj \text{ belongs to the partial order or } j \text{ is the end marker} \\ \text{false otherwise} \end{cases}$$

It uses a vector LOC to keep track of the location of objects as they are transposed, and it calls on procedure PROCESS to process the generated arrangement **(Fig. 1)**.

In our procedure when a transposition cannot take place, i.e. $M[I, OBJ\_K1]$ is true, a cyclic right rotation of the objects $P[I], \ldots, P[\text{LOC}[I]]$ is performed. This primitive operation, which we shall refer to as rotate, can be performed very efficiently on some computers. This algorithm can also be derived by modifying the Johnson–Trotter algorithm (Sedgewick, 1977). Such a derivation would have been too lengthy and involved. However, it would have highlighted the fact that the algorithm places a mark in every possible location relative to the other marks, which is the basis of a proof that all topological sorting arrangements are generated.

Consider finding all the permutations of the set of marks $\{a, b, c, d, e\}$ subject to the constraints cRa, bRc and bRe, where R implies that the mark to its left must precede the one to its right. One such permutation is $bcade$. Using this and renaming we reformulate the problem as: Starting from 12345, generate all

```
PROCEDURE TOPSORT(VAR N:INTEGER;LOC:LIST1; P:LIST2; M:TABLE);
(* OBJECT IS A DATA TYPE DEFINED IN A CALLING PROGRAM, LIST1 IS A      *)
(* DATA TYPE OF THE FORM ARRAY [1 . . X] OF INTEGER, LIST2 IS A DATA   *)
(* TYPE OF THE FORM [1 . . X + 1] OF OBJECT, AND TABLE IS A DATA TYPE  *)
(* OF THE FORM ARRAY [1 . . X, OBJECT] OF BOOLEAN, WHERE X IS AN INTE- *)
(* GER ⩾ N. THE PROCEDURE WILL ACCESS ONLY THE FIRST N ELEMENTS        *)
(* OF LOC, THE FIRST N + 1 ELEMENTS OF P, AND THE FIRST N ROWS OF M.   *)
VAR I, K, K1, L:INTEGER;
    OBJ_K, OBJ_K1:OBJECT;
BEGIN
      PROCESS;
      I := 1;
      WHILE I < N DO
      BEGIN K := LOC[I]; K1 := K + 1; OBJ_K := P[K]; OBJ_K1 := P[K1];
            IF M[I, OBJ_K1] THEN BEGIN FOR L := K DOWNTO I + 1
                                       DO P[L] := P[L - 1]; P[I] := OBJ_K;
                                          LOC[I] := I; I := I + 1
                                       END
                            ELSE BEGIN P[K] := OBJ_K1;
                                       P[K1] := OBJ_K; LOC[I] := K1;
                                       I := 1; PROCESS
                                 END
      END
END
```

Fig. 1

*Computer Science Department, Southern Illinois University, Carbondale, Illinois 62901, USA.
†Computer Science Department, University of Waterloo, Waterloo, Ontario N2L 3E5, Canada.

CCC-0010-4620/81/0024-0083 $01.00

Table 1

12345	*bcade*	41253	*dbcea*	15234	*becad*
*12345		*14253		*15234	
*12345		14523	*bdeca*	*12534	
12435	*bcdae*	41523	*dbeca*	12543	*bceda*
*12435		*14523		*12543	
14235	*bdcae*	*12453		15243	*becda*
41235	*dbcae*	*12345		*15243	
*14235		12354	*bcaed*	15423	*bedca*
*12435		*12354		*15423	
12453	*bcdea*	*12354		*12543	
*12453		12534	*bcead*	*12354	
14253	*bdcea*	*12534		*12345	

Table 2

12345	*bcade*	12543	*bceda*	*14235	
12354	*bcaed*	15243	*becda*	41235	*dbcae*
12534	*bcead*	*12435		41253	*dbcea*
15234	*becad*	14235	*bdcae*	41523	*dbeca*
*12345		14253	*bdcea*	*41235	
12435	*bcdae*	14523	*bdeca*	*12345	
12453	*bcdea*	15423	*bedca*	*12345	
				*12345	

the permutations of five marks subject to the constraints 1R2, 2R3 and 1R5. The sequence of permutations generated by our algorithm is shown in **Table 1.** The star implies that the permutation is a result of rotate operation, and thus is a duplicate. To generate **Table 2**, the same algorithm and input data was used but the direction of motion for the marks was reversed and the main loop was started with $i = N$. The important thing to note here is that Table 2 has much fewer duplicates than in Table 1, i.e. fewer executions of the rotate operation. This is due to the fact that in the example considered the marks 4 and 5 are less constrained than the other marks.

Let us now derive an upper bound for the average number of times the rotate operation will be executed when there are t permutations that satisfy a given partial order. If there were no restrictions and all $N!$ permutations had to be generated, then $(N - i + 1)! - (N - i)! = (N - i)(N - i)!$ of them would be due to a transposition of mark i with its right hand neighbour. Assuming that the t topological sortings are uniformly distributed over transpositions of various marks, we can say that $\frac{t}{N!}(N - i)(N - i)!$ are due to a transposition of mark i. After each such transposition we may have to rotate all the marks less or equal to i. Thus, the total number of times the rotate operation will be executed is less than or equal to

$$\sum_{i=1}^{N-1} i \frac{t}{N!}(N - i)(N - i)! \leqslant t\left(1 + \frac{2}{N}\right).$$

```
K := 1;
WHILE K > 0
 BEGIN
  WHILE there exists mark i with no predecessors and has
  not been already considered with the current partial con-
  figuration
 BEGIN P[K] := I;
          erase all relations of the form iRj;
          K := K + 1
 END;
 IF K := N + 1 THEN PROCESS;
 K := K - 1; I := P[K];
 retrieve all relations of the form iRj
END
```

Fig. 2

The conceptual difference between our algorithm and the one given by Knuth and Szwarcfiter ($K - S$ algorithm) (1974), can be summarised as follows. We fix k marks in relative positions and then use adjacent transpositions to cover all possibilities of placing the $k + 1$ 'st mark relative to the initial k marks. The $K - S$ algorithm having fixed k marks in the first k absolute locations considers all possible marks that could be used to fill in the next location. For further comparison let us briefly outline the $K - S$ algorithmin **(Fig. 2)**.

We notice that this routine requires frequent erasure and retrieval of relations, which compared with rotate is considerably more costly. In our algorithm we can have at most $N - 1$ successive executions of rotate operations in between the generation of two topological sortings. In fact, this worst case happens only at the very end when all the required permutations have been generated. Thus, our algorithm takes at most $0(N)$ units of time per output. On the other hand, the $K - S$ algorithm may take $0(m + N)$ units of time to generate the next topological sorting, where m is the number of pairwise relations in the partial order. Finally, we remark that the program implementation of our algorithm is much simpler and straightforward. This is mainly due to the absence in our algorithm of dynamic changes in the data structure which are necessary in the $K - S$ algorithm.

Acknowledgement

The authors are indebted to the referee who helped to improve the presentation and eliminate certain errors.

References

KAHN, A. B. (1962). Topological Sorting of Large Networks, *CACM*, Vol. 5, pp. 558-562.

KNUTH, D. E. (1968). Fundamental Algorithms, in *The Art of Computer Programming* 1, Addison-Wesley, Reading, Mass.

KNUTH, D. E. and SZWARCFITER, J. L. (1974). A Structured Program to Generate all Topological Sorting Arrangements, *Inf. Proc. Lett.*, Vol. 2, pp. 153-157.

ORD-SMITH, R. J. (1970). Generation of Permutation Sequences: Part 1, *The Computer Journal*, Vol. 13 No. 2, pp. 152-155.

SEDGEWICK, R. (1977). Permutation Generation Methods, *Computing Surveys*, Vol. 9, pp. 137-164.

SZWARCFITER, J. L. and WILSON, L. B. (1978). Some Properties of Ternary Trees, *The Computer Journal*, Vol. 21 No. 1, pp. 66-72.

WELLS, M. B. (1971). *Elements of Combinatorial Computing*, Pergamon Press, Elmsford, N.Y.