Generation of permutation sequences: Part 1

R. J. Ord-Smith

Computing Laboratory, University of Bradford, Bradford 7

There has been considerable interest in the last ten years or so in methods of generating sequences of arrangements of n elements in such a way that each of the $n!$ arrangements is generated once, and only once, in the sequence. We call such sequences of arrangements *permutation sequences*. In part 1 we consider several kinds of permutation sequences and describe some of their properties. Part 2 is devoted to a detailed examination of the practically most efficient six published algorithms and a discussion of implementation difficulties and compiler overheads. An Appendix to Part 2 contains an extensive bibliography of related work.

(Received November 1969)

1. Introduction

Much use of permutation algorithms has been made in the last few years in studies in Combinatorial Mathematics. Many conjectures have been proved or have fallen by computer techniques involving systematic searches. Increased efficiency in such searches is due in part to improvements in the speed of permutation algorithms. Timing these algorithms on one computer shows speed increases of a hundred or two hundredfold between the earliest and the latest.

One of the interesting applications has been the search for orthogonal Latin Square pairs of order 10, which Euler had conjectured did not exist. Searches began in earnest in the mid-1950s with the construction of large fast computers, but it was not until 1959 that the existence of the first two such pairs was announced. By 1962 thousands of pairs had been discovered, and the speed of search had increased by a factor of 10^{12}. Some of these developments are nicely described by Gardner (1966). The author (Ord-Smith, 1965) has described an application of Block Design techniques to the problem of constructing redundant fault reducing circuits using majority votetakers. Block Design theory usually regards as isomorphic two designs whose incidence matrices have permuted rows and columns, since they constitute merely re-labelled varieties within re-labelled blocks. But by successively presenting fault carrying information to a fault reducing circuit based on such a design, a particular labelling of input and output wires can produce greater fault reducing efficiency, i.e. the elimination of faults in fewer passes. Thus, two isomorphic designs can differ in this respect. Systematic generation of incidence matrices by permutation has been used to find best fault reducing circuits.

2. Computer permutation sequence algorithms

Apart from a few algorithms describing some specialised purposes involving permutations (see, for example, Hill 1968), the general algorithms all provide a common facility, the systematic generation of $n!$ arrangements of n *marks*. It is usual to provide a procedure which, on successive calls, will carry out permutations on a set of marks so that, after $n!$ calls, each arrangement of the marks has been generated once and once only. In some, one can initialise the process by providing a boolean parameter set **true**. This will be returned **false** and, when subsequently given **false** in each call, will return **false** until, after $n!$ calls, it will be returned **true** again.

It is necessary, at each procedure call, to recall the point which has been reached in the sequence of arrangements. If the marks are distinct and numerical then the arrangement can itself be used to give this information. This technique has been described by Mok-Kong Shen (1962 and 1963) and featured in several of the algorithms to be described below. However, if there is to be no such restriction on the marks then the information has to be kept separately and, for this purpose, a *signature* is contained within the procedure.

3. The Tompkins algorithms

The first explicit description of computer algorithms for the generation of permutation sequences seems to have been given by Tompkins (1956). Incidentally, his paper also reviews some of the problems for which permutation algorithms are required.

The basis of the Tompkins algorithms is that the signatures are modified at each call by a process involving mixed radix arithmetic. The simplest version, attributed to M. Hall, Jnr., will show the mechanism.

Hall algorithm

A signature consists of n elements $t_1\ t_2 \ldots t_n$ constituting a mixed radix number in which the element t_k has radix k. t_1 is effectively a dummy element which remains zero.

Successive calls produce modulus arithmetic counting which takes place in the signature from the most significant radix end. For example, with $n = 5$ we would have signatures as shown in the first panel of **Table 1**.

Each signature defines an arrangement of a set of simple marks $1, 2, \ldots, n$ such that the value of t_k in the signature tells how many marks $<k$ are to the right of the mark k in the arrangement. Thus, signatures shown in panel 1 of Table 1 correspond to the arrangements of panel 2. Tompkins was already aware of advantages in generating a sequence of arrangements in 'a convenient order'. He describes a variation to this end attributed to Paige.

Table 1

Generation of Hall sequence

SIGNATURE t_1	t_2	t_3	t_4	t_5	ARRANGEMENT				
0	0	0	0	0	1	2	3	4	5
0	0	0	0	1	1	2	3	5	4
0	0	0	0	2	1	2	5	3	4
0	0	0	0	3	1	5	2	3	4
0	0	0	0	4	5	1	2	3	4
0	0	0	1	0	1	2	4	3	5
0	0	0	1	1	1	2	4	5	3
0	0	0	1	2	1	2	5	4	3
.	.	.	.	.	.	.	.	.	.

Tompkins–Paige algorithm

In this algorithm modification of the signature again involves modulus arithmetic counting, though this time from the least significant radix end (see **Table 2**). If t_k

Table 2

Generation of T–P sequence

SIGNATURE t_5	t_4	t_3	t_2	t_1		ARRANGEMENT				
0	0	0	0	0		1	2	3	4	5
0	0	0	1	0		1	2	3	5	4
0	0	0	(2)	0	*	1	2	3	4	5
0	0	1	0	0		1	2	5	3	4
0	0	1	1	0		1	2	5	4	3
0	0	1	(2)	0	*	1	2	5	3	4
0	0	2	0	0		1	2	4	5	3
0	0	2	1	0		1	2	4	3	5
0	0	2	(2)	0	*	1	2	4	5	3
0	0	(3)	0	0	*	1	2	3	4	5
0	1	0	0	0		1	5	2	3	4
0	1	0	1	0		1	5	2	4	3
.	.	.	.	.		.	.	.	.	.

is the most significant digit to be modified in a particular call, taking account of a possible carry, then the arrangement of marks suffers a cyclic permutation of the k right most marks. If the next signature obtained by a further step involves a carry, the corresponding arrangement will have occurred before. Tompkins calls these recurring arrangements 'useless starred permutations'. They have to be removed by continuing with the transmission of the carry through several digit positions, if necessary.

This algorithm, organised in reverse order, with counting down in the signature, was the second of the permutation algorithms published in the *Communications of the A.C.M.* as Algorithm 86 (Peck and Schrak, 1962). An important improvement in the rules for constructing the arrangement from the signature is that only the position of the marks is important and not their value. As already mentioned in Section 2, if one can rely on distinct numerical marks then no signature is needed at all. In fact, as we shall see in detail, the fastest algorithm of all, an improved version of Algorithm 28 (Phillips, 1967) exploits this most efficiently. Use of a signature should allow the greater generality of any marks. The early published ACM Algorithm 71 (Coveyou and Sullivan, 1961) suffers the same disadvantage as the Hall algorithm in that the marks are restricted in spite of the use of a signature.

4. Nested cycle methods

Inner–outer method

The Tompkins–Paige algorithm is effectively an example of a nested cycle method. The permutations carried out are all cyclic permutations, though the rules of construction are such as to minimise the length of the cycles and thus reduce the total number of transpositions made. There is, however, the cost of carrying and updating a signature and that of generating the unwanted starred permutations. This is sometimes called the 'inner–outer' nested cycle method in which least work is done in the innermost cycles.

Outer–inner method

Nested cycle methods have been revived more recently by Langdon (1967). He suggests the use of rotational registers to exploit cyclic generation. Langdon's is an 'outer–inner' nested cycle method in which the length of cycle is maximised whilst the generation of starred permutations is minimised. The simplest way in which to conceive the method is to regard it as identical to the Tompkins–Paige algorithm but using a Hall signature. However, by observing that there are always K successive cycles of order K it is possible to dispense with the use of a complicated signature. This is, therefore, a method requiring neither signature nor imposing restriction on the values of the marks. But the number of transpositions required is large and implementation in a high level language is very inefficient. The method is justified only if fast hardware rotation registers are available. See also references, Langdon (1968), Hill (1968), Ord-Smith (November 1967).

5. The Wells, Johnson and Trotter algorithms

The essential distinction in these and other efficient algorithms lies not in the construction of the signature, which remains virtually the same as that of Tompkins–Paige; it lies in the construction of the corresponding arrangement of marks. The inefficiency of the T–P algorithm depends not only in the production of the starred arrangements but in the use of cyclic permutations. Unless one can exploit these in a special way as Langdon suggests, they involve, in the organisation available to most computer programmers, the successive interchange of a number of pairs of marks. Wells (1961) has shown that each arrangement of a sequence can be generated from its predecessor by just a single such transposition. Although effectively of similar construction to the T–P signature, the signature of Wells uses an element t_k of radix k but with allowed values $1(1)k$ rather than $0(1)k - 1$. This facilitates the des-

cription of an arrangement as a function of the corresponding signature (see details of Section 6). Thus the complete generation of $n!$ arrangements involves the generation of just $n!$ transpositions. In the T–P algorithm the number of transpositions tends to $(e-1)n! = 1{\cdot}718n!$ transpositions as n increases. Johnson (1963) has shown further that a sequence of arrangements can be found for any n in which these transpositions are adjacent. There may possibly be certain combinatorial advantages in an adjacent transposition sequence but there is a severe penalty in the complication of the algorithm.

Boothroyd's *Computer Journal* Algorithm 29 is a direct implementation of Wells' method. In his Algorithm 30 he makes use of the fact that, for $n \geqslant 5$, there is a very simple pattern in the successive generation of 23 arrangements of the four least significant marks. This gives a very fast algorithm, (Boothroyd, 1967). Improved versions of these algorithms will be given and discussed in Part 2. The existence of both Wells and Johnson sequences shows that transposition sequences are not unique. For $n = 3$ there are six distinct transposition sequences. A sequence and its exact reverse are regarded as the same. ACM Algorithm 115 (Trotter, 1962) is an efficient transposition algorithm in which the administration of the signature vector is particularly elegant and which produces a bell-ringing sequence. This is included among the six algorithms of Part 2.

6. The lexicographic algorithms

The lexicographic sequence is perhaps the most natural. It is most readily imagined by regarding the marks as labelled A, B, C, . . . then the sequence of arrangements places these in dictionary order. For example, on three marks the sequence is ABC, ACB, BAC, BCA, CAB, CBA. It is at once clear that the sequence involves more than $n!$ transpositions. ACB → BAC requires two in this example.

A succession of ACM Algorithms 87, 106, 130, 202 gradually improved the efficiency by a speed factor of 60, though Algorithm 202 remained more than twice as slow as Algorithm 115. A speed comparison of these algorithms for one computer has been made by the author, Ord-Smith (July, 1967).

Rules for the generation of the lexicographic sequence have been given by Mok-Kong Shen (1962) and are included here with a practical improvement. In the case of a lexicographic sequence there is the further problem of determining the number of transpositions involved in generating the full sequences of $n!$ arrangements. A careful examination of the lexicographic sequence shows that generation can be described as a recursive application of a simple set of rules which can be obtained from a signature of precisely Wells kind. The rules for construction of a Wells sequence of signatures and the formation of corresponding arrangements into a lexicographic sequence read right to left are as follows:

Let t_i with $i = 2(1)n$ be a set of elements of a signature in the usual way. There is no need to use the 'dummy' element t_1 which would remain unity. The elements t_i are initially set 1. At any moment let k' be the smallest k for which $t_k \neq k$.

1. (i) If $t_2 = 1$ then $t_2 := 2$.
   (ii) The permutation P_1P_2 is performed on the arrangement of marks.
2. (i) If $t_2 = 2$ then determine k' by examining t_i with increasing i and all the while setting t_i unity for $i < k'$.
   (ii) The permutation $P_{k'}P_{t_{k'}}$ is performed on the arrangement of marks followed by a reversal of the $k' - 1$ least significant position marks.
   (iii) $t_{k'} := t_{k'} + 1$.
3. (i) Generation is completed when $t_n = n$.
   (ii) Reversal of all n marks restores the identity.

Rules 1 are included within 2 but an increased efficiency is gained in a computer program by dealing with the case explicitly. Generation of the first few arrangements in the sequence with $n = 5$ will make the process clear and this is shown in **Table 3**.

Table 3

Generation of lexicographic sequence

SIGNATURE t_2 t_3 t_4 t_5	ARRANGEMENT	RULE NO.	k'	$t_{k'}$
1 1 1 1	1 2 3 4 5	1		
2 1 1 1	2 1 3 4 5	2	3	1
	(3 1 2 4 5)			
1 2 1 1	1 3 2 4 5	1		
2 2 1 1	3 1 2 4 5	2	3	2
	(3 2 1 4 5)			
1 3 1 1	2 3 1 4 5	1		
2 3 1 1	3 2 1 4 5	2	4	1
	(4 2 1 3 5)			
1 1 2 1	1 2 4 3 5			
. . . .	. . . . .			

It can be seen that in the lexicographic sequence there is a complete generation of $(k-1)!$ arrangements of the first $k-1$ marks before the kth mark is involved. Then there is a single transposition followed by a reversal of the first $k-1$ marks. This reversal, if carried out without the single transposition, would have completed generation of the first $(k-1)!$ arrangements of marks and restored the identity. This process is repeated k times, whilst each of the first k marks occupies the kth position, before the $(k+1)$th mark is involved. It follows that, if S_k is the number of transpositions involved in complete generation of the kth subsequence before the $(k+1)$th mark is involved,

$$S_k = k(S_{k-1} + 1).$$

If this is used inductively in the calculation of S_n for a given n we must add a further term. In the final regeneration of the identity with n marks, only a complete reversal of marks is involved (see 3(ii) above). This only involves an additional transposition compared with reversal of $n-1$ marks if n is even. Hence, if S_{k-1} is truly the number of transpositions involved in generation of $(k-1)!$ arrangements of $k-1$ marks then $S_k = (S_{k-1} + 1)k - \delta(k)$ where $\delta(k)$ is one or zero

as k is odd or even. In this way we can successively evaluate

$$S_2 = 2$$
$$S_3 = (2 + 1)3 - 1$$
$$S_4 = ((2 + 1)3 + 1)4 - 1.4$$

and in general

$$S_n = (\ldots((2 + 1)3 + 1)4 + \ldots + 1)n$$
$$-(\ldots(4.5 + 1)6.7 + \ldots + 1)n \quad \text{for } n \text{ even}$$
$$= n!\left\{1 + \tfrac{1}{2}! + \tfrac{1}{4}! + \ldots + \frac{1}{(n-2)!}\right\}$$

Hence $S_n \to \cosh 1 \times n!$

$= 1{\cdot}583n!$ as n increases.

An explicit computer algorithm using these rules was published by the author as A.C.M. Algorithm 323 (Ord-Smith, 1968). This algorithm shows that, although involving 1·583 times as many transpositions as the Wells sequence, the rules for generating the sequence are so simple that very little additional time is taken in its construction. A certification and some discussion of this algorithm has subsequently been given by Leitch in Comm. A.C.M. (Leitch, 1969). Phillips has constructed a fast lexicographic algorithm requiring numerical and distinct marks and using no signature. In the case of non-distinct marks Algorithm 28 will produce only those distinct orderings of a higher lexicographical order within the numerical code values of the particular data representation. Algorithm 323 on the other hand will generate, from the initial marks ABCDE, all conventionally accepted dictionary orderings independent of the coded numerical values of the alphabetic characters. Since Algorithm 323 always produces $n!$ orderings some of these will be repeated if the marks are not distinct. Ord-Smith and Phillips algorithms comprise two more of the set of six fastest algorithms discussed in Part 2 (Phillips, 1967).

7. A pseudo-lexicographic algorithm

It is interesting to note that a slightly modified lexicographic sequence preserves most of its properties, including that of preserving the position of the kth element during generation of a $(k - 1)$th arrangement of marks, whilst demanding many fewer transpositions in its generation. This is obtained from the lexicographic rules simply by replacing 2(ii) by:

a reversal of the k' least significant position marks.

The recursive formula for S_n now becomes:

$$S_k = (S_{k-1} + 1 - \delta(k))k$$

and the number of transpositions $S_n \to \sinh 1 \times n! = 1{\cdot}178n!$ as n increases. The algorithm was published by the author as A.C.M. Algorithm 308 (Ord-Smith, July 1967) but an improvement to copy exactly the rules 1, 2, 3 given above, with the modification to 2(ii), slightly improves the performance (Ord-Smith, 1969). This algorithm is discussed further in Part 2 of this paper.

8. Conclusions

Evolution of permutation algorithms has led to the production of six which are to date the fastest published. Three of these have appeared in *The Computer Journal* (Boothroyd, 1967, Phillips, 1967) and three in *Communications of A.C.M.* (Trotter, 1962, Ord-Smith, 1967, Ord-Smith, 1968). Three are transposition sequence generators and three are lexicographic.

Each of these, given explicitly in Part 2, has been rewritten in a standard form to ensure that comparisons of essential methods are, so far as is possible, compiler-independent. In the process every opportunity has been taken to implement each algorithm in the most efficient manner and this has led to worthwhile improvements in some cases. The author is indebted to the work of Mr. J. Boothroyd of the Hydro-University Computing Centre, University of Tasmania in this connection and for discussions concerning much of the material of the paper.

References

Boothroyd, J. (1967). Algorithms 29, 30, *The Computer Journal*, Vol. 10, p. 310.

Coveyou, R. R., and Sullivan, J. G. (1961). Permutation Algorithm 71, *Comm. ACM*, Vol. 4, p. 497.

Gardner, M. (1966). *New Mathematical Diversions from Scientific American*, Simon and Schuster, New York.

Hill, G. (1968). *Computing Reviews*, Vol. 9, Review No. 13891.

Johnson, S. M. (1963). An Algorithm for Generating Permutations, *Math. Comp.*, Vol. 17, p. 28.

Langdon, G. G. (1967). Generation of Permutations by Adjacent Transposition, *Comm. ACM*, Vol. 10, p. 298.

Langdon, G. G. (1968). Letter to Editor, *Comm. ACM*, Vol. 11, p. 392.

Leitch, I. M. (1969). Certification of Algorithm 323, *Comm. ACM*, Vol. 12, p. 512.

Mok-Kong Shen (1962). On the generation of Permutations and Combinations, *BIT*, Vol. 2, p. 228.

Mok-Kong Shen (1963). Generation of Permutations in Lexicographic Order. Algorithm 202, *Comm. ACM*, Vol. 6, p. 517.

Ord-Smith, R. J. (1965). An extension of block design methods and an application in the construction of redundant fault reducing circuits for computers, *The Computer Journal*, Vol. 8, p. 28.

Ord-Smith, R. J. (1967). Generation of Permutations in Pseudo-Lexicographic Order, Algorithm 308, *Comm. ACM*, Vol. 10, p. 452.

Ord-Smith, R. J. (July 1967). Remarks on Algorithms 87, 102, 130, 202, *Comm. ACM*, Vol. 10, p. 453.

Ord-Smith, R. J. (Nov. 1967). Remarks on Langdon's Algorithm, *Comm. ACM*, Vol. 10, p. 684.

Ord-Smith, R. J. (1968). Generation of Permutations in Lexicographic Order, Algorithm 323, *Comm. ACM*, Vol. 11, p. 117.

Ord-Smith, R. J. (1969). Remark on Algorithm 308, *Comm. ACM*, Vol. 12, p. 638.

Peck, J. E. L., and Schrak, G. F. (1962). Permute, Algorithm 86, *Comm. ACM*, Vol. 5, p. 208.

Phillips, J. P. N. (1967). Algorithm 28, *Comp. J.*, Vol. 10, p. 311.

Rodden, B. E. (1968). In defence of Langdon's Algorithm, *Comm. ACM*, Vol. 11, p. 150.

Tompkins, C. (1956). Machine attacks on problems whose variables are Permutations, *Sec.* 3, *Proc. 6th Symp. App. Maths. Amer. Maths. Soc.*, McGraw-Hill, p. 198.

Trotter, H. F. (1962). Perm, Algorithm 115, *Comm. ACM*, Vol. 5, p. 434.

Wells, M. B. (1961). Generation of Permutations by Transposition, *Math. Comp.*, Vol. 15, p. 192.