On the scheduling of jobs by computer

By E. S. Page

The problem of deciding the best order in which to process a set of jobs in a factory has so far eluded theoretical solution except in very simple cases. A similarity between the scheduling problem and the sorting one suggests adapting well-known sorting methods. A scheduling method based on merging is shown to give results that are better than those from known methods.

Introduction

The scheduling of jobs in a factory is one of the problems which offers rich rewards for any satisfactory solution. The aim is to allocate the order in which a set of jobs shall be passed through the factory so that some desirable criterion should be achieved. Each of these jobs requires several operations to be performed, and in many cases these operations can be performed only by certain machines. This is, of course, a combinatorial problem and, like many other similar problems of practical and theoretical interest, the number of possible combinations arising can be very large. Even one of the simplest types of this problem is still formidable. The simple case we consider here is where all the jobs have the same number of operations to be performed and all in the same order. There is also to be the further restriction that once an order of processing shall have been decided, that order shall be maintained throughout all the operations. Thus, the problem is only that of ordering the jobs themselves, and not of ordering the operations within the jobs.

For example, if there are just three jobs *A*, *B*, *C* each needing work on three machines in the same order we just have to decide which of the six permutations of (*A*, *B*, *C*) should be selected for the processing order. Once this is chosen the jobs follow one another through all the machines in the same order. The situation can be represented on a chart (Fig. 1). In the diagram *A* requires 10 units of time on the first machine, 3 on the second, and 4 on the third, and the corresponding times for *B*, *C* are (2, 5, 8) and (4, 4, 4). The letter *Q* indicates that a job is waiting for attention and an *I* that a machine is idle. For the order *ABC* the jobs are finished in 30 time units; other orders can finish more quickly (e.g. *BCA*) and the queueing and idle patterns change.

There are three common methods of attack on combinatorial problems. That which springs readily to the mind of anyone with a high-speed computer at hand is the enumeration of the possibilities and then the selection from them of the best order. Unfortunately, even for such a small number of jobs as ten or a dozen, the number of possible orders is very large, 12!=479,001,600, and this size of number commands respect. The mathematician would prefer a theoretical attack, but so far only simple special cases of even this simple problem have been successfully solved. Johnson (1954) has solved the problem for one cost function for any number of jobs which require operations on only two machines and has extended this to a special case where there are three machines. Further generalizations seem to lead into extremely difficult mathematics. Some of these problems have been formulated as an integer linear-programming problem, for which algorithms exist, but which are as yet too lengthy for the sizes of problem of practical interest. Several practical complications have not yet been included in this formulation. The last method, favoured by practical men, is to play a hunch—that is to use the method dignified by mathematicians with the name "heuristic." It has occasionally been suggested that the alternative of trying a few of the combinations at random might be worth while, but this is really only a special case of the heuristic method where counsels of despair prevail.

Various methods that are operated in practice usually stipulate some simple rule-of-thumb which is to be followed in allocating the orders: for example, those jobs which are nearest their promised delivery date, or perhaps the most valuable jobs, should be given priority. The success of rules like these are judged according to certain criteria of efficiency, and these same criteria can be used to compare the results obtained by employing different rules. In a recent paper (Page, 1961) it has been suggested that some of the methods used for sorting data in an automatic computer might be applied to this scheduling problem. It was shown that when the criterion of efficiency is the total completion time of the jobs, a method based upon the merging technique gives results which compare very favourably with some other methods. Here we consider the merging method further. We take another cost function, the total idle time in all stages of the work. For small numbers of jobs, up to seven, all possible orders are examined and both the optimum order and optimum cost for this cost function are compared with that obtained by the merging method. For larger numbers of jobs, merging is compared with another method based on a sequential allocation of jobs to the order, and also with a Monte Carlo approach,

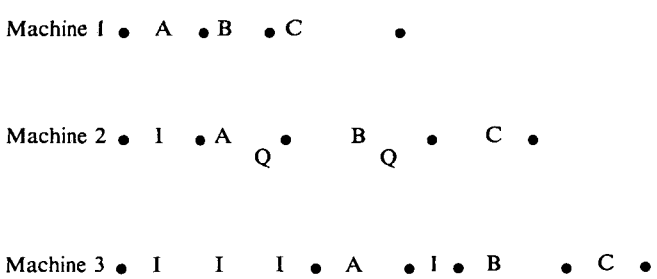

Fig. 1.—Chart of machine loading

which, in spite of the earlier comments, has some initial attraction; in both cases merging performs appreciably better than the alternative methods.

Methods

The merging method for scheduling

A method for sorting data commonly used on an automatic computer is merging. When we are placing items in order of key number, using merging with two strings of fixed length, we build up first pairs, then quartets, octets, and so on in the correct sequence until the complete ordering is reached (see e.g. Friend, 1956). At each stage we use the knowledge that the two sets we are merging are in the correct sequence themselves, so that we only need to consider the relative order of the two current leading items of the sets. Similar principles can be adopted to provide an order for the scheduling problem which is usually a good one even if not the optimum, and further, which actually attains the optimum in a certain range of cases (Page, 1961). The aim is to build up sequences of jobs which fit together well in the sense of the criterion of efficiency we are using, and to form from these sequences longer ones and yet to allow a certain amount of further rearrangement if it is advantageous.

Suppose that we have some cost function by which to measure the effectiveness of different orders for any number of jobs. This cost function might be the total value of work in progress, the completion time of all the work, the total delays, the total idle time of machines in the various stages, or similar and perhaps more complicated functions. Although it may be difficult to express this cost function explicitly it is clear that such a function exists whenever there is a criterion for saying that one order of jobs is better than another order. We suppose that we are given some such criterion or cost function, and that we have to schedule a set of jobs, distinguished say by letters $a, b, c, \ldots$ The merging method proceeds as follows.

Consider the first two jobs presented a, b; decide according to the given criterion which of the orders (a, b), (b, a) is preferable for processing and record the better order, say (a, b). Repeat this stage for all succeeding non-overlapping pairs of jobs and so obtain a set of pairs like (a, b), (d, c), (f, e), (g, h), . . . In order to maintain a correspondence with the merging method for ordering items of data by key numbers, we can think of these pairs of jobs in two strings,

$$(a, b), (f, e), (i, j) \ldots$$
$$(d, c), (g, h), (k, l) \ldots$$

If the number of jobs is odd we can insert a dummy one, requiring no processing, to complete the pair. The next stage is to form quartets from

$$\{(a, b), (d, c)\}, \{(f, e), (g, h)\}, \ldots$$

We consider the orders (a, d), (d, a) and pick the better. Suppose it is (a, d). Then we start our quartet with job a, and now proceed to find an order for the remaining three jobs. We compare the orders (a, b, d) and (a, d, b); we choose the better one, say (a, b, d), and take its first two jobs as the first two of the quartet. Finally, we compare (a, b, d, c) with (a, b, c, d). The merging of data corresponds closely with this method of scheduling; at each stage the current heads of the two strings are examined, but for the job application both orders have to be tried and the cost functions calculated, whereas it is only necessary to compare the key numbers in the context of data-sorting. We proceed from quartets to octets, and so on until an order is obtained for the whole number of jobs.* A flow chart is shown in Fig. 2.

It is clear that this method is one which places importance on jobs "fitting together"; when a good fit is obtained at an intermediate stage it is likely to be maintained subsequently, but there is still the opportunity provided for a change in case a better fit can be found. Naturally, not all the possible orders are generated and examined; only a subset of these orders is considered, but it is a subset which seems to have a reasonable chance of containing a good order even if not the best one. The bigger the subset we look at the better are our chances of getting a good final order, but we have to take care that the computing involved does not increase too rapidly. We can get an idea of the rate of increase of the computing work with the number of jobs. Let u_n be the amount of computing required for 2^n jobs. The time to calculate many cost functions for a sequence of jobs will be about proportional to the length of sequence; thus when two pairs are to be merged into a quartet there will be two sequences each of 2, 3, 4 jobs to be considered, and similarly for later stages in the merge. Accordingly,

$$u_{n+1} = 2u_n + 2(2 + 3 + \ldots + 2^{n+1})$$

since we have to merge to get two sequences of 2^n jobs and then to combine these. Hence

$$u_n = 2^{2n+1} + (n - 4)2^n + 2. \qquad (1)$$

Thus we might expect the computing work to increase by a factor of about four when the number of jobs doubles. However, at each comparison we have two sequences of jobs which are identical save for the last pair which are in their two possible orders. Consequently, for some interesting cost functions, for example, the total time of completion, only little extra computation would be necessary for each comparison if it is possible to store the intermediate information. The rates of increase in computation for a Pegasus program using the total time criterion are shown in Table 1. No particular attempt was made to streamline the program, but

* There is a slight similarity between this approach and that by Dynamic Programming. At each stage, however, Merging selects the optimum from only a very restricted range of possibilities, while d.p. makes *the* optimum selection at greater effort, but in return assures that the final order is optimum, which Merging does not.

it is seen that the actual rate of increase as the number of jobs doubles is by a factor of about three and not four as suggested by (1).

Table 1

Rate of increase in computation for scheduling by merging

TIME RATIO \ MACHINES	3	4	8	CALCULATED FROM (1)
8 jobs : 4 jobs	2·3	2·7	2·5	4·7
16 jobs : 8 jobs	3·1	2·8	2·8	4·2
32 jobs : 16 jobs	3·0	3·0	3·0	4·1
64 jobs : 32 jobs	3·1	3·1	3·1	4·0

Other methods for scheduling

The other methods with which we compare merging are as follows.

Pairing

This method, a variant of merging which tries many fewer combinations, was suggested in an earlier paper (Page, 1961). A pair, quartet, etc., is regarded as permanent once it is made, and subsequently only the two orders of the sets so far achieved are compared. For example, if the pairs (a, b), (c, d) are fixed at the second stage we only compare orders (a, b, c, d), (c, d, a, b). Although the number of orders tried is much less than in merging, some of them are different, and it is occasionally possible to get a better result from pairing than merging. Usually, however, merging does better than pairing, but it takes more computing. A similar argument to that used to derive (1) shows that the time for pairing 2^n jobs will be proportional to $n2^{n+1}$.

Exchanging

Starting from an arbitrary order, adjacent jobs are interchanged, proceeding from first to last, and the exchanged order retained if an improvement is made in the cost. The exchanging continues until an order is maintained through a whole cycle. This method is one of the simplest conceptually, and is perhaps the first which comes to mind for improving a given order. It has, however, disadvantages in its application quite distinct from the results it achieves which we find to be disappointing (cf. Page (1961), and the section on results in this paper). First, and most important, the number of exchanges required to complete the process as outlined is not known in advance, and can be very large for quite moderate (30–60) numbers of jobs. The improvements in cost that are made are usually each small, and a more intelligent method of selecting combinations to try would get quicker results. Further, no provision is made for the exchanging of groups of jobs,

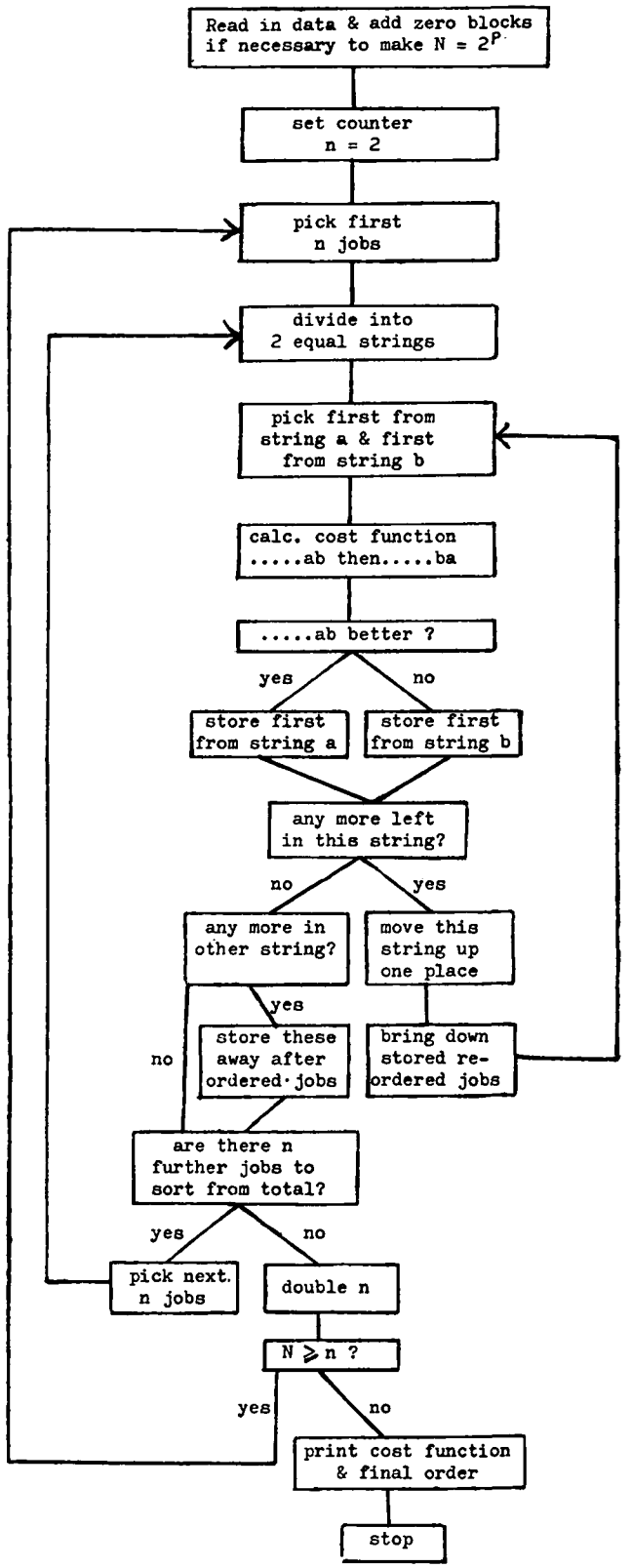

Fig. 2.—Flow diagram of scheduling byMerging

and this approach may help where exchanging adjacent jobs does not. It is not practicable to keep a record and subsequently explore all cases of exchanges which leave the cost unchanged. In spite of all these comments the method, which has been suggested several times, needs to be tried if only to demonstrate further that other approaches get better results more quickly. A possible exception may be when the starting order is a good one derived by some other method.

Selection

This method is the one used in an experimental program by J. D. Croston of Kodak Ltd. In this method each of the jobs is placed in the first position for processing, and the total idle time to the completion of the job is calculated; that job with least idle time is selected and fixed in first position. Then each of the remaining jobs is tried in the second position and the total idle time for the two jobs is found and the least one again selected, and its job allocated. At each stage there is one less job to try in the next position. This method clearly corresponds to the Linear Selection method of ordering items of data (e.g. Friend, 1956); the data are examined to find the first item, and then the second, and so on. This is usually a very inefficient method of sorting, and it is interesting to see how it fares when applied to scheduling.

Monte Carlo

Given the task of scheduling n jobs this method merely selects permutations at random from the $n!$ possibilities and picks the best one. There is scope for examining sequential rules for deciding how many trials to make, but here we consider only a cruder approach.

Several methods of picking random permutations, just as with scheduling, can be obtained from the adaptation of methods of sorting. One of the simplest, and one that is widely used by hand, depends on the Selection method. A random integer in a range larger than 1 to n is generated, and if it corresponds to an item, that item is selected for the first member of the permutation and is no longer available; further random integers are generated until the whole permutation is formed. At some stage or stages in the hand method, the items are renumbered in order to avoid rejecting too many random integers—in this case usually selected from a table. This procedure is conveniently coded for a binary computer so that renumbering occurs as the number of items unallocated reaches a power of 2. The number of bits used from the random-number generator also falls by one.

If at some stage we are using k bits from a random-number generator to pick the next item from r remaining items, we need $2^k/r$ trials on the average. Hence the average number of number generations required for the whole permutations is

$$S = 1 + 2^2(\tfrac{1}{3} + \tfrac{1}{4}) + \ldots + 2^{k-1}\left(\frac{1}{2^{k-2}+1} + \ldots + \frac{1}{2^{k-1}}\right) + 2^k\left(\frac{1}{2^{k-1}+1} + \ldots + \frac{1}{n}\right).$$

Using the Euler–Maclaurin summation formula and crude approximations we have

$$S \sim 2n \log 2$$

or better,

$$S \sim 2^k\{\log 2n - \log (2^{k-1} + 1)\}$$

where $2^{k-1} < n \leqslant 2^k$. The point to be made is that the time for the construction of the random permutation by this method increases linearly. The items selected one by one can be marked by the appropriate bit in a word, and if the word-length allows, there can also be a record of the number of unallocated items within that word. For example, the Pegasus word-length of 39 bits allowed the items to be marked in groups of 32, while leaving enough bits for the count of items. Computers with a fast "BITS" operation might find the recording of the count unnecessary.

One simple rule for deciding the number, N, of trials to make is to choose N so that there is at least a specified probability α, that at least one of the trials shall lie in the best $100P\%(0 < P < 1)$ of the distribution over the $n!$ possible permutations. Then

$$1 - (1 - P)^N \geqslant \alpha,$$

so that we need to take $N \geqslant \log (1 - \alpha)/\log (1 - P)$. The drawback is that this is not the type of rule that is really needed; even after the best permutation tried has been chosen there is no information at all about how much better the best order is. The only consolation is that we know about how hard it will be for anyone else to find a better order than we have, if they search by this method.

Simulation results

For all the examples we consider, the times required on the various machines were generated by the multiplicative method for pseudo-random numbers to give integers in the range 1–16. This just happened to be a convenient method of obtaining data which would provide a good test of the methods of scheduling, in the difficult type of problem where the total loadings of all the machines are about the same.

Small numbers of jobs

First we obtain by enumeration the optimum cost for a small number of jobs. For 7 jobs there are 5,040 orders, and on our Pegasus computer, the complete enumeration took about half an hour when each job had to be processed on seven machines. The construction of the permutations was achieved by the method given by Wells (1961). For each set of machine times the jobs were scheduled by the five methods—Merging, Pairing, Exchanging, Selection, and Monte Carlo. The number of jobs concerned here is very small, much smaller than the number in most practical situations, but the results displayed in Table 2, show that the methods tried give orders of jobs costing from 7%

Table 2

Comparison of costs for scheduling 7 jobs each on 7 machines

Cost function: total wasted time. Averages over 25 sets of 7 jobs

METHOD	MERGING	PAIRING	EXCHANGING	SELECTION	MONTE CARLO	BEST
Average cost	231·2	251·8	253·0	270·3	240·8	215·4
Percentage exceeding best	7·3	16·9	17·5	25·5	11·8	—
Computing time (seconds)	20	12	21	10	26	1925

to 25% more on the average than the best order, and in individual cases, up to 60% more than the best. Thus even in such an apparently simple and small problem, big differences in cost can occur by a poor approach.

The figures on the first three methods (M, P, E) confirm the previous ones (Page, 1961) for up to 32 jobs, 3, 4, or 8 machines, and cost function taken as the total time of completion. Merging again shows up best, and again, Pairing is about as good as Exchanging and takes less time. Previously it had only been possible to compare the costs with a crude lower bound; it now appears that the orders achieved have costs which are much nearer to the optimum than had been suspected. It is not possible in this case to say whether the percentage differences from the best cost will decrease as the number of jobs increases.

The next point concerns the very poor performance of the Selection method; it is quick to schedule in this way and so perhaps is feasible even by hand, but the results it achieves are so far below that of merging that the extra computing will be justified in all but a few applications.

The Monte Carlo method was used for 20 trials on each set of jobs, and the best order selected for comparison. The computing time was a little longer than that for merging, and the order not as good. However, experience with other problems indicates that if Monte Carlo has any advantages at all it will be on large rather than small problems. We return to this question later; at present this crude Monte Carlo approach has no advantage over merging.

The conclusion we draw is that Merging provides an order with a cost usually nearer to the optimum than the other methods considered for small numbers of jobs.

The actual order produced by any of the methods will not necessarily have much in common with the optimum order (or one of the optimum orders if there are several with equal costs). A measure of the agreement that can be adopted is the number of adjacent pairs of jobs in the order achieved that are in the same order as the optimum. In the Appendix the distribution of this number of successors in a random permutation agreeing with a specified permutation is derived. This distribution is shown to tend rapidly, as the number of items increases, to the Poisson form with unit mean so that

$$\text{prob}\{r \text{ correct successors}\} \doteqdot e^{-1}/r!$$

If a permutation with r successors is λ times as likely to occur as one with $(r-1)$ successors, the distribution of r tends to Poisson with mean λ. Table 3 shows the average number of correct successors for the four deterministic methods in the 25 sets used.

Table 3

Average number of correct successors

MERGING	PAIRING	EXCHANGING	SELECTION
3·2	1·7	2·4	1·8

The amount of agreement achieved by all the methods is better then than that for a random permutation at a high level of significance, and Merging is clearly the best of the methods by this criterion. In order that a random sample of m permutations should contain at least one with k or more correct successors with probability P, we must have, approximately,

$$1 - P = \left\{ \sum_{k}^{\infty} e^{-1}/r! \right\}^{m};$$

for $P = 0{\cdot}99$ and $k = 3$ we need $m \doteqdot 55$, and thus the Monte Carlo method would need more than three times as much computing as merging for comparable results.

The distribution of the cost function over the 7! permutations was formed for a few of the enumerations. Most of the distributions were unimodal, roughly symmetrical; others had nearly constant frequencies over the middle of the range and sometimes appeared bimodal. In nearly all cases it was clear that an order achieving a cost within 10% of the optimum would fall into the top one or two per cent of the distribution.

Larger numbers of jobs

The Merging, Pairing and Exchanging methods have been compared (Page, 1961) using a total time of completion cost function for as many as 64 jobs and 8 machines. The pattern of these results is just that shown by the present comparisons with a different function for small numbers of jobs; Merging gets the

Table 4

Comparison of Merging and Selection methods of scheduling:

total wasted times

METHOD	NUMBER OF JOBS 8	16	32	128
Merging	243	277	337	539
Selection	297	353	425	631
Percentage differing (S-M)/M	22%	27%	26%	17%

best results, while pairing and exchanging do about as well as each other, and the former takes less computing. Accordingly, we do not continue these comparisons.

The Merging and Selection methods have been tried on sets of artificial machine times. Fifty sets of 8, 16, and 32 jobs, and ten sets of 128 jobs using seven machines have been scheduled by both methods. The means of the total idle times are shown in Table 4.

It is seen that the Merging method produces an order which has 17–25% less wasted time on the average than Selection. The Merging method produces a better order than Selection in over 90% of the cases tried, and where its order is worse it is only very little worse. The average gain is so large that it is certainly worth considering applying the Merging method in spite of its greater computing time.

As the size of problems increases Monte Carlo methods can become more efficient than conventional deterministic methods—for example, in the solution of sets of simultaneous linear equations (Curtiss, 1954). We have performed more Monte Carlo runs on the sets of 128 jobs previously used with the results shown in Table 5, together with the computing time on a Ferranti Pegasus. For each set, 300 random permutations were selected and the order with the best cost Selected; this number of trials gives a probability of about 0·95 that the best order found lies in the best 1% of the distribution. Table 5 shows a very poor achievement for Monte Carlo compared with Merging; there is little difference in the results for Pairing and these Monte Carlo trials, and the computing for Pairing is very small. It is clear that we can achieve much better results than Monte Carlo in the same computer time, or about the same results with only a fraction of the effort.

Much less efficient than Merging is Exchanging; the computing time taken varies according to the number by cycles of exchanging necessary, but the order of time is about the same. The only (slight) thing that can be said for Exchanging is that it is easy to program, but so too is Selection, which is quicker and gives better results. Accordingly, we discard both Exchanging and Monte Carlo. Pairing takes such a short time that there is something to be said for it and also for the slightly slower but more effective ordering obtained by Selection. Merging gives appreciably the best order at the cost of most computing, but the additional computing expenditure will usually be amply regained by the increased efficiency of production.

Conclusion

It must be repeated that, while the Merging method will give an exactly optimum order of jobs for some cost functions and for some sets of jobs, usually the order produced is one which is "good" or "very good" but not "best." Even so, it is suggested that the method has some merit in practice; it is easy to apply, it can include a wide range of practical conditions, such as groups of identical machines, machines identical in function operating at different speeds and so on, and it can use a wide range of cost functions—indeed, all that is required is a rule for deciding whether one order is better than another. Jobs uncompleted after one scheduling period can be inserted for scheduling in the next, and provision made in the cost function to ensure that they are not displaced in the production line. This approach which seeks (rather empirically) a good method over a wide range of conditions is lent respectability by the way it seems to conform to the views expressed in a more general context by J. W. Tukey (1961). In so many practical situations (all?) the criterion for optimality is known only approximately, and lack of precision may enter in other ways; there seems little point in labouring heavily to obtain an exactly optimum solution according to an approximate criterion, and instead a method may be welcomed which compares well with several other methods when judged by several different criteria.

Acknowledgement

I wish to thank Miss M. G. Robson for programming and running all the calculations described above.

Table 5

Comparison of methods on 128 jobs, 7 machines

Monte Carlo: best order in 300 trials

	PAIRING	MERGING	EXCHANGING	SELECTION	MONTE CARLO
Total wasted time	729	539	882	631	725
Percent over merging	35%	—	64%	17%	34%
Computer time (minutes)	4	35	30–45	9·5	115

References

CURTISS, J. H. (1954). "Comparison of Classical and Monte Carlo Methods for Simultaneous Linear Equations," *Symposium on Monte Carlo Methods*, New York: Wiley, pp. 191–233.

DAVID, F. N., and BARTON D. E. (1962). *Combinatorial Chance*, London: Griffin.

FRIEND, E. H. (1956). "Sorting on Electronic Computer Systems," *J. Ass. Comp. Mach.*, Vol. 6, pp. 169–74.

JOHNSON, S. M. (1954). "Optimal Two- and Three-stage Production Schedules with Set-up Times Included," *Nav. Res. Log. Quart.*, Vol. 1, pp. 61–68.

PAGE, E. S. (1961). "An Approach to the Scheduling of Jobs on Machines," *J. Roy. Statist. Soc.*, B, Vol. 23, pp. 484–93.

TUKEY, J. W. (1961). "The Future of Data Analysis," *Ann. Math. Statist*, Vol. 43, § 1.5.

WELLS, M. B. (1961). "Generation of Permutations by Transpositions," *Math. Comp.*, Vol. 15, pp. 192–5.

WHITWORTH. W. A. (1901), *Choice and Chance*, 5th edition, Cambridge: Deighton Bell.

Appendix: the agreement of permutations

A standard permutation of n objects $(A_1, A_2, \ldots A_n)$ is given together with a sample permutation $(a_1, \ldots a_n)$ where the a's are some rearrangement of the A's. We wish to find the distribution of the number of pairs (A_iA_{i+1}). Let this number of correct successors be r; we derive the distribution of r under the null hypothesis that all sample permutations have the same probability $1/n!$ of selection.

Let the number of permutations of n items with r items following the same one as in the standard permutation, i.e. r successors be $u_{n,r}$; then on the null hypothesis the probability of obtaining r successors in a randomly chosen permutation is

$$p_{n,r} = u_{n,r}/n!$$

If there are r successors there are n-r-1 places at which the order differs from the standard; these places can be selected in

$$\binom{n-1}{n-r-1} = \binom{n-1}{r}$$

ways. There are thus $n - r$ sections of the standard order which can be arranged in any sequence which does *not* preserve the order of two adjacent sections of the standard, i.e. in $u_{n-r,0}$ ways. Hence

$$u_{n,r} = \binom{n-1}{r} u_{n-r,0}.$$

The permutations, $u_{n,0}$ in number, of n items which have no successors can be obtained from permutations of $n - 1$ items with 0 or 1 successors by inserting the nth item in a suitable position. Thus A_n can be put first or after any of the $n - 1$ items *except* A_{n-1} in any of the $u_{n-1,0}$ permutations with no successors; and A_n must separate the pair in correct order in the $u_{n-1,1}$ permutations with one successor. Hence

$$u_{n,0} = (n-1)u_{n-1,0} + u_{n-1,1},$$

and so $$u_{n,0} = (n-1)u_{n-1,0} + (n-2)u_{n-2,0}.$$

By inspection, one solution is $n! + (n-1)!$ and the general solution follows by variation of parameters. We have

$$u_{n,0} = \frac{(n+1)!}{n}\left[\frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^{n+1}}{(n+1)!}\right],$$

and $$u_{n,r} = \frac{(n-1)!}{r!}(n-r+1) \times \left[\frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^{n-r+1}}{(n-r+1)!}\right]$$

Whitworth (1901) derives $u_{n,0}$ by enumeration of the possibilities, and David and Barton (1962) by another method.

The mean number of successors μ_1' is given by

$$\begin{aligned}\mu_1' &= \sum_{r=0}^{n-1} r p_{n,r} \\ &= \sum_{s=0}^{n-2} \frac{1}{n}\frac{(\overline{n-1}-s+1)}{s!} \times \left[\frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^{\overline{n-1}-s+1}}{(\overline{n-1}-s+1)!}\right] \\ &= \frac{n-1}{n}\sum_{s=0}^{n-2} p_{n-1,s}.\end{aligned}$$

Hence $$\mu_1' = 1 - \frac{1}{n}.$$

Similarly we can snow that the factorial moments

$$\mu_{(k)}' = E\{r(r-1)\ldots(r-k+1)\} = 1 - k/n.$$

For n large and $r \ll n$, $p_{n,r}$ is approximately the probability of r in a Poisson distribution of unit mean. The approximation is best for $r = 1$, while the distribution has a slightly higher proportion of zeros than the Poisson.

On an alternative hypothesis that permutations with r successors are λ times as likely to occur as those with $(r-1)$, we can show that the expected number of successors is

$$S_n = (n-1)\lambda C_{n-1}(\lambda)/C_n(\lambda)$$

where $$C_n(\lambda) = \sum_{r=0}^{n-1} \lambda^r u_{n,r}$$

and $C_n(\lambda)$ satisfies a recurrence relation

$$C_n(\lambda) = (\lambda + n - 1)C_{n-1}(\lambda) + (n-2)(1-\lambda)C_{n-2}(\lambda).$$

The distribution approximates to a Poisson with mean λ. Thus a range of approximate procedures for tests and estimation of successors is available based on methods for the Poisson distribution.