

J. Opl Res. Soc. Vol. 44, No. 4, pp. 375-382
Printed in Great Britain. 

0160-5682/93 $8.00+0.00


# Improving the Efficiency of Tabu Search for Machine Sequencing Problems

COLIN R. REEVES
School of Mathematical and Information Sciences, Coventry University

Tabu search (TS) is becoming increasingly recognized as an efficient way of finding high-quality solutions to hard combinatorial problems. It may be described as an intelligent meta-heuristic for controlling simpler local search procedures. However, reported applications have often used a 'brute-force' approach without considering the most effective use of the computing effort available. This paper intends firstly to give a basic introduction to the ideas of TS, and then it will report some computational experiments on TS in the context of machine sequencing. These have shown that it is important to define the balance between exploration of the solution space and exploitation of the information obtained. The results will be compared with those obtained from a proven Simulated Annealing (SA) algorithm, which tends to confirm the general opinion that when implemented efficiently, TS is a more effective search paradigm than SA.

*Key words:* tabu search, machine sequencing, simulated annealing

## INTRODUCTION

It is now widely recognized that finding the optimal solution to large combinatorial problems is virtually impossible. The development of the concept of NP-completeness (Garey and Johnson[1]) has meant that much research has been concentrated on the design of heuristics.

A popular heuristic with wide applications is that of neighbourhood search. The idea is very simple: one starts with a (sub-optimal) solution to a particular problem and searches a defined neighbourhood of this solution for a better one. Having found one the process re-starts from the new solution. The procedure iterates in this way until no improvement can be found to the current solution. This final solution may well not be the global optimum—usually it is merely a local optimum.

The most famous example of this approach is the $\lambda$-optimal heuristic for the Travelling Salesman Problem of Lin[2], where the neighbourhood is defined by deleting $\lambda$ edges in the current tour and attempting to re-connect the tour in such a way that the total tour length is reduced. Similar approaches have been reported for many combinatorial problems including the subject of this paper—the Flowshop Sequencing problem (e.g. Baker[3]).

The phenomenon of local optimality is one that is well-known and various methods of dealing with it have been used. An obvious approach is to enlarge the neighbourhood—for example, by moving from a 2-optimal heuristic to a 3-optimal one. Another popular idea is to start the whole procedure again from different initial solutions, in the hope that it will converge to different local optima. The best of the many solutions thus found is then chosen.

A third option is to allow 'uphill moves', whereby the solution from which the search is re-started is allowed to be worse than the previous one. There has to be some restriction on accepting such moves, otherwise the procedure would amount to a search of the whole solution space, but by accepting some there is a chance that the process can 'climb out' of a local optimum and find a better solution.

The methods of Simulated Annealing and Tabu Search have received considerable coverage in the last few years, and there have been several applications to various NP-hard problems which have achieved considerable success. These methods are both examples of this third approach to avoiding local optima.

Recently, both these methods have been applied to machine sequencing problems. Osman

*Correspondence: C. R. Reeves, School of Mathematical and Information Sciences, Coventry University, Coventry, UK*

and Potts[4] described a simulated annealing approach to the permutation flowshop scheduling problem. This problem, which is known to be NP-hard[5], is defined as follows:

There are $n$ jobs to be processed (in the same order) on $m$ machines. The object is to find the permutation of jobs which will minimize the 'makespan', i.e. the time at which the last job is completed on machine $m$. The problem investigated in this paper is conventionally given the notation $n/m/P/C_{\max}$.

In mathematical terms, if we have processing times $p(i, j)$ for job $i$ on machine $j$, and a job permutation $\{J_1, J_2, \ldots, J_n\}$, then we calculate the completion times $C(J_i, j)$ as follows:

$$C(J_1, 1) = p(J_1, 1),$$

$$C(J_i, 1) = C(J_{i-1}, 1) + p(J_i, 1) \quad \text{for } i = 2, \ldots, n,$$

$$C(J_1, j) = C(J_1, j-1) + p(J_1, j) \quad \text{for } j = 2, \ldots, m,$$

$$C(J_i, j) = \max\{C(J_{i-1}, j), C(J_i, j-1)\} + p(J_i, j) \quad \text{for } i = 2, \ldots, n; \quad j = 2, \ldots, m,$$

$$C_{\max} = C(J_n, m).$$

(Notice that the job order is assumed to be the same on all machines; in the more general flowshop sequencing problem $n/m/F/C_{\max}$, this is not necessarily so.)

## SIMULATED ANNEALING

Since its introduction by Kirkpatrick *et al.*[6], the method of Simulated Annealing has received considerable attention, and there have been two recent applications[4,7] to the $n/m/P/C_{\max}$ problem.

The idea is simple—during the search of a neighbourhood, 'downhill' moves are always accepted while 'uphill' moves are accepted with a probability $P$, given by

$$P = \exp(-\Delta v/T)$$

where $\Delta v$ is the increase in value of the objective function, and $T$ is a parameter called the 'temperature'.

The temperature is slowly reduced from an initial setting which allows most uphill moves until, at its final value, an uphill move is extremely unlikely. The sequence of temperatures employed is called a 'cooling schedule'.

Osman and Potts[4] experimented with various control parameters and neighbourhoods, and found that the best results were obtained from a 'shift' or 'insertion' neighbourhood searched in a random order, using the following parameter settings:

$$\text{Initial temperature } T_0 = \sum_{i,j} p(i, j)/(5mn)$$

$$\text{Final temperature } T_f = 1$$

$$\text{Cooling schedule } T_{k+1} = T_k/(1 + aT_k)$$

The cooling schedule adopted is essentially that proposed originally by Lundy and Mees[8], while the value $a$ is effectively defined by $f$, the number of objective function evaluations to be carried out:

$$a = \frac{T_0 - T_f}{(f-1)\,T_0 T_f}$$

The 'best' value of $f$ was found by experiment to be

$$f = \max\{3300 \log n + 7500 \log m - 18\,250, 2000\}$$

The effect of this choice of parameters is that SA can be implemented in $O(mn \log(m + n))$ steps, so the time complexity of the procedure will grow at a very acceptable rate.

A slightly different implementation of SA was reported by Ogbu and Smith[7]; in a more recent paper[9] they show that a comparison of the two approaches gives broadly similar results in about the same computation time, and they suggested that SA should now be the method of choice for this class of problem.

## TABU SEARCH

A more recent idea, introduced by Glover[10,11], is to consider all moves both 'up' and 'down' except for a certain prohibited or 'tabu' set. The 'tabu' moves are kept as a list of length $L$ which effectively prevents the $L$ most recent moves from being reversed. Each time a move is made its 'inverse' is added to the list, while the oldest move on the list is dropped. If the length $L$ is too small, there is a chance that the method will simply cycle round the same few sequences indefinitely, but this seems not to occur if $L$ is large enough. This can be thought of as simulating a form of 'short-term memory', so the procedure will recognize (and avoid) areas of the solution space that it has already encountered. Glover[11] also discussed ways of simulating 'long-term memory', and procedures for overriding the basic algorithm by using 'aspiration levels'. These are sometimes useful, but the focus of this research is on the basic method.

There are several ways to implement the basic idea. The value of $L$ is clearly a key parameter, although there is a lot of empirical evidence that, provided $L \geqslant 7$, there is not too much dependence on its actual value. (In all the experiments reported here, $L$ was set at 7. No cycling was ever detected; as a check, some runs were also made with larger values of $L$, but these showed no consistent improvements or deteriorations in solution quality, so it was concluded that $L = 7$ was a reasonable choice.) There is also the problem of what type of neighbourhood to use, and there is a clear need for some form of stopping criterion, to limit the number of iterations.

A further question relates to the nature of the move from one sequence to another. In some neighbourhood search applications, an ordered search is made of a neighbourhood, where the heuristic looks for the steepest descent by finding the move which offers the greatest improvement of all moves in the complete neighbourhood. An alternative is to look for the 'fastest' descent by accepting the first move which gives an improvement.

Using steepest descent in the context of 'tabu search' (TS) is simple to implement, and this was the starting point of the development reported here. But it is a very slow process, as $O(n^2)$ function evaluations are needed before each move. In fact, unless the number of iterations is large enough to reach a local optimum before an acceptable amount of computing time has been expended, the tabu search proper never actually starts.

One way of coping with this is to use a better heuristic to generate a good initial solution. By adopting this as a starting solution, the TS method can be speeded up considerably. Recently, Widmer and Hertz[12] developed a TS algorithm using a heuristic of their own to provide a starting point. They then applied TS, where it appears that they used a steepest descent on a 'shift' neighbourhood with either $n$ or $(n + m)$ iterations. However, there is some ambiguity here: Taillard[13], whose paper will be discussed later, interprets Widmer and Hertz as accepting a non-tabu downhill move immediately it is discovered, only searching the whole neighbourhood if no such moves can be found. If the latter is the case, the best (uphill) non-tabu move is then used.

The effect of this is that the time complexity of their procedure cannot be completely characterized. If no downhill move is found, so that the whole neighbourhood is searched, then an iteration performs $O(mn^3)$ computations, so that the overall complexity could be as much as $O(mn^4)$. There will be some iterations when considerably less work is done, but overall, the Widmer/Hertz approach looks rather uncompetitive with the Osman/Potts SA algorithm. A comparison of the two methods was carried out, in which the Widmer/Hertz (WH) heuristic using Taillard's interpretation was allowed the same number of function evaluations used by Osman and Potts. The problem instances were generated with uniformly distributed processing times between 1 and 100, and with various numbers of jobs and machines. There were seven groups of problems, each consisting of six instances. The heuristics were programmed in PASCAL, and run on Coventry University's Sequent S82 computer. They were given the same maximum number of function

evaluations prescribed for the Osman/Potts SA method. The actual computer times needed for a single run on a given problem were (as expected) very similar for each method, ranging from 10 seconds for a 20/5 problem to 11 minutes for a 75/20 problem.

Table 1 displays the mean percentage difference (MPD) between the makespans found by WH and OP, and the number of instances (NI) out of six for which the OP makespan was superior. A positive value for MPD and a NI 'score' of more than 3 imply that OP is performing better than WH, and it is thus clear that, while WH gave comparable results for some of the smaller problems, it was clearly inferior to the OP heuristic for the larger problems when restricted in this way. The tabu search could, of course, be prolonged in order to produce better results, but in order to compare favourably with OP it needed unreasonable amounts of computer time. For example, over 2 hours computing time was needed for WH to generate a solution to any of the 75/20 problems of a comparable standard to the corresponding OP solution—more than ten times that needed by OP.

TABLE 1. *Comparison of the Widmer/Hertz and Osman/Potts methods*

| Problem group $n/m$ | MPD | NI |
|---|---|---|
| 20/5 | 2.10 | 6 |
| 20/10 | −0.23 | 2 |
| 20/15 | 0.11 | 4 |
| 30/10 | 1.43 | 5 |
| 30/15 | 2.13 | 5 |
| 50/10 | 2.87 | 6 |
| 75/20 | 3.73 | 6 |

It is worth pointing out that in a real-world application, such a time-limit on the amount of computation might well have to be employed. Real problems are dynamic, so that to use an algorithm based on a static formulation is an approximation anyway. To have an algorithm which takes a long time may be quite unacceptable, since by the time a 'solution' has been obtained, the problem may have changed (for example, more jobs may have arrived), and the quality of the static solution might be of rather academic interest.

These results are disappointing, as in other applications[15] TS has nearly always outperformed SA. The purpose of the research reported here is to attempt to improve the efficiency of TS, using WH as a basis, in such a way that the time complexity of the procedure is the same as OP.

An obvious starting point in trying to improve the performance of TS is to use a better heuristic for the initialization, so that the algorithm spends less time in finding a local optimum, and more in searching from it. The method of Nawaz *et al.*[16] has been extensively tested, and is generally regarded as the best construction method available. This was therefore used in place of the one used in Widmer and Hertz.[12]

### *Taillard's method*

While this research was in progress, Taillard[13] published a paper detailing other implementations of tabu search for flowshop sequencing which improve on WH. As in the above, he also used the NEH algorithm of Nawaz *et al.*[16] in initializing the search. However, the key factor in improving the efficiency of TS was an ingenious algorithm which he credits to unpublished work by Mohr for performing an important part of the NEH algorithm more efficiently.

The NEH algorithm is a step-by-step procedure which first creates a sequence of 2 jobs, then $3, 4, \ldots, n$, by successively inserting one unsequenced job into the existing partial sequence. At its heart is the idea that the job chosen to be inserted should be the one which minimizes the increase in the objective function thus caused. If at stage $k$ we have a partial sequence $\{J_1, \ldots, J_k\}$, this implies $k$ calculations of the effect of inserting the unsequenced job $J_r$, say, between jobs $J_1$, $J_2$, $J_3$ and so on, or after $J_k$ ($J_1$ is regarded as fixed). Each requires $O(mk)$ computations, so the overall time complexity is $O(mk^2)$. Mohr's procedure calculates all these effects in a single pass which has a time complexity of only $O(mk)$.

In the TS context, Taillard carried out an iteration of tabu search by computing the effect of removing each job $J_i$ in turn, and finding the best non-tabu position for re-inserting it by Mohr's algorithm. This clearly has time complexity $O(mn^2)$. Thus, he was able to implement the basic steps of TS more efficiently, and to examine various strategies for its overall operation. As in the experiments reported here, he found that the WH implementation was inferior, but came to no firm conclusions on the best method of implementing TS. One reasonable method was to iterate $K$ times, which implies a time complexity $O(Kmn^2)$; however, as it frequently seems necessary for $K$ to be $O(n)$, this should perhaps more accurately be described as $O(mn^3)$. In the light of this, Taillard recommends implementing TS in parallel in order to reduce the computation times.

However, parallelization is still rather a specialized field, and it seemed worthwhile to continue with the attempt to produce a TS heuristic for a sequential computer which performs as well as SA.

## EXPLORATION AND EXPLOITATION

Having decided to use Mohr's implementation of the NEH heuristic to provide the initial solution, the next consideration was how to divide the computational effort between *exploration* on the one hand and *exploitation* on the other. By exploration we mean the process of examining new candidate solutions by a tabu-restricted neighbourhood search, while exploitation takes place when there is a move to the best of these candidates to start a new search phase. By fixing a maximum number of function evaluations, as in the case of the Osman/Potts SA heuristic, which is proportional to $\log(n + m)$, searching a neighbourhood of size $O(n^2)$ is clearly favouring exploration over exploitation, as few iterations can be carried out. Further, if we search an entire neighbourhood in order to make a single move, it is likely that many of the moves considered on the next search phase will be very similar if not identical to the ones considered on the previous search, so that much of the exploration effort is wasteful duplication. It would seem sensible, therefore, to choose a smaller neighbourhood and allow more iterations.

If we limit the size of the neighbourhood, we first of all have the problem of defining it in a satisfactory way. Thus we could examine the effect of removing and re-inserting each of jobs $\{J_1, \ldots, J_p\}$ where $p < n$, move to the best solution thus found, then try moving jobs $\{J_{p+1}, \ldots, J_{2p}\}$ and so on. When we reach job $J_n$ the search restarts from $J_1$. Of course, there is nothing special about this 'natural' job ordering, and this procedure was generalized by randomizing the order in which the $p$-job subsets were chosen, generating a random permutation of the numbers $\{1, \ldots, n\}$ whenever the $n'$th job is reached, where $n'$ is an integer multiple of $n$. This permutation then defines the order in which the next $n$ job moves will be explored. Randomization in the context of TS is an approach which previous implementations have been reluctant to use, but it seemed reasonable to try it.

It was thought possible that the best value of $p$ might vary with $n$, so in another series of experiments, in which a shift neighbourhood was used with both a natural and a random order of jobs, the parameter that was varied was $\phi = p/n$. The problems were as before, as was the computer time allowed for each problem. The results, in Table 2, show the mean percentage difference (MPD) expressed relative to the minimum makespan known for each problem, and averaged over six problems in each group.

As can be seen, having a small neighbourhood (and hence a large number of iterations) appears to be a much better use of resources. An analysis of variance showed a very significant difference between the levels of $\phi$. This was confirmed by the non-parametric Friedman test. Perhaps surprisingly, in view of the implicit consensus against randomization in TS, a random search performed better overall than an ordered one, although for the best (i.e. low) values of $\phi$ the differences were of little significance.

On closer examination, it was the difference between high and low $\phi$ values that was the major factor; there is a reasonable spread of low values of $\phi$ which gave broadly comparable results. However, the evidence seemed also to indicate, not that the best value of $p$ varies with $n$, but rather that a fixed value of $p$, at somewhere between 5 and 10, would give the best results. On reflection, this is not too surprising, since by using Mohr's algorithm we are actually searching a neighbourhod

TABLE 2. *Performance of revised TS heuristic*

| Problem group $n/m$ | MPD $\phi = 0.1$ | $\phi = 0.2$ | $\phi = 0.3$ | $\phi = 0.5$ | $\phi = 1.0$ |
|---|---|---|---|---|---|
| | | Natural Order | | | |
| 20/5 | 0.09 | 0.31 | 0.32 | 0.88 | 0.46 |
| 20/10 | 1.10 | 0.20 | 0.09 | 1.79 | 2.15 |
| 20/15 | 1.60 | 0.55 | 0.55 | 1.07 | 2.31 |
| 30/10 | 0.41 | 0.68 | 0.75 | 1.89 | 2.15 |
| 30/15 | 0.95 | 0.38 | 1.09 | 1.63 | 2.66 |
| 50/10 | 0.32 | 0.64 | 0.83 | 0.92 | 1.40 |
| 75/20 | 0.62 | 1.23 | 1.30 | 1.64 | 2.70 |
| | | Random Order | | | |
| 20/5 | 0.22 | 0.28 | 0.17 | 0.35 | 0.26 |
| 20/10 | 0.63 | 0.19 | 0.86 | 0.63 | 2.16 |
| 20/15 | 1.11 | 0.61 | 0.54 | 0.30 | 1.53 |
| 30/10 | 0.39 | 0.35 | 0.71 | 0.93 | 0.88 |
| 30/15 | 0.53 | 0.03 | 0.40 | 0.74 | 1.43 |
| 50/10 | 0.69 | 0.73 | 0.69 | 0.97 | 1.03 |
| 75/20 | 0.73 | 1.27 | 1.39 | 1.69 | 1.80 |

of size $O(pn)$ at each iteration. If $p$ did vary with $n$, the revised TS method would be reducing the size of the neighbourhood but not essentially affecting the complexity. These experiments appear to indicate, however, that it is possible to obtain improved results while reducing the computation in the exploration phase by an order of magnitude. If so, this would be a substantial advance in the implementation of tabu search, and it clearly requires further testing on different problems that were not used in the earlier experiments.

## BENCHMARK COMPARISONS

Recently, Taillard[14] has produced a set of test problems for machine sequencing which he found to be particularly difficult, in the sense that the best solutions he could find by applying his tabu search heuristic with very many iterations were still substantially inferior to their lower bounds. The problems range from small instances with 20 jobs and five machines to large ones with 500 jobs and 20 machines. There were ten instances for each size of problem, and all the processing times were generated randomly from a $U(1, 100)$ distribution. There were 12 groups of problems.

The revised TS implementation (RevTS) described above with a randomized order of exploration and $p = 6$ was used to solve these instances, and the results compared with those obtained from the Osman/Potts heuristic (OP). Both methods were again restricted to run in the same computer time—that needed for the maximum number of function evaluations prescribed by OP. The details, in terms of the MPD relative to Taillard's best-known makespan values, are shown in Table 3; also shown are the results of using 'brute-force' TS (BfTS) with a random search and $p = n$. (This choice of parameters is roughly equivalent to one of Taillard's implementations, and heavily favours exploration over exploitation.) Also shown are the number of instances (NI) out of 10 for which each method was the best—sometimes there were ties, so the sum for each row may exceed 10.

It is clear from these figures that the revised TS implementation is superior to the Osman/Potts SA procedure, particularly for the larger problem instances. It is worth noting that, in the case of the larger instances, the revised TS heuristic several times discovered makespans which improved on Taillard's best known solutions (hence the negative MPDs in the last row). Thus, these experiments would also appear to confirm our conjecture that the more traditional 'brute-force' TS approach in which *complete* neighbourhoods are explored is not an efficient use of computing resources. Such an approach may be adequate for relatively small problems, but it would appear that overall it is better to make fairly limited explorations in the search space and exploit the best of these, rather than pursue a lengthy search before each move is made.

It is, of course, possible that the 'brute-force' TS method would eventually prove superior to

TABLE 3. *Performance on Taillard's benchmarks*

| Problem group $n/m$ | MPD RevTS | MPD OP | MPD BfTS | NI RevTS | NI OP | NI BfTS |
|---|---|---|---|---|---|---|
| 20/5 | 1.01 | 1.27 | 1.15 | 6 | 5 | 7 |
| 20/10 | 1.07 | 1.71 | 2.02 | 7 | 3 | 1 |
| 20/20 | 0.34 | 0.86 | 1.53 | 7 | 3 | 0 |
| 50/5 | 0.18 | 0.78 | 0.19 | 6 | 3 | 7 |
| 50/10 | 0.86 | 1.98 | 1.52 | 7 | 1 | 2 |
| 50/20 | 1.55 | 2.86 | 2.29 | 9 | 0 | 1 |
| 100/5 | 0.19 | 0.56 | 0.21 | 9 | 0 | 8 |
| 100/10 | 0.69 | 1.33 | 0.97 | 7 | 1 | 3 |
| 100/20 | 1.07 | 2.32 | 1.95 | 9 | 0 | 1 |
| 200/10 | 0.23 | 0.83 | 0.30 | 8 | 0 | 6 |
| 200/20 | 0.08 | 1.74 | 0.86 | 10 | 0 | 0 |
| 500/20 | −0.79 | 1.42 | −0.45 | 10 | 0 | 0 |
| Average | 0.54 | 1.42 | 1.04 | 7.91 | 1.33 | 3.00 |

the revised implementation, and it might be conjectured that, given enough time, the graphs of best-solution-so-far against time would 'cross over'. In order to test this, each was allowed ten times the amount of computer time prescribed by Osman and Potts and run on some of the larger problems. In each case, the revised procedure was still superior at all points in time, and there was no sign that the graphs would cross over. A typical graph (for a 500-job, 20-machine problem) is shown in Figure 1.

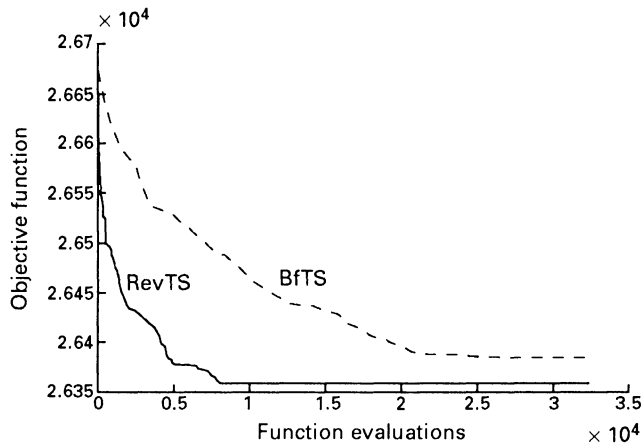


FIG 1. *Convergence of RevTS versus BfTS.*

## CONCLUSIONS

Tabu search is a powerful method for obtaining near-optimal solutions to combinatorial problems; its appeal is largely intuitive, and little theoretical analysis is available, so finding ways of making it even more effective is necessarily an experimental procedure. In the experiments reported here, it has been shown that it is important to consider the definition of a neighbourhood carefully, and to balance the demands of exploration and exploitation. Previous implementations of TS have often been characterized by a 'brute-force' approach to searching which means either that exploitation must be curtailed or that obtaining a good solution to a problem will take an unnecessarily large amount of computing effort.

This work has also demonstrated that tabu search, when implemented efficiently, is superior to one of the best simulated annealing procedures, in that, on average, better solutions can be produced even when the computing effort is restricted to a value which has been found to be effective for SA. In addition, it has found new upper bounds for a number of benchmark problems.

It would be interesting to investigate other NP-hard problems, in order to determine if experiments similar to those performed here for the $n/m/P/C_{\max}$ problem reveal a similar relationship

between exploration and exploitation in tabu search. These experiments have shown that for the problem investigated here, the amount of computation in the exploration phase can be reduced by an order of magnitude relative to the neighbourhood size. If this behaviour is repeated for other problems, a significant and general improvement may be possible in the implementation of tabu search.

Finally, it should be observed that there is nothing in this implementation which in any way prevents the sort of parallelization suggested in Taillard[13]. For large problems even this implementation of tabu search will still take a fairly long time, and there are obvious advantages in increasing efficiency by means of a parallel implementation.

## REFERENCES


1. M. R. GAREY and D. S. JOHNSON (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.
2. S. LIN (1965) Computer solutions of the traveling salesman problem. *Bell Syst. Tech. J.* **44**, 2245–2269.
3. K. R. BAKER (1974) *Introduction to Sequencing and Scheduling*. John Wiley, New York.
4. I. H. OSMAN and C. N. POTTS (1989) Simulated annealing for permutation flow-shop scheduling. *Omega* **17**, 551–557.
5. A. H. G. RINNOOY KAN (1976) *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague.
6. S. KIRKPATRICK, C. D. JELATT and M. P. VECCHI (1983) Optimisation by simulated annealing. *Science* **220**, 671–679.
7. F. A. OGBU and D. K. SMITH (1990) The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem. *Comps Opns Res.* **17**, 243–253.
8. M. LUNDY and A. MEES (1986) Convergence of an annealing algorithm. *Math. Prog.* **34**, 111–124.
9. F. A. OGBU and D. K. SMITH (1991) Simulated annealing for the permutation flow-shop problem. *Omega* **19**, 64–67.
10. F. GLOVER (1986) Future paths for integer programming and links to artificial intelligence. *Comps Opns Res.* **13**, 533–549.
11. F. GLOVER (1989) Tabu search—Part I. *ORSA J. Comput.* **1**, 190–206.
12. M. WIDMER and A. HERTZ (1989) A new heuristic method for the flow shop sequencing problem. *Eur. J. Opl Res.* **41**, 186–193.
13. E. TAILLARD (1990) Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Opl Res.* **47**, 65–74.
14. E. TAILLARD (1992) Benchmarks for basic scheduling problems. *Eur. J. Opl Res.* (to appear)
15. D. DE WERRA and A. HERTZ (1989) Tabu search techniques. *OR Spektrum* **11**, 131–141.
16. M. NAWAZ, E. E. EMSCORE JR. and I. HAM (1983) A heuristic algorithm for the $m$-machine, $n$-job flow-shop sequencing problem. *Omega* **11**, 91–95.