Published in *European Journal of Operational Research*, vol. 47, no. 1, pp. 65-74, which should be cited to refer to this work.

DOI: 10.1016/0377-2217(90)90090-X

Some efficient heuristic methods for the flow shop sequencing problem

E. Taillard

Abstract

We compare in this paper the best heuristic methods known up to now to solve the flow shop sequencing problem and we improve the complexity of the best one. Next, we apply to this problem taboo search, a new technique to solve combinatorial optimization problems, and we report computational experiments. Finally a parallel taboo search algorithm is presented and experimental results show that this heuristics allows very good speed-up.

Keywords: Flow shop, taboo search techniques, parallel algorithm, combinatorial optimization.

1. Introduction

First, we try to answer the question: "What is the problem ?". Although many researchers have been working on the flow shop sequencing problem for many years, we found nowhere any results about the distribution of the objective function and the distribution of the optima of this function. In effect, such an approach gives an intuitive idea about the problem and is important to allow the reader to judge the quality of heuristic methods used for this problem.

Then we compare the classical heuristics and improve the complexity of the best one. But this one does not give very good solutions on average (less than one or two percent above the optimal solution). So we propose a heuristics improving the mean quality of solutions when running longer, based on taboo search technique. As this technique has been recently developed, we do not only give the best implementation we found, but some variants of this method too.

Finally, we propose two parallel versions of taboo search, in order to reduce the unavoidable expansive calculation times needed by this method.

2. The flow shop sequencing problem

The flow shop sequencing problem is a production planning problem: n jobs (items, tasks...) have to be processed in the same sequence on m machines; the processing time of job *i* on machine *j* is given by t_{ij} (i = 1...n, j = 1...m). These times are fixed, non negative and some of them may be zero if some job is not processed on a machine.

The problem consists of minimizing the time between the beginning of the execution of the first job on the first machine and the completion of the execution of the last job on the last machine; this time is called *makespan*. For this problem the following assumptions are made:

- Every job has to be processed at most once on machine 1, 2...m (in this order).
- Every machine processes only one job at a time.
- Every job is processed at most on one machine at a time.
- The operations are not preemptable.

- The set-up times of the operations are included in the processing time and do not depend on the sequence.
- The operating sequences of the jobs are the same on every machine and the common sequence has to be determined.

This problem is NP-hard and can be solved exactly only for small sizes [2]. It consists of finding a sequence σ that minimizes the makespan $M(\sigma)$. So the number of possible schedules is n!

Some observations on small problems

First we give in fig. 1 the distribution of all the possible makespans obtained by complete enumeration of 500 problems with 9 jobs and 10 machines. This distribution is given relatively to the optimal solution. The processing times were randomly uniformly generated (integers between 1 and 100). We choose this problem size because it is possible to calculate $M(\sigma)$ for every solution σ in a reasonable calculation time.

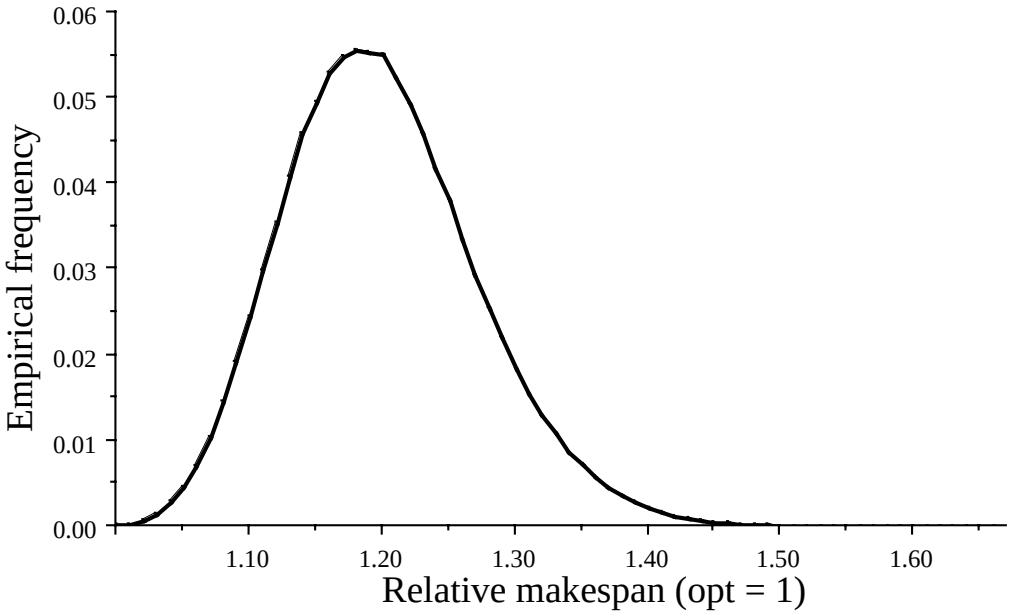

Fig. 1 Empirical distribution of the makespans

We can observe that the distribution is not symmetrical. Less than 0.02% of the $M(\sigma)$ are between $M(\sigma_{opt})$ and $1.01 \cdot M(\sigma_{opt})$ (where σ_{opt} is an optimal schedule). So finding a solution at 1% above the optimal one is generally very hard, but a random solution is in mean only at 20% above the optimum. Then we give in fig. 2 the distribution of the optimal makespans $M(\sigma_{opt})$ for these problems.

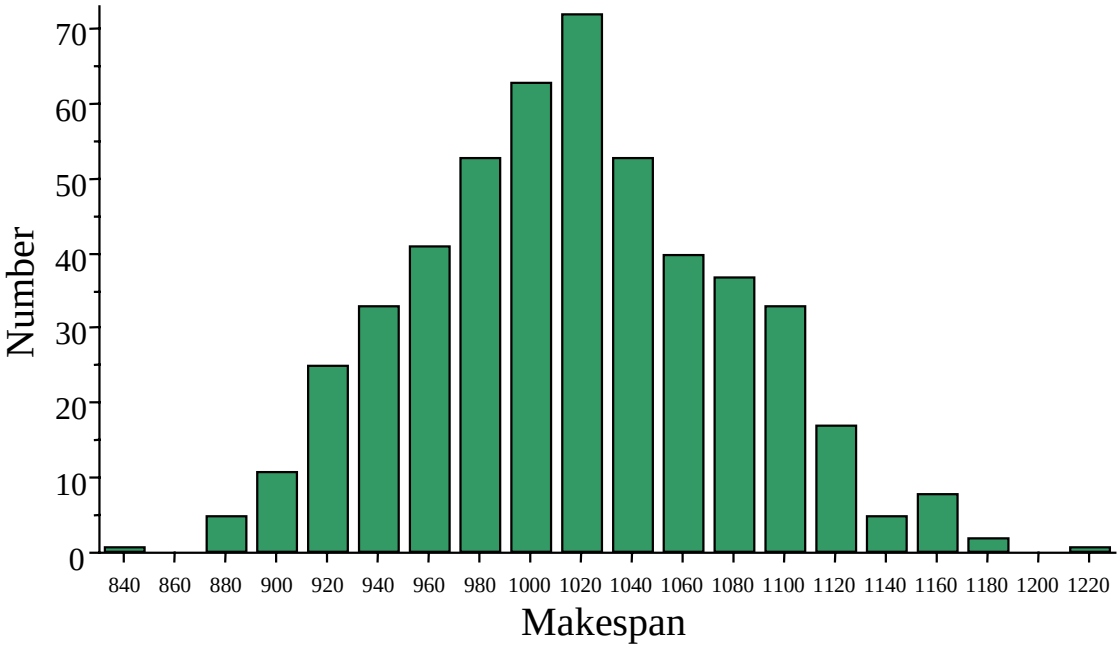

Fig. 2. Distribution of the optimal makespans

This distribution seems to be almost symmetrical and its range (for these 500 problems) is contained in an interval of 20% around the mean. The mean value of the makespan is 1016.1 and the value of the standard deviation is 62.1. A χ^2 test does neither confirm nor refute that this distribution is gaussian. So, speaking of the mean makespan given by a heuristics seems to be a meaningful measure.

3. Comparison of classical heuristics

Many heuristics have been proposed to solve the flow shop problem; we compare in table 1 the quality of the solutions and the complexity of some of them. One can find the descriptions of these methods in [1] for Gupta, Johnson, Palmer and CDS (algorithm of Campbell, Dudek and Smith), in [3] for RA (rapid access procedure) and in [8] for NEH (algorithm of Nawaz, Enscore and Ham).

	Complexity	Quality					
Problems	-	500(*)	100	100	100	50	50
Jobs	n	9	10	20	20	40	50
Machines	m	10	10	10	20	10	10
Gupta	nlog(n) + nm	13.4	12.8	19.6	18.8	18.9	17.1
Johnson	nlog(n) + nm	10.9	11.8	16.7	16.8	17.3	16.3
RA	nlog(n) + nm	8.5	9.1	12.5	13.4	13.5	11.2
Palmer	nlog(n) + nm	8.3	9.0	13.3	12.5	10.9	10.7
CDS	nm^2 + mnlog(n)	4.5	5.2	9.7	8.6	9.9	9.3
NEH	n^2m	2.1	2.2	3.9	3.8	2.6	2.1

Table 1: Comparison of the classical heuristics

The complexity includes the computation of the makespan. The quality of the solutions is given in percent above the mean of the optima (*) or of the makespan obtained after 1000 iterations of taboo heuristics.

NEH appears to be the best polynomial heuristics in practice. The heuristics RA or Palmer may also be useful when short computation times are required. Other results about these heuristics are discussed in [9]. Note that the new method described below [7] permitted us to reduce the complexity of the NEH algorithm from $O(n^3m)$ to $O(n^2m)$.

Naturally, descent algorithms may be applied to the solutions given by these heuristics, but one cannot give the complexity anymore and the improvements are small: for NEH, the mean improvement of solutions is less than 1% and the calculation time becomes as important as NEH's for the other heuristics.

4. An improvement of NEH heuristics

We will first recall the NEH algorithm:

1) *Order the n jobs by decreasing sums of processing times on the machines.*

2) *Take the first two jobs and schedule them in order to minimize the partial makespan as if there were only these two jobs.*

3) *For k=3 to n do*

4) *Insert the* k*th job at the place, among the k possible ones, which minimizes the partial makespan.*

The complexity of step ***1)*** is O(nlog(n)); that of step ***2)*** is O(m). In order to calculate one partial makespan in step ***4)*** one needs O(km) operations. However, it is possible to calculate the k makespans of this step in O(km):

Algorithm (to find M_i, the makespan after insertion of job k at the *i*th place)

1) Compute the earliest completion time e_{ij} of the *i*th job on the *j*th machine; the starting time of the first job on the first machine is 0 (see fig. 3a).
   $e_{0j} = 0,\ e_{i0} = 0 \qquad e_{ij} = \max(e_{i,j-1}, e_{i-1,j}) + t_{ij} \qquad (i = 1...k-1)\ (j = 1...m)$
2) Compute the tail q_{ij}, i.e.the duration between the starting time of the *i*th job on the *j*th machine and the end of the operations (fig. 3b)
   $q_{kj} = 0,\ q_{i,m+1} = 0 \qquad q_{ij} = \max(q_{i,j+1}, q_{i+1,j}) + t_{ij} \qquad (i = k-1...1)\ (j = m...1)$
3) Compute the earliest relative completion time f_{ij} on the *j*th machine of job k inserted at the *i*th position (fig. 3c)
   $f_{i0} = 0 \qquad f_{ij} = \max(f_{i,j-1}, e_{i-1,j}) + t_{kj} \qquad (i = 1...k)\ (j = 1...m)$
4) The value of the partial makespan M_i when adding job k at the *i*th position is:
   $M_i = \max_j(f_{ij} + q_{ij}) \quad (i = 1...k)\ (j = 1...m)$

All these steps can be executed in time O(km). Consequently, step ***4)*** of NEH algorithm has a complexity of O(km). We conclude that NEH algorithm runs in time $O(n^2m)$.

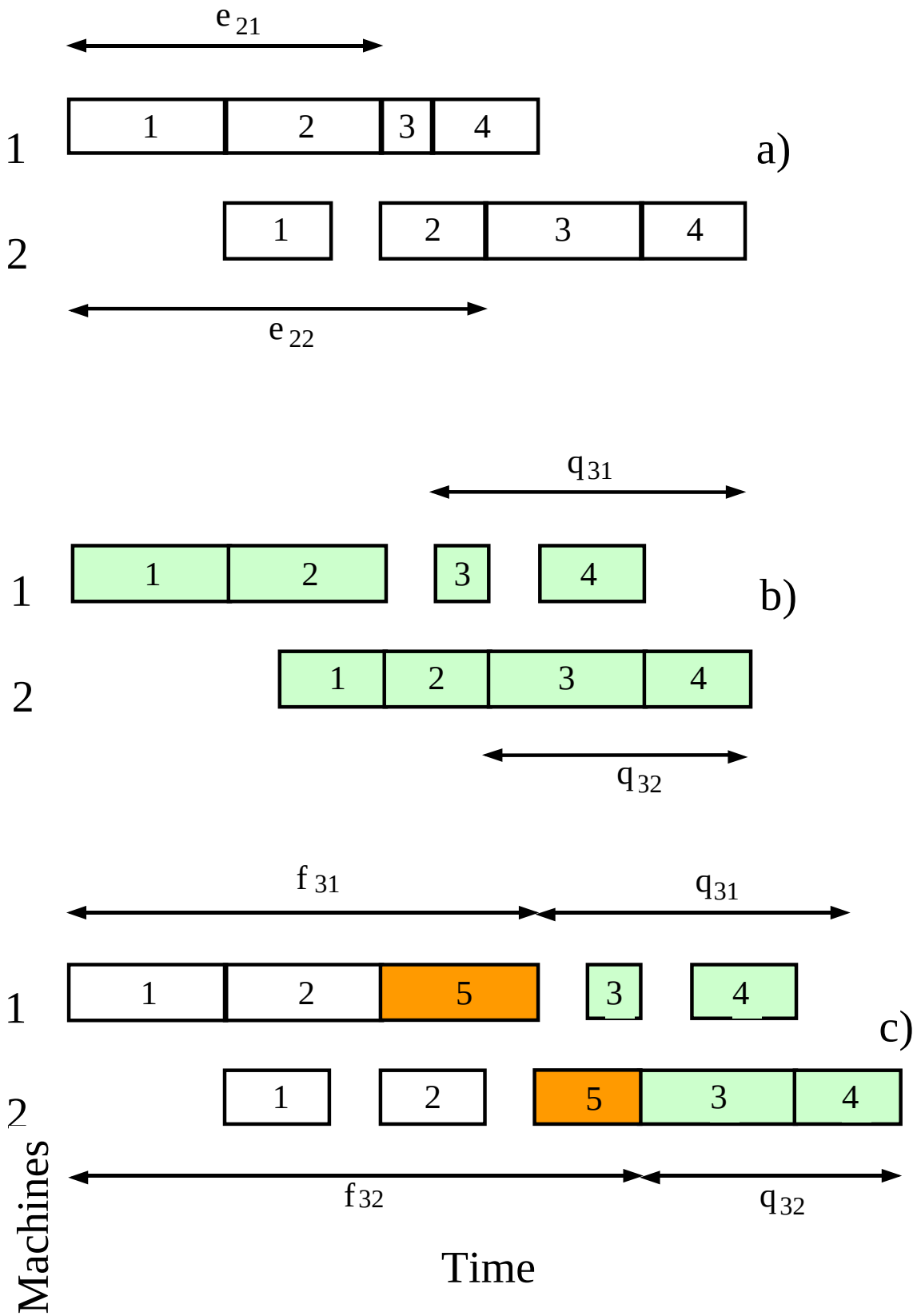
e 21
1
2
3
4
a)
1
2
3
4
e 22
q31
b)
1
2
3
4
1
2
3
4
q32
f 31
q31
c)
1
2
5
3
4
1
2
5
3
4
f32
q32
Machines
Time

Fig. 3 Illustration of the algorithm: insertion of job 5 at the third place.

5. Taboo search techniques

Let us briefly describe taboo search techniques, before presenting how they can be applied to the flow shop problem; this technique is exposed in [4]. An application to the flow shop problem is proposed in [9].

Taboo search may be useful to find a good, or possibly optimal solution of problems which are of the type:

$$\text{Min } \mathbf{c(x)}$$
$$\text{s.t. } \mathbf{x} \in \mathbf{X}$$

Where **c(x)** is any function of a discrete variable **x**, and **X** is the set of feasible solutions. A step of taboo search starts with the current feasible solution $\mathbf{x} \in \mathbf{X}$ to which is applied a function $\mathbf{m} \in \mathbf{M(x)}$ that transforms **x** into **x',** a new feasible solution (**x'** = **m(x)**). This transformation is called a *move*, and {**x'**: **x'** = **m(x)**; **x, x'** ∈ **X**; **m** ∈ **M(x)**} is called the *neighbourhood* of **x.**

In order to avoid as much as possible cycling, an element **t** is associated with **m** and **x**; this element defines a set of moves that are taboo (forbidden) now; it is stored in a set **T** called *taboo list*. In particular **t** forbids to apply **m'** to **x'** which would transform **x'** back to **x**; but **t** may forbid other moves too. The elements of **T** define all taboo moves that cannot be applied to the current solution; in practice, the size of **T** cannot increase indefinitely and has to be bounded by a parameter **s**, called *taboo list size*. If |**T**| = **s**, before adding **t** to **T**, one must remove an element, generally the oldest one.

An application of taboo search is characterized by:

1) The set **M(x)** of moves applicable to a feasible solution **x** (**neighbourhood**).
2) The type of the elements of the set **T** which define the taboo moves (**taboo list**).
3) The size **s** of the set **T** (**taboo list size**).
4) A stopping condition.

The generic procedure of taboo search techniques is:

0) Start with any feasible solution $\mathbf{x_0}$, an empty taboo list **T**. Let $\mathbf{x^*} = \mathbf{x_0}$, $\mathbf{c^*} = \mathbf{c(x_0)}$ and k = 0. ($\mathbf{x^*}$ is the best solution found up to now and $\mathbf{c^*}$ the value of the objective function for this solution)

1) In $\mathbf{M(x_k)}$ choose **m**, a move transforming $\mathbf{x_k}$ that minimizes $\mathbf{c(m(x_k))}$ and that is not forbidden by the elements of **T**. The move can be chosen by complete or partial examination of $\mathbf{M(x_k)}$. Let $\mathbf{x_{k+1} = m(x_k)}$.

2) If $\mathbf{c(x_{k+1})} < \mathbf{c^*}$, let $\mathbf{c^*} = \mathbf{c(x_{k+1})}$ and $\mathbf{x^*} = \mathbf{x_{k+1}}$.

3) If $|\mathbf{T}| = \mathbf{s}$ remove the oldest element of **T**; add the element **t** defined by **m** and $\mathbf{x_{k+1}}$. Increment k by 1.

4) Go back to **1)** if the stopping condition (optimum reached, k larger than a fixed limit...) is not satisfied.

Applications of taboo search techniques

The objective function of our flow shop problem is the makespan and the set of feasible solutions is any permutation σ of {1...n}:

$$\text{Min } M(\sigma)$$
$$\text{s.t. } \sigma\text{: permutations of } 1...n$$

For this problem, the neighbourhood may be defined in several ways:

1) Exchange two adjacent jobs.

A move **m** is entirely defined by i. The size of the neighbourhood is $|\mathbf{M(\sigma)}| = n-1$. Our experiments show that these moves are bad, both for quality of schedules and global calculation time.

2) Exchange the jobs placed at the *i*th and the *k*th position.

A move **m** is entirely defined by i and k. The size of the neighbourhood is **Erreur!** . The evaluation of all the makespans σ**'**, neighbour of σ , can be executed in time $O(n^3m)$. [9] proposes this kind of neighbourhood. Our experiments show that such a neighbourhood is not better than the next one to find good schedules with taboo search techniques; furthermore, the complexity of each single step is higher.

3) Remove the job placed at the *i*th position and put it at the *k*th position.

A move **m** is entirely defined by i and k. The size $|\mathbf{M}(\sigma)|$ of this neighbourhood is $(n-1)^2$. The evaluation of all the makespans can be executed in time $O(n^2m)$, using the insertion algorithm described in the NEH heuristics. We choose this type of neighbourhood because of the efficiency of the moves, both for quality and computation times.

Next we have to define how to examine the neighbourhood before choosing a move leading to the next step:

a) Examine the neighbours and take the first which improves the current solution. If there is no move that improves the solution (or if all improving moves are taboo) then one has to examine the whole neighbourhood. For this method, the mean calculation time of a step is less than the one needed for method c). But this time is not constant, and the steps are not as good. [9] proposes this examination.

b) Examine a fixed number of moves that are not taboo, randomly generated. This method is useful for problems for which the size of the neighbourhood is very large: but our experiments have shown that it does not suit for middle-size flow shop problems.

c) Examine the entire neighbourhood and take the best move that is not taboo. This method needs more (but constant) calculation time for each step than partial enumeration, but the moves are better. If one wants to examine the neighbourhood in parallel, this method allows to balance very well the work between the processors.

The taboo list may also be of several types:

i) Prevent a job from returning to a fixed place before one has made **s** steps (**s**: length of the taboo list). In this case, **s** is a sensitive parameter; [9] proposes to fix it at the value of 7.

ii) Prevent the new makespan from coming back to a makespan that was already obtained in the **s** previous steps. If **s** varies in an acyclic way (by example if **s** simply grows) then cycling is well prevented.

These two taboo lists are good but the last one avoids the use of another parameter called *aspiration level* (i.e. a taboo move is allowed if it improves the objective function of more than a value, the aspiration level, which has to be defined and depends on **m** and σ). We have chosen the latter type of taboo list.

Performances of taboo search

In order to evaluate the performances of taboo search, we have first randomly generated 200 flow shop problems of 9 jobs and 10 machines, for which the optimal makespan was known. Then we have solved these problems 100 times with taboo search, starting from various initial solutions. We have done the same with 8 problems given by [2] of various sizes (11 jobs x 5 machines, 13x4, 12x5, 14x4, 10x6, 8x9, 7x7 and 8x8). We can make the following remarks:

1) There are a few problems that are very easy to solve (always less than 15 iterations) and a few ones that are much more difficult (sometimes more than 800 iterations).

2) For a fixed problem, the number of iterations (steps or moves) can be very variable. (From 10 to 800 iterations, depending on the starting point)

In fig. 4 we give the empirical distribution of the CPU time required by a resolution on a Vax 8600. This distribution is tabulated for two types of neighbourhood examination.

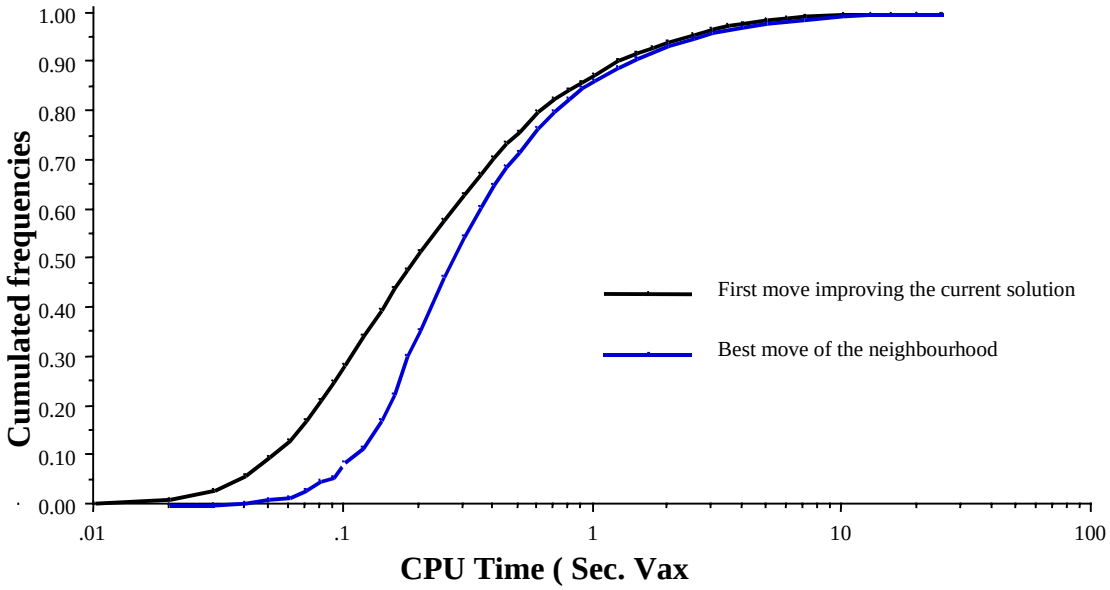

Fig. 4: CPU time to find the optimal makespan (9 Jobs, 10 Machines)

1) Best move: all the neighbours are evaluated and the best becomes the next current solution.
2) First move improving the current solution: the examination of the neighbours is stopped when a non taboo move leads to a better solution than the current one.

The second rule of examination is slightly better than the first one. The mean resolution time is 546 ms. for the first one versus 675 ms. for the second one but the mean number of iterations is higher: 29 versus 24. However, these measures may be meaningless, because of the extent between the extremities of the distribution's curve.

However, for practical problems, one cannot know, and even characterize the optimal makespan; so determining a good stopping condition of taboo search is not trivial. We have tried three stopping conditions:

1) Stop if the number of iterations is greater than k, an a priori fixed constant. The taboo search completes in time $O(kn^2m)$ with this stopping condition. In fig. 5 and 6, the evolution of the mean makespan is plotted as a function of the number of steps of taboo search. There were 100 problems of the following size: 10 jobs x 10 machines, 20x5, 20x10 and 20x20 , and 50 problems of 30x10, 40x10 and 50x10. All these problems were randomly generated, the processing times of the jobs on the machines being uniformly distributed integers between 1 and 100. The taboo list size was growing from 7 (for the first iterations) to 100 (for the last ones).

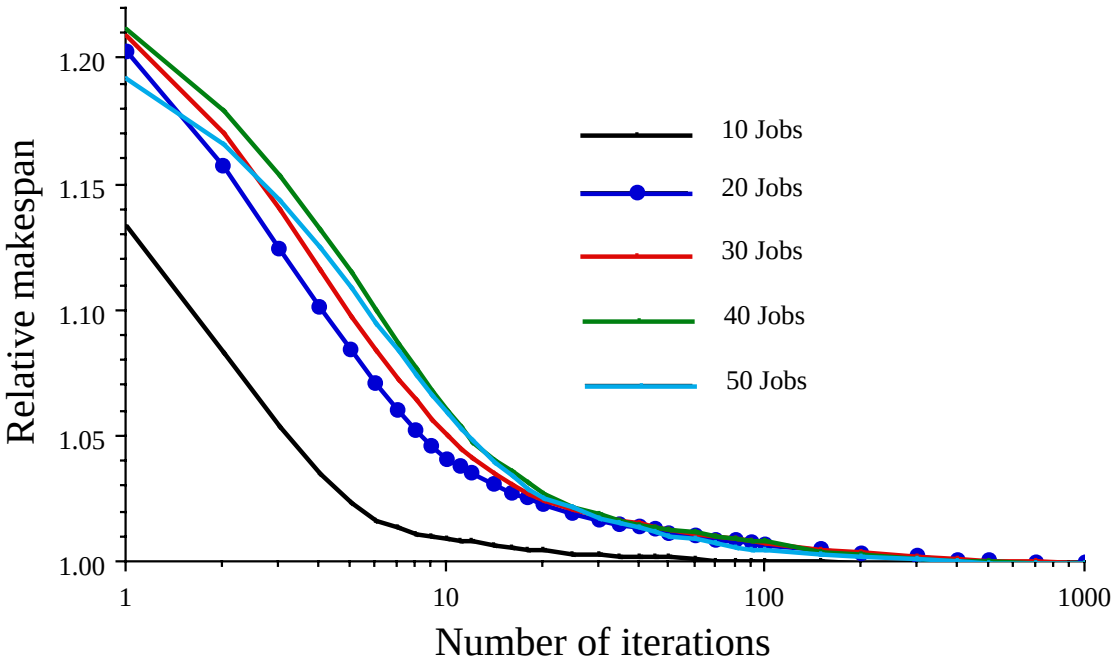

Fig. 5 Evolution of the makespan: 10 Machines

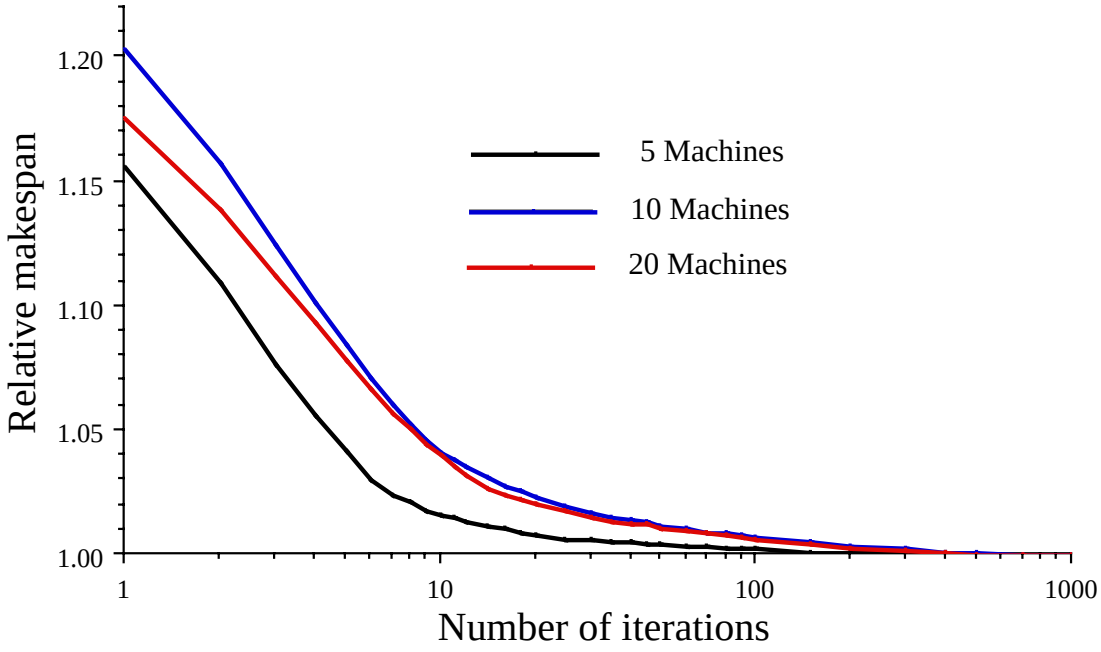

Fig. 6 Evolution of the makespan: 20 jobs

The curve of the evolution of the makespan may be interpreted as follows: for the first iterations (< n), taboo search is the same as an improving heuristics which goes into a local minimum. Then taboo search goes from a local minimum to another one and the improvements become less and less frequent (in an over exponential way).

2) Stop if the number of iterations without improving the best solution is greater than a constant a priori fixed. We compare in fig. 7 the evolution of the makespan as a function of the total number of iterations (stopping condition 1) and this evolution, function of the number of iterations without improving the best solution. We can see that this second stopping condition provides a more regular progression of the mean makespan. In fact this curve is more or less the same than the first one but without the preliminary way down to a local minimum; it is important to mention that both evolutions of makespan, function of *CPU time* provide the *same* curve whichever stopping condition is chosen. The duration of taboo search, for this second stopping condition depends on the problem and on the initial solution.

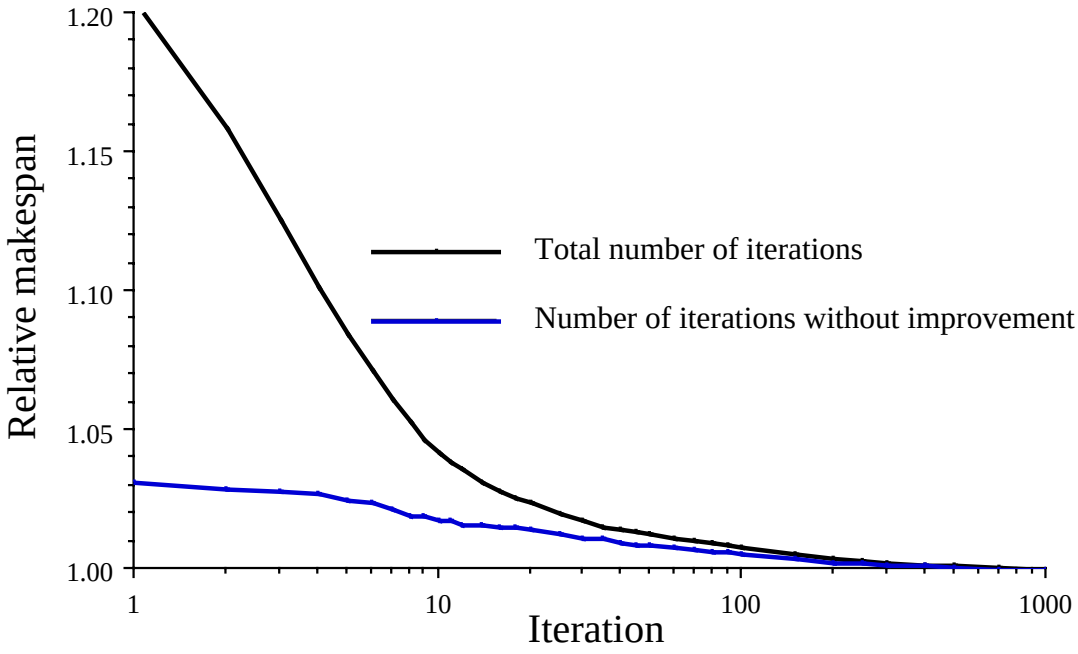

Fig. 7 Stopping condition: evolution of the makespan (20 Jobs, 10 Machines)

3) We have seen that the number of iterations needed to find the optimal solution of the flow shop problem (and other ones too, see [6]) depends strongly on the initial solution; what is more, one continues working even if one has the optimal solution, because one cannot characterize it. In order to reduce useless work, the following algorithm may provide good results: Let p be a fixed

number of processes; each of them executes independently a taboo search with a different initial solution. After a while (which has to be defined), the processes have to be stopped and their respective best solutions compared. If two or more of them are the best of all, then the algorithm ends; otherwise, the processes continue their taboo searches from where they are and so on. Some simulations (200 problems, 10 jobs x 10 machines and p = 4 processes) of this algorithm on a sequential machine show us that the mean of the total CPU time is more or less the same as with the previous algorithm, with stopping condition 2). The longer the time between comparisons is, the better the solutions are. For this algorithm, the solutions (schedules) may not be the same, even if the makespans are equal. This algorithm may not end (because of cycling), consequently another stopping condition has to be added, for example: stop if the number of comparisons without improving the best makespan is greater than a constant.

6. Parallelization of taboo search

The third stopping condition leads to a trivial parallelization; it is well adapted for small numbers of processes (typically from 3 to 6) but there is a limit to the speed-up due to the way down to the first local minimum; and at this limit, the algorithm becomes a simple improving heuristics which needs no taboo list at all!

Another approach of parallelization is the following: we have remarked that the time needed to evaluate the value of the makespans of the neighbourhood is almost the entire calculation time; in order to speed up the algorithm, one has to reduce this calculation time. This may be realized by parallelizing the search of the best neighbour: each processor inspects only a fraction of the neighbourhood. Then, the best allowed moves are compared and the best of all is chosen (see fig. 8). In order to do a step of taboo search, the algorithm then becomes (for processes without common memory):

(Assume that the master process has a current move and that each slave process has the same current solution, taboo list, and so on, but a different subset of neighbours to examine)

Master process:

M.1) Send to every slave process the current move.

M.2) Wait the best moves of each slave process and choose among them the best of all. Go back to **M.1)** unless a stopping condition is satisfied.

Slave process:

S.1) Wait for a move, given by the master process, perform it and update the taboo list.

S.2) Try all the moves among the partial neighbourhood; choose the best non taboo one and send it to the master process. Go back to **S.1)**

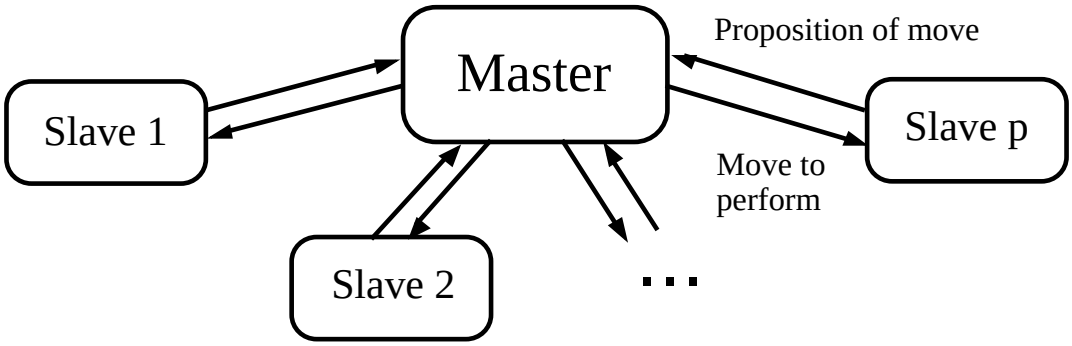

Fig. 8 Exchanges of information between the master and the slave processes.

The work done by each slave process is very long, if it is compared to the work of the master process, and, in practice, the master process and one slave process run on the same processor. With this technique, we could work 1.92 and 1.99 times faster with 2 processors than with only one, for problems of 10 machines and respectively 10 and 40 jobs.

For our experiments of parallelization, we used two 32-bits transputers (one T414 and one T800 which has the same integer calculation power). In very few words, a transputer is a microprocessor equipped with 4 bidirectional communication links, especially designed to create multiprocessor networks with distributed memory (MIMD machines); the synchronizations between processors are made by the messages. Readers are referred to [5] for more details see.

Fig. 9 represents the mean evolution of the makespan as a function of CPU time for 100 problems, 10 jobs and 10 machines.

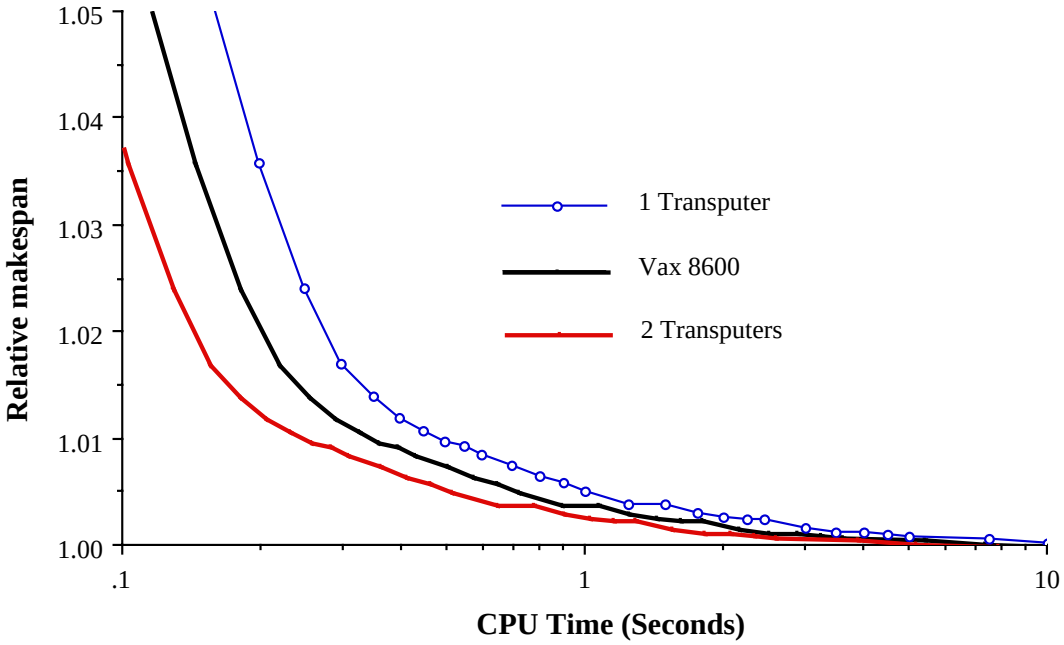

Fig. 9 Evolution of the makespan function of the CPU time of several machines

This evolution is given for:

1) The sequential algorithm, run on a T414 transputer.
2) The sequential algorithm, programmed in Pascal and run on a Vax 8600.
3) The parallel algorithm, run on two transputers (T414 and T800).

It is interesting to mention that two transputers have a calculation power comparable with a mini-computer, for well parallelizable algorithms.

7. Conclusions

In this paper, we have first presented the empirical profile of randomly generated instances of the flow shop problem; then, we have shown that NEH was the best heuristics among the classical ones. A new method permitted us to reduce its complexity to $O(n^2m)$.

The problem may be solved efficiently by a taboo search technique, and we can get better solutions than NEH. The optimality of the solutions cannot be proved, but we found

every time the optimal solution of the problems for which the exact solution was known, if we allowed sufficient CPU time.

Such a technique is very flexible and more general flow shop problems for which the already existing heuristics are not designed may be treated without great changes (set-up, processing times not fixed...). However, we have remarked that flow shop problems with ordering that can vary on each machine cannot be treated with exactly the same taboo search technique; some refinements have to be introduced.

Unfortunately, taboo search needs great calculation times; in order to reduce them, we have presented two methods of parallelization that can be simultaneously applied. These methods may lead to a parallelization with O(n) processors.

References

[1] K.R. BAKER. *Introduction to Sequencing & Scheduling*, Wiley, New-York, 1974.

[2] J. CARLIER. *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité*, Méthodologie & architecture des systèmes informatiques, institut de programmation, Université P. et M. Curie, Paris, 1984.

[3] D.G. DANNENBRING. *An Evaluation of Flow Shop Sequencing Heuristics*, Management Science, Vol. 23, no 11, pp. 1174-1182, July 1977.

[4] F. GLOVER. *Tabu search*. Preliminary draft, US West Chair in System Science, Center For Applied Artificial Intelligence, University of Colorado, 1987.

[5] INMOS LIMITED. I*MS T414 transputer, Engineering data*, INMOS Limited, 1000 Aztec West, Aldmondsbury, Bristol BS 124 SQ U.K.

[6] TH. MOHR. *Parallel Tabu Search Algorithms for the Graph Coloring Problem*, OR Working Paper no 88/11. Ecole Polytechnique Fédérale de Lausanne, département de mathématiques, Lausanne, Switzerland, 1988.

[7] TH. MOHR**.** Private communication.

[8] M. NAWAZ, E. ENSCORE JR., I. HAM. *A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem*, OMEGA, The int. J. of Mgmt Sci. Vol 11, no 1, pp 91-95. 1983.

[9] M. WIDMER, A. HERTZ. *A new heuristic method for the flow shop sequencing problem*. To appear in European Journal of Operational Research.