APLICACIÓN DEL ALGORITMO LOMPEN A LOS PROBLEMAS Fm | prmu | C_{max} Y Fm | block | C_{max}

- Different behaviour of a double branch-and-bound algorithm on Fm | prmu | C_{max} and Fm | block | C_{max} problems (*accepted for publication, C&OR*)
- Some comments to the editor and referee's remarks
- Annex 1: Solutions given by algorithms
- Annex 2: Paper's abstracts

por R. Companys, M. Mateo y A. Alemán
Laboratori d'Organització Industrial

Barcelona, març de 2005

Document Intern de Treball

Different behaviour of a double branch-and-bound algorithm on $Fm \mid prmu \mid C_{max}$ and $Fm \mid block \mid C_{max}$ problems

Ramon Companys[•] Manel Mateo
Laboratori d'Organització Industrial, DOE – ETSEIB – Universitat Politècnica de Catalunya, Avda. Diagonal, 647, 7th. Floor, 08028 Barcelona, Spain.

Abstract

In this paper we face the permutation flow-shop scheduling problem with a makespan objective function in two variants, with and without storage space between machines. We use an improved branch and bound algorithm, suitable for parallel computation, to solve these problems, and auxiliary heuristics to attain an initial good solution. The auxiliary heuristics proposed are built by two steps: in the first step a permutation is obtained; in the second step a local search procedure is applied. The improvement obtained by the local search procedure on NEH heuristic as first step is shown. Since the flow-shop scheduling problem with storage space is a relaxation of the problem without storage space, some elements and procedures developed for that problem can be used in both problems. In particular, some bounding procedures, for instance Nabeshima or Lageweg bounding schema, can be adapted. Moreover, the reversibility property holds on both problems. Consequently the branch and bound algorithm can be applied simultaneously to the direct and the inverse instances. The same sets of data are submitted to heuristics and to the double branch-and-bound algorithm, LOMPEN, assuming first they are instances of flow-shop scheduling problem with storage space and later they are instances of flow-shop scheduling problem without storage space. The algorithms are coded in a similar way; therefore the behaviour and performance can be compared.

Keywords: Scheduling, permutation flow-shop, blocking flow-shop, double branch-and-bound algorithm

1. Introduction.

This paper considers the permutation flow-shop scheduling problem with and without storage space between machines. Both problems consist in scheduling n jobs that must be processed in m stages or machines in the same order. The processing time $p_{j.i}$, of each job i on every machine j, is known. If there is no storage space between stages, then intermediate queues of jobs waiting in the system for their next operation are not allowed. If a job i finishes its operation on a machine j and if the next machine, j+1, is still busy on the previous job, the completed job i has to remain on the machine j blocking it. If there is enough storage space in between machine j and machine j+1, the job i can wait there for the next operation, and machine j is released and can work on another job. We assume that storage space, if it exists, is unlimited. The objective function considered in this paper is the minimisation of the makespan. Using notation of Lawler *et al*. (1993) these problems are denoted by $Fm \mid block \mid C_{max}$ (no storage space) and $Fm \mid prmu \mid C_{max}$ (unlimited storage space).

[•] Corresponding Autor. Email: ramon.companys@upc.edu, Fax: (34)934016054

The flow-shop problem has become one of the most intensively investigated topics in scheduling since the publication of the paper of Johnson (1954) on the $Fm \mid prmu \mid C_{max}$ problem. If the number of machines is two it is easy to attain an optimal schedule in O(n log n) time using the Johnson's algorithm. For m≥3 the problem is shown to be strongly NP-hard (Garey et al. 1976). Lomnicki (1965) and Ignall and Schrage (1965) published almost simultaneously the first branch-and-bound algorithms for the $Fm \mid prmu \mid C_{max}$ problem. Several additional branch-and-bound algorithms have been published after the initial works such as Ashour (1970), Potts (1980), Carlier and Rebai (1996), Cheng *et al*. (1997), Companys (1999), etc.

The complexity of $Fm \mid prmu \mid C_{max}$ problem has encouraged the development of heuristic procedures. A non-exhaustive list of heuristics is Giglio and Wagner (1964), Dudek and Teuton (1964), Palmer (1965), Companys (1966), Campbell *et al*. (1970), Gupta (1971), Dannenbring (1977), Nawaz *et al*. (1983), etc. We can add at this list the application of sophisticated local search procedures usually named metaheuristics.

Recently Ladhari and Haouari (2004) presented a branch-and-bound algorithm that builds the permutations placing alternatively jobs in both ends. It uses as bounding procedure a 2-machine subproblem that gives a tight lower bound, but it cannot be computed in polynomial time. An efficient branch-and-bound algorithm developed by Haouari and Ladhari (2000) is used to solve the subproblem but it is expensive in time (for this reason the tight bound is computed only at certain tree levels). The Ladhari and Haouari algorithm uses NEH heuristic to obtain an initial best solution value, or, for the hard instances, a search heuristic based on an original branch-and-bound schema (Haouari and Ladhari, 2003). Using a Pentium IV 1.8 GHz PC, this algorithm solves 55 of 60 benchmark instances in Taillard (1993) for n=20,50,100 and m=5,10 within a time limit of 3 hours.

Considering now the $Fm \mid block \mid C_{max}$ problem, Hall and Sriskandarajah (1996) published a review on flow-shop with blocking and no-wait in-process. If the number of machines is two, Reddi and Ramamoorthy (1972) showed there exists a polynomial algorithm, which gives an exact solution. The problem $F2 \mid block \mid C_{max}$ can be reduced to a travelling salesman problem (TSP) with n+1 towns (0, 1, 2, ..., n), the distance from town h to town i ($h \neq i$) being:

$$d_{h,i} = \mid p_{2,h} - p_{1,i} \mid \qquad \text{for } h, i = 0, 1, 2, ..., n$$

assuming $p_{1,0} = p_{2,0} = 0$. The sequence of towns in an optimal path corresponds to an optimal permutation for the original problem. There exists a polynomial algorithm to solve this problem, proposed by Gilmore and Gomory (1964). This algorithm is O(n log n) time (Gilmore *et al.*, 1985).

Hall and Sriskandarajah (1996) showed, using a result from Papadimitriou and Kanellakis (1980), that the $Fm \mid block \mid C_{max}$ problem for $m \geq 3$ machines is strongly NP-hard. McCormick *et al*. (1989) proposed heuristic approaches based on an equivalent maximum flow problem. Leisten (1990) compared 11 heuristics using 90 buffer patterns (with m=2 and m=3) and concluded that the NEH heuristic (Nawaz *et al*., 1983), initially proposed for unlimited buffer, has superior performance than the 10 others.

Our purpose is to analyse the differential behaviour of $Fm \mid prmu \mid C_{max}$ and $Fm \mid block \mid C_{max}$ problems. The same sets of data (n, m and $p_{j,i}$) are submitted to some heuristics (in particular to NEH+) to obtain an initial permutation and then to a double branch-and-bound algorithm, LOMPEN, assuming first as instances of $Fm \mid prmu \mid C_{max}$ problem and then as instances of $Fm \mid block \mid C_{max}$ problem. The algorithms are coded in a similar way; therefore, the results are comparable.

The paper is organized as follows. Section 2 presents a brief description of the problem. Section 3 describes the NEH+ heuristic and the LOMPEN algorithm for the $Fm \mid prmu \mid C_{max}$ problem. Section 4 shows how the algorithm has been adapted to the $Fm \mid block \mid C_{max}$ problem. Section 5 describes the computational experience and Section 6 presents some conclusions.

2. Problem description.

We restrict ourselves to the case where machines and jobs are ready at time 0, and consider the same permutation of jobs in all machines. The processing time of job i (i=1,2,...,n) on machine j (j=1,2,...,m) is $p_{j,i.}$

Let [k] denote the job in the position k for a given permutation of the n jobs. Let $e_{j,k}$ be the instant when job [k] enters into stage j coming from stage j−1 and $f_{j,k}$ the instant when job [k] quits stage j to go to stage j+1 or to storage space if it exists. Taking into account the precedent considerations and assuming $p_{j,i} > 0$, we obtain the following equations for the $Fm \mid prmu \mid C_{max}$ case:

$$e_{j,k} = \max \{ f_{j,k-1} , f_{j-1,k} \} \qquad (1)$$

$$f_{j,k} = e_{j,k} + p_{j,[k]} \qquad (2)$$

where j=1,2,…,m and k=1,2,…,n. For the $Fm \mid block \mid C_{max}$ case we must modify equation (2):

$$f_{j,k} = \max \{ e_{j,k} + p_{j,[k]} , f_{j+1,k-1} \} \qquad (2')$$

Consequently, the $Fm \mid prmu \mid C_{max}$ problem can be seen as a relaxation of the $Fm \mid block \mid C_{max}$ problem.

For k=1, j=1 and j=m, it is understood some appropriate definitions for $f_{j,0}$, $f_{0,k}$ and $f_{m+1,k}$ are necessary. Given a specific permutation [k] (k=1,2,...,n) and taking $f_{j,0}=0$ for all j, and $f_{0,k}=f_{m+1,k}=0$ for all k, the values $e_{j,k}$ and $f_{j,k}$ can be obtained using equations (1) and (2) or (1) and (2'). The makespan is then $F_{max}=C_{max}=f_{m,n}$. Both problems can be stated in the following terms: "given n, m and the $p_{j,i}$ find a permutation of n jobs that minimizes C_{max}".

3. The heuristic and LOMPEN algorithms for $Fm \mid prmu \mid C_{max}$.

3.1. The reversibility property.

Given an instance I, which can be called direct instance, with processing times $p_{j,i}$. Another associated instance I', which can be called inverse instance, can be determined with processing times $p'_{j,i}$:

$$p'_{j,i} = p_{m-j+1,i} \qquad j = 1, 2, ..., m \qquad i = 1, 2, ..., n$$

The value C_{max} in I for a permutation S is the same as the one given in I' for the inverse permutation S'. So, the minimum of maximum completion time is the same for I and I', and the permutations associated to both instances are inverse one each other. It does not matter to solve I or to solve I'. Applying a branch-and-bound algorithm, for instance that proposed by Lomnicki (1965), some authors, as Brown and Lomnicki (1966), McMahon and Burton (1967), have found from computational results that the inverse instance was sometimes solved more efficiently than the direct one. Sometimes the direct instance behaves better for solutions, whereas the inverse instance behaves better for bounds.

3.2. The NEH+ heuristic.

The NEH+ heuristic is the composition of NEH heuristic and a local search improvement procedure. The NEH heuristic was proposed by Nawaz *et. al.* (1983) and it can be summarized as follows:

- step 1: order the n jobs by decreasing $\sum_{j=1}^{m} p_{j,i}$;
- step 2: take the first two jobs and schedule them so as to minimize the partial makespan as if the problem was $Fm \mid prmu \mid C_{max}$ with 2 jobs;
- step 3: for k=3 to n, insert the k-th job into the location in the partial schedule, among the k possible, which minimizes the partial makespan for the $Fm \mid prmu \mid C_{max}$ problem with k jobs; to break ties we take the schedule with less total idle time.

Applying NEH heuristic to an instance I we get a permutation S_1, and a makespan $C_{max}(S_1)$. If we apply NEH heuristic to the inverse instance I', the permutation S_2' obtained is not, in general, the inverse of S_1, and $C_{max}(S_2')$ can be different from $C_{max}(S_1)$. We call NEH2 the heuristic consisting in applying NEH to both instances, direct and inverse, retaining the best solution attained. NEH2 is more efficient than NEH but spends more computational time.

Obviously, in the NEH heuristic, steps 2 and 3 can be applied to an initial job order different from the order indicated in step 1, as do Watson *et al*. (1999), Nagano and Moccellin (2002) and Ronconi (2004).

The NEH+ heuristic performs a local search using as initial incumbent solution that given by NEH. The incumbent solution neighbourhood is defined as the set of $n \cdot (n-1)/2$ permutations obtained swapping any two jobs in the incumbent permutation. The neighbourhood is explored according to an *a priori* fixed order. When a neighbour is

better than the incumbent solution, it becomes the new incumbent solution and the exploration follows in a new neighbourhood. If no neighbour is better than the incumbent solution, a new exploration of the neighbourhood is done now taking into account the permutations with equal C_{max} value than the incumbent solution. These become the new incumbent solutions with a certain probability (for instance, 0.5) only. The number of different incumbent solutions accepted without improvement of C_{max} is limited (for instance, the limit can be 40). We call this local search procedure NEDM-RCT (non-exhaustive descent method with random consideration of ties).

We call NEH2+ the heuristic consisting in applying NEH+ to both instances, direct and inverse, and retaining the best solution attained.

3.3. The LOMPEN algorithm.

Companys (1993, 1999) developed an algorithm (called LOMPEN) based on the reversibility property for the $Fm \mid prmu \mid C_{max}$ problem. LOMPEN algorithm (LOMnicki PENdular algorithm) consists in applying simultaneously branch-and-bound algorithms to instances I and I'. In the following description we introduce first the algorithm applied to each instance and below the links and data exchanges between both processes. Each process consists in the exploration of a tree.

Node definition: At level r (r=0,1,..,n−1) a tree's node corresponds to an initial partial sequence σ of r jobs. At level 0 there is only a node, the tree's root, where σ is void.

Branching rule: Each node can be represented by (σ), a permutation of r jobs (r < n). **J** denotes the set of sequenced jobs in σ and $\overline{\mathbf{J}}$ the set of unsequenced ones. A typical immediate successor of (σ) is (σi) with $i \in \overline{\mathbf{J}}$ for any feasible, at position r+1, job i (to clarify the term feasible, see Rule 2 below).

Lower bounds: Associated to the node (σ), there is a lower bound on the maximum completion time for any sequence beginning with the initial partial sequence σ. Lageweg et al. (1978), including ideas from Nabeshima (1967), proposed the *two-machine bound* where bounds are built by adding three different terms. Given two machine numbers j and k ($1 \le j \le k \le m$), the terms are:

a) the time $f_{j,r}(\sigma)$, in which the machine j is released by the jobs from the partial initial permutation σ;
b) a bound for the time spent by the last job from $\overline{\mathbf{J}}$ after leaving machine k;
c) a bound for the makespan of the subproblem defined with all the operations of jobs from $\overline{\mathbf{J}}$ on machines between j and k, both included.

Every pair (j,k) leads to a bound and the lower bound of (σ), denoted by b(σ), is chosen as the maximum value of the computed two-machine bounds. For k=j, the term (c) is $\sum_{i \in \overline{\mathbf{J}}} p_{j,i}$; for k=j+1, it can be obtained using Johnson's algorithm (Johnson, 1954); and for k>j+1, it must be obtained through a relaxation of the subproblem. We use an associated $F2 \mid l_i, prmu \mid C_{max}$ problem as relaxation.

Current best solution value: Some authors prefer to use the name "upper bound". Initially it is the value of a good solution found using a heuristic method. We obtained good results using the NEH+ heuristic.
When a node at level n–1 is reached, $\overline{\mathbf{J}}$ is a set with only one element, a permutation of n jobs is completely defined adding this left element and b(σ) is the value of the maximum completion time for this permutation. If b(σ) is lower than the current best solution value, then it becomes the new current best solution value.

Node elimination: When all the feasible immediate successors of a node have been generated, the parent node is fathomed. Likewise, if the bound of a node is greater or equal to the current best solution, the node is fathomed. Another way to eliminate a node is by dominance as it can be seen below.

Search strategy: The number of active nodes N is taken into account to select the node from which to branch. Two integer numbers N_0 and N_1 are previously fixed ($0<N_0<N_1$). If $0<N<N_0$, then an active node with lower b(σ) on the lowest level is selected; if $N_0 \le N<N_1$, then an active node with lower b(σ) is selected, and ties are broken choosing the node on the highest level; if $N_1 \le N$, then a node on the highest level is selected, and ties are broken choosing the node with lower b(σ).

When the algorithm is running it can be shown that, in certain cases, no better solutions can be generated branching from a particular active node than those generated branching from another active node. The first node is dominated and, in a single tree, it can be eliminated. For instance, if two nodes (σ_1) and (σ_2) are different permutations of the same set **J** of r jobs, and $f_{j,r}(\sigma_1) \le f_{j,r}(\sigma_2)$ for j=1,2,...,m, then (σ_1) dominates (σ_2), and (σ_2) can be fathomed.

At certain levels of the tree, a better solution than the current best solution can be sometimes obtained completing the partial permutation associated to a node by a heuristic procedure (*razzia* procedure). The *razzia* procedure, for instance, can be implemented by adapting NEH+ heuristic. Given a node σ that defines an initial part of the permutation, with r jobs (r < n), **J** denotes the set of sequenced jobs in σ, and $\overline{\mathbf{J}}$ the set of unsequenced ones. We apply NEH in the following form:

- step 1: order the n–r jobs of $\overline{\mathbf{J}}$ by decreasing $\sum_{j=1}^{m} p_{j,i}$;
- step 2: take the first two jobs and schedule them in positions r+1and r+2 so as to minimize the partial makespan as if the problem was $Fm \mid prmu \mid C_{max}$ with r+2 jobs;
- step 3: for k=3 to n–r, insert the k-th job of $\overline{\mathbf{J}}$ (in the order defined at step 1) into the location in the partial schedule, among the k possible (from r+1 to r+k), which minimizes the partial makespan for the $Fm \mid prmu \mid C_{max}$ problem with r+k jobs; to break ties we take the schedule with less total idle time.

We perform the local search improvement (NEDM-RCT) using as initial incumbent solution that given by the NEH heuristic. The incumbent solution neighbourhood can be defined as the set of $(n–r)\cdot(n–r–1)/2$ permutations obtained swapping any two jobs in the incumbent permutation between the positions r+1 and n. The neighbourhood is explored according an *a priori* fixed order. When a neighbour is better than the

incumbent solution, it becomes the new incumbent solution and the exploration follows in a new neighbourhood. If no neighbour solution is better than the incumbent solution, a new exploration of the neighbourhood is done now taking into account the permutations with equal C_{max} value than the incumbent solution. One of this solutions becomes the new incumbent solution with a certain probability (for instance, 0.5) only. The number of different incumbent solutions accepted without improvement of C_{max} is limited (for instance, the limit can be 40). The *razzias* are applied periodically, for instance each 1000 node explorations, to nodes with $r \geq \alpha \cdot n$, where $0<\alpha<1$ (for instance $\alpha=0.3$).

*3.4. Interrelation between both exploration processes***.**

Both simultaneous processes share the best bound and the best solution value reached, and other data deduced from the following four rules, designed to improve the performance of the algorithm:

Rule 1: While the algorithm is running, let T be the set of active nodes of I, and T' the set of active nodes of I'. If the first jobs in the sequences σ of the nodes in T are only a subset of all jobs, these jobs are only feasible for the last jobs of T' sequences with better value than the value of the current best solution. This information can be useful to improve the bounds.

Rule 2: If a job i is placed before position r+1 in all nodes in T with bound $b(\sigma)<b_0$, then the bound of all nodes in T' with i in the first n–r positions must be, at least, $b(\sigma') \geq b_0$. If b_0 is the current best solution value, the nodes of T' with i in the first n–r positions can be fathomed.

Rule 3 (split function): LOMPEN algorithm can be seen as the application of a set of rules to the couple (T,T'). Let be a partition of T in two subsets T1 and T2. The algorithm is equivalent to the parallel application of the rules to both couples (T1,T') and (T2,T'), both applications sharing the best solution attained (but not the best bound).

Rule 4: The same properties stand from T' versus T.

If rules 1 and 2 are used, the above elimination of nodes based on dominance must be employed only in T or in T'.

A special kind of *razzia* is called "plug-in". It consists into linking a node of T, σ, with r jobs placed defining the subset **J,** with a node of T', σ', with r' jobs placed defining the subset **J'**. **J** and **J'** must do not have common jobs. A variant of NEH+ places the n–r–r' jobs, not included on $\mathbf{J} \cup \mathbf{J'}$**,** between the positions r+1 and n–r'–1, taking into account the initial partial permutation σ and the final partial permutation, the inverse of σ'. Plug-in is applied periodically, for instance each 1000 node explorations, to compatible nodes with $r+r' \geq \beta \cdot n$, where $0<\beta<1$ (for instance $\beta=0.4$).

4. The heuristic and LOMPEN algorithms for Fm | block | C_{max}.

4.1. The NEH+ heuristic.

The NEH+ heuristic can be implemented in a similar way, using equation (2') instead of (2), as it was described in Section 3.1. In the NEH initial step, the ties in the partial completion time are broken taking into account the sum of idle and blocked times on machines. The reversibility property of unconstrained flow-shop problems can be extended to flow-shop without storage space between stages. We can apply NEH2+ heuristic to Fm | block | C_{max} instances.

4.2. The LOMPEN algorithm.

We can adapt the LOMPEN algorithm, designed for unconstrained flow-shop to this problem. There is no difference in *node definition, branching rule, node elimination* and *search strategy*. The bounding procedures can be improved taking into account the following observations, deduced from equation (2').

At node (σ) from level r>0, with r jobs placed, the unknown values $f_{j,t}(\sigma)$ (t=r+1,r+2,..., n; j=1,2,...,m) are bounded by the values $D_{j,t}$. For machine j=1:

$$D_{1,r+1} = f_{2,r}(\sigma)$$
$$D_{1,r+2} = f_{3,r}(\sigma)$$
$$\text{.....}$$
$$D_{1,r+m-1} = f_{m,r}(\sigma)$$

and, more generally for machines 2,3,..,m–1:

$$D_{j,r+s} = f_{j+s,r}(\sigma) \qquad \text{with } r + s \le n \text{ and } j \le m - s$$

These expressions allow the bounding of blocking time in machines produced by the last n–r jobs by a comparison of the values $D_{j,t}$ with the cumulated processing time of these n–r jobs, in a similar way as Ronconi and Armentano (2001) do.

For each machine j, the values $p_{j,i}$ for $i \in \overline{\mathbf{J}}$ are considered in a decreasing order and recalled as $\overline{p}_{j,s}$ with s=1,2,...,n–r, and $\overline{P}_{j,s} = \sum_{t=1}^{s} \overline{p}_{j,t}$. A lower bound, $BT_j(\sigma)$, of the blocking time on machine j produced by the processing of the jobs from $\overline{\mathbf{J}}$ can be computed in the following form:

$$bt_s = bt_{s-1} + \max \{ 0 ; D_{j,r+s} - (f_{j,r}(\sigma) + \overline{P}_{j,s} + bt_{s-1}) \} \text{ for } s = 1, 2, .., u \text{ with } bt_0 = 0$$

$$BT_j(\sigma) = bt_u \qquad \text{where } u = \min \{ n–r , m–j \}$$

The two-machine bounds can be adapted to the constrained flow-shop problem modifying the term (c) in Section 3 in the following way:

If k=j, term (c) is $\sum_{i \in \overline{\mathbf{J}}} p_{j,i.} + BT_j(\sigma)$.

If k=j+1, term (c) can be estimated, following the suggestion by Reddi and Ramamoorthy (1972) as extension of the Nabeshima bounds (1967), solving the associated TSP with n–r+1 towns. The distances of the associated TSP are based on the processing times on machines j and j+1 related to the n–r jobs of the set $\overline{\mathbf{J}}$ and a supplementary town 0 with fictitious processing times:

$p_{1,0} = 0$

$p_{2,0} = f_{j+1,r}(\sigma) - f_{j,r}(\sigma)$

If k>j+1, the third component can be estimated in the same way to the unconstrained case.

Initially, the *current best solution value* is the value of a solution found using a heuristic procedure, for instance the NEH+ heuristic. When a node σ at level n–1 is reached, $\overline{\mathbf{J}}$ is a set with only one element, for instance the job i. A permutation of n jobs is completely defined adding this left element to σ, (σi); the maximum completion time for this permutation, $f_{m,n}(\sigma i)$, can be calculated (the obtained $b(\sigma)$, adapting the two-machine bound, is not necessarily $f_{m,n}(\sigma i)$). If $f_{m,n}(\sigma i)$ is lower than the current best solution value, then it becomes the new current best solution value.

As in the unconstrained case, branch-and-bound algorithms are applied to I and to I' simultaneously. Both simultaneous processes share the best bound and the best solution value reached, and other data provided by the application of the four precedent rules. It is possible to eliminate nodes in T or in T' using dominance rules.

5. Computational experience.

Vivó (1994) developed a version of LOMPEN, that we call LOMPEN_2, coded in C++, to experiment with instances of moderate size. We used with success this old code on 20 instances drawn from Taillard (1993) on $Fm \mid prmu \mid C_{max}$ case but it failed to solve instances for n≥50 and m≥5. Additionally this code was unable to deal with $Fm \mid block \mid C_{max}$ case**.** Both reasons suggested the interest on the development of new codes for this purpose.

We developed two independent experiences with two different codes: the first experience used the code LOMPEN_4 on nine generated sets of a thousand instances each one (m=3,4,5; n=13,14,15); and the other experience used the code LOMPEN_5 on the Taillard instances (m=5,10,20; n=20,50,100). The codes are identical for $Fm \mid prmu \mid C_{max}$ and $Fm \mid block \mid C_{max}$ except the use of equations (2) or (2') and the bounding procedure for k=j and k=j+1.

The first experience was designed specially to evaluate the differential impact of local search (NEDM-RCT) to improve the initial best solution value on the $Fm \mid prmu \mid C_{max}$ and the $Fm \mid block \mid C_{max}$ cases. The second experience had as the main

objective to test the performance of the LOMPEN algorithm on larger size instances belonging to both cases.

5.1. Experience 1.

The performance of the procedures was evaluated, as we said, on nine generated sets of a thousand instances, the value m being 3, 4 and 5 and the value n being 13, 14 and 15. Processing times were randomly generated (using a uniform distribution between 1 and 25). For each instance we computed the permutation and the associated value C_{max}, given by LOMPEN. NEH+, as described in sections 3.2 and 4.1, was used to obtain the initial best solution value.

LOMPEN_4 was coded in Visual BASIC and implemented on a Pentium IV 2.8 GHz PC with 512 Mb RAM. The implemented algorithm does not use additional rule 3 (split function), *razzia* procedure or dominance elimination of nodes. The values N_0 and N_1, used by the search strategy, are: $N_0 = 100$ and $N_1 = 150$.

To measure the NEDM-RCT impact, we evaluated the improvement on NEH+ solution in relation to NEH solution. We determine the relative discrepancy in percentage for each instance before and after NEDM-RCT is applied. The relative discrepancy is equal to the difference between the heuristic value and the optimum value (obtained later by LOMPEN_4) divided by the optimum value.

Table 1 presents the mean and maximum relative discrepancy associated to the NEH and NEH+ heuristics in the Fm | prmu | C_{max} case. The mean discrepancy is reduced by the local search procedure; the reduction factor is a value between 0.2 and 0.4.

m	*n*	*mean discrepancy*		*maximum discrepancy*	
		NEH	NEH+	NEH	NEH+
3	13	0.360	0.071	10.215	4.566
	14	0.353	0.086	6.944	4.020
	15	0.301	0.061	6.731	6.161
4	13	1.256	0.317	12.500	6.341
	14	1.092	0.345	9.322	5.856
	15	1.063	0.319	8.190	7.087
5	13	2.076	0.793	11.688	8.444
	14	2.037	0.793	10.569	9.211
	15	2.017	0.775	11.161	7.200

Table 1: Mean and maximum relative discrepancies (%) of solutions given by heuristics for the Fm | prmu | C_{max} case.

The computational times of LOMPEN_4 (including the determination of the initial solution) in the Fm | prmu | C_{max} case are presented in table 2.

		mean CPU *time* m=3	m=4	m=5	*max* CPU *time* m=3	m=4	m=5
n	13	0.048	0.135	1.208	14.6	7.3	931.2
	14	0.042	0.208	0.611	8.6	19.2	140.2
	15	0.036	0.290	1.250	5.1	61.5	541.9

Table 2: Mean and maximum CPU time (seconds/instance) of LOMPEN_4 for the Fm | prmu | C_{max} case.

Instance #952, for m=5 and n=13, is the hardest one for the LOMPEN_4 algorithm. This instance spends 931.2 seconds of CPU time.

m *n*		*mean discrepancy*		*maximum discrepancy*	
		NEH	NEH+	NEH	NEH+
3	13	4.065	1.846	13.636	6.736
	14	4.160	1.918	13.107	9.501
	15	4.471	2.096	12.613	8.439
4	13	4.289	2.211	13.475	7.725
	14	4.412	2.312	11.842	10.044
	15	4.917	2.496	13.063	9.009
5	13	4.152	2.239	11.715	9.623
	14	4.484	2.346	12.955	7.510
	15	4.782	2.501	14.286	8.108

Table 3: Mean and maximum relative discrepancies (%) of solutions given by heuristics for the Fm | block | C_{max} case.

We have used the same set of data considering the Fm | block | C_{max} problem and the coherent LOMPEN_4 implementation (with the same features that were used to obtain the results in tables 1 and 2).

Table 3 presents the mean and maximum relative discrepancy associated to the NEH and NEH+ heuristics in the Fm | block | C_{max} case. The impact of NEDM-RCT on the initial solution is important. The average absolute improvement on the mean relative discrepancy produced by the local search is about 2%.

The computational times of LOMPEN_4 algorithm (including the determination of the initial solution) in the Fm | block | C_{max} case are presented in table 6.

		mean CPU *time* m=3	m=4	m=5	*max* CPU *time* m=3	m=4	m=5
n	13	0.705	2.026	3.639	207	42	67
	14	3.348	7.441	18.984	1128	180	372
	15	7.772	28.321	70.737	787	879	1494

Table 4: Mean and maximum of CPU time (seconds/instance) of LOMPEN_4 for the Fm | block | C_{max} case.

The greater the number of machines is, the lower the quality of heuristic solutions becomes, and also the time spent in the LOMPEN_4 algorithm increases exponentially. Incidentally, instance #952 for m=5 and n=13 spends only 3.24 seconds in the

Fm | block | C_{max} case; in this case only three instances spend more than 931.2 seconds. Instance #104 for m=5 and n=15 spends 1494 seconds of CPU time. On the other hand, this instance in the Fm | prmu | C_{max} case spends less than 1 second.

The mean relative discrepancies in table 1 are shorter than those in table 3 showing the superior efficiency of NEH+ on Fm | prmu | C_{max} than on Fm | block | C_{max}. Additionally, the average CPU times are shorter in table 2 than those showed in table 4. The superior performance of LOMPEN_4 on Fm | prmu | C_{max} case is due not only to the superior efficiency of NEH+ heuristic, but also to the superior efficiency of two-machine lower bound procedure for Fm | prmu | C_{max} problem. To confirm this we define the relative discrepancy on the initial bound as the difference between the optimum value and the initial best lower bound value divided by the optimum value. Table 5 presents the mean and maximum bound relative discrepancy for the Fm | prmu | C_{max} and Fm | block | C_{max} case.

m	*n*	*Fm | prmu | C_{max}*		*Fm | block | C_{max}*	
		mean	*max*	*mean*	*max*
3	13	0.188	3.382	3.388	11.707
	14	0.167	3.766	3.370	13.500
	15	0.036	5.051	3.217	10.385
4	13	0.619	5.504	5.964	15.353
	14	0.546	6.667	5.837	16.438
	15	0.475	4.390	5.950	15.200
5	13	1.457	9.217	7.467	17.460
	14	1.296	7.265	7.672	17.467
	15	1.122	6.224	7.828	16.471

Table 5: Mean and maximum relative discrepancies (%) of initial best lower bound given by two machines bound procedure for the Fm | prmu | C_{max} and Fm | block | C_{max} cases.

The mean bound relative discrepancy for the Fm | block | C_{max} case is of the same order than the maximum bound relative discrepancy for the Fm | prmu | C_{max} case.

5.2. Experience 2.

This experience had as main objective to test the performance of the LOMPEN algorithm on large size instances belonging to Fm | prmu | C_{max} and Fm | block | C_{max} cases. The code, LOMPEN_5, was done by Alemán (2004) in C using Dev-C++, compiled with GCC and carried out on a Pentium IV 2.8 GHz PC with 512 Mb RAM running KNOPPIX/Debian GNU/Linux 3.4. LOMPEN_5 obtains initially ten solutions by applying five direct heuristic procedures, including Palmer (1965), trapezes (Companys, 1966), Gupta (1971), NEH (Nawaz *et al.*, 1983) and PF (McCormick *et al.*, 1989) to the direct and inverse instance. These solutions are improved by local search. The best of the last ten solutions is retained as the initial best solution value. LOMPEN_5 uses the split function and the *razzia* procedure. The values N_0 and N_1 are function of the number of jobs, n: $N_0 = 2 \times n$ and $N_1 = 2 \times n + 100$.

The experimental data considered were the well-known instance sets elaborated by Taillard (1993), though the criticism of some authors as Watson *et al* (1999). In the

Fm | prmu | C_{max} case, the performance of LOMPEN_5 was evaluated on 70 instances drawn from Taillard, with m=5,10 and n=20,50,100,200. In all of these instances LOMPEN_5 attained an optimum solution. Additionally LOMPEN_5 solved tail0104 and tail0106 with n=200 and m=20 in 28.4 hours (Taillard, 2005), but it was unable to solve exactly the whole Taillard instances with m=20.

Table 6 shows the results of LOMPEN_5 when the algorithm attained the optimum in the Fm | prmu | C_{max} case. The headings have the following meaning:

instance:	name of the instance
FBSV:	final best solution value provided by LOMPEN_5 (corresponding to the optimum solution in the Fm | prmu | C_{max} case)
NN:	number of explored nodes
time:	total CPU time to run LOMPEN_5 in seconds

For m=5, all optimum solutions were attained in a time of less than five minutes per instance. Three instances belonging to the set m=10 and n=50 (tail042, tail043 and tail050) spend more time than the hardest instance with m=10 and n=200 (tail092).

For the Fm | block | C_{max} case, LOMPEN_5 do not attain, in a reasonable time, any checked optimum solution on Taillard instances. We use initially LOMPEN_5 as a heuristic, limiting CPU time to 20 minutes, on 100 Taillard instances (n=20,50,100,200 and m=5,10,20). The solutions attained are shown in Table 7.

The final best solution values, FBSV, is, in general, no optimal, but the 7 solutions marked with * are checked to be optimal. In order to confirm the optimality, we have used the developed codes (LOMPEN_4 and LOMPEN_5) feeding as initial solution the best solution known, with a CPU time limited to 12 hours. The instances tail003 and tail004 are equally hard on the Fm | prmu | C_{max} case, but on the Fm | block | C_{max} case tail003 is harder than tail004.

n × m	*instance*	*FBSV*	*NN*	*time*	*n × m*	*instance*	*FBSV*	*NN*	*time*
20 × 5	tail001	1278	0	~0	20 × 10	tail011	1582	279868	224
	tail002	1359	6443	~0		tail012	1659	1102037	899
	tail003	1081	9092	1		tail013	1496	1082041	766
	tail004	1293	8684	1		tail014	1377	59330	510
	tail005	1235	2643	~0		tail015	1419	179341	133
	tail006	1195	2397	~0		tail016	1397	32961	39
	tail007	1234	1392	~0		tail017	1484	9008542	2029
	tail008	1206	18	~0		tail018	1538	1374952	1168
	tail009	1230	1740	~0		tail019	1593	1576	3
	tail010	1108	308	~0		tail020	1591	1267178	1055
50 × 5	tail031	2724	4	1	50 × 10	tail041	2991	4299295	8979
	tail032	2834	11315	5		tail042	2867	56621519	144884
	tail033	2621	9356	6		tail043	2839	59515405	99773
	tail034	2751	43507	11		tail044	3063	761658	1208
	tail035	2863	22071	8		tail045	2976	10846799	24220
	tail036	2829	7921	4		tail046	3006	2453293	5804
	tail037	2725	16631	5		tail047	3093	23202058	48905
	tail038	2683	0	1		tail048	3037	3721133	5726
	tail039	2552	44072	18		tail049	2897	619246	1301
	tail040	2782	1	1		tail050	3065	111514192	243182
100 × 5	tail061	5493	0	10	100 × 10	tail071	5770	606706	2266
	tail062	5268	70082	139		tail072	5349	422039	1888
	tail063	5175	12790	13		tail073	5676	187156	1846
	tail064	5014	29898	50		tail074	5781	2846107	11045
	tail065	5250	15	11		tail075	5467	2949863	11283
	tail066	5135	20	10		tail076	5303	295	109
	tail067	5246	100444	266		tail077	5595	501507	3532
	tail068	5094	23445	73		tail078	5617	2029669	9643
	tail069	5448	0	13		tail079	5871	3230743	16725
	tail070	5310	25331	53		tail080	5845	331807	1371
					200 × 10	tail091	10862	778448	18558
						tail092	10480	2846647	59250
						tail093	10922	3315177	50720
						tail094	10889	332285	7555
						tail095	10524	171161	14337
						tail096	10329	696102	25231
						tail097	10854	1187177	27310
						tail098	10730	1356655	27875
						tail099	10438	893185	14842
						tail100	10675	2334787	33592
					200×20	tail104	11275	n.a.	n.a.
						tail106	11176	12457281	138217

Table 6: number of explored nodes and CPU times (in seconds) for Taillard instances using LOMPEN_5 (Fm | prmu | C_{max} case)

$n \times m$	*instance*	*FBSV*	$n \times m$	*instance*	*FBSV*	$n \times m$	*instance*	*FBSV*
	tail001	1374*		tail011	1701		tail021	2436
	tail002	1408*		tail012	1833		tail022	2236
	tail003	1280		tail013	1659		tail023	2479
	tail004	1448*		tail014	1535		tail024	2348
20 × 5	tail005	1341*	20 × 10	tail015	1617	20 × 20	tail025	2439
	tail006	1363*		tail016	1590		tail026	2389
	tail007	1381*		tail017	1622		tail027	2390
	tail008	1379		tail018	1731		tail028	2328
	tail009	1373*		tail019	1747		tail029	2363
	tail010	1283		tail020	1782		tail030	2324
	tail031	3024		tail041	3710		tail051	4574
	tail032	3234		tail042	3559		tail052	4357
	tail033	3055		tail043	3556		tail053	4358
	tail034	3145		tail044	3728		tail054	4449
50 × 5	tail035	3189	50 × 10	tail045	3696	50 × 20	tail055	4346
	tail036	3209		tail046	3662		tail056	4379
	tail037	3071		tail047	3751		tail057	4391
	tail038	3101		tail048	3634		tail058	4415
	tail039	2940		tail049	3603		tail059	4412
	tail040	3163		tail050	3700		tail060	4508
	tail061	6222		tail071	7119		tail081	7969
	tail062	6107		tail072	6881		tail082	7986
	tail063	5986		tail073	7009		tail083	7968
	tail064	5827		tail074	7272		tail084	7952
100 × 5	tail065	6073	100 × 10	tail075	6944	100 × 20	tail085	7980
	tail066	5916		tail076	6743		tail086	8033
	tail067	6077		tail077	6877		tail087	8086
	tail068	5984		tail078	6955		tail088	8127
	tail069	6214		tail079	7167		tail089	8021
	tail070	6199		tail080	7093		tail090	8066
				tail091	13543			
				tail092	13484			
				tail093	13623			
				tail094	13464			
			200 × 10	tail095	13498			
				tail096	13299			
				tail097	13699			
				tail098	13643			
				tail099	13458			
				tail100	13579			

Table 7: Solutions attained using LOMPEN_5 as a heuristic, maximum CPU time = 20 minutes (Fm | block | C_{max} case).

6. Conclusions.

The impact of local search improvement on NEH heuristic is shown in this paper. This impact can be extended to other similar heuristics. We defined a local search improvement procedure that takes into account the ties (NEDM-RCT). An improvement consisting on the application of heuristics to direct and inverse instances is suggested.

We showed the application of LOMPEN to Fm | block | C_{max} problem, the adaptation of two machines bound to this problem and the links between the direct and inverse explorations.

The differential behaviour of Fm | prmu | C_{max} and Fm | block | C_{max} instances, when they are submitted to heuristics and to LOMPEN, is described. We pointed the reasons, at the present level of knowledge, that provoked the Fm | block | C_{max} problem is harder than Fm | prmu | C_{max} problem.

The results of application of LOMPEN to large instances drawn from Taillard instance sets are included. The first computational experiences suggested the potential of the LOMPEN algorithm for large instances. A LOMPEN implementation solved 8 unsolved instances: tail106 (m=20, n=200) in the Fm | prmu | C_{max} case and tail001, tail002, tail004, tail005, tail006, tail007, tail009 in the Fm | block | C_{max} case.

The same data set can correspond to a hard instance on the Fm | prmu | C_{max} and to a soft instance Fm | block | C_{max} case, and reciprocally. We think it is not possible to estimate the behaviour of a set of data on the Fm | block | C_{max} case knowing the behaviour of the same set on the Fm | prmu | C_{max} case.

The behaviour of instance # 952 (n=13, m=5) on the Fm | prmu | C_{max} case could indicate that 1000 random instances are not enough to deal with a complete range of variability (the initial best solution in # 952 instance had the optimum value).

The behaviour of the 20×50 Taillard instances suggests that the Taillard instances are not homogenously hard. The 20×50 instances are harder than the 100×50 or 200×50 instances, despite the increment of complexity induced by the higher number of jobs.

The main idea of the LOMPEN algorithm is the ability to run simultaneously two parallel branch and bound procedures exchanging data between them. Therefore LOMPEN is suitable for parallel computation, but a further examination will be required to fit this objective.

Acknowledgements

The authors are indebted to editor and two anonymous referees whose comments helped to improve the paper. Grant DPI2001-2169 (MCYT) supported this research.

References.

Alemán, A., 2004. *Estudio y Aplicación del Algoritmo Lomnicki Pendular al Problema Fm|block|Fmax*, PFC (Engineer Thesis), ETSEIB-UPC.

Ashour, S. A., 1970. "A branch-and-bound algorithm for the flow shop scheduling problem", *AIIE Transactions* **2**, pp. 172-176.

Brown, A. P. G., Lomnicki, Z. A., 1966. "Some applications of the 'branch-and-bound' algorithm to the machine scheduling problem", *Operational Research Quarterly*, **17** (4), pp. 173-186.

Campbell, H. G., Dudek, R. A., Smith, M. L., 1.970. "A heuristic algorithm for the n job m machine sequencing problem", *Management Science* **16** (10), pp. 630-637.

Carlier J., Rebaï, I., 1996. "Two branch and bound algorithms for the permutation flow shop problem", *EJOR* **90**, pp. 238-251.

Cheng, J., Kise, H., Matsumoto, H., 1997. "A branch-and-bound algorithm with fuzzy inference for a permutation flowshop scheduling problem", *EJOR* **96**, pp. 578-590.

Companys, R., 1966. "Métodos heurísticos en la resolución del problema del taller mecánico", *Estudios Empresariales* **5** (2), pp. 3-14.

Companys, R., 1993. "Algoritmo de Lomnicki pendular", *Notas sobre el problema del taller mecánico*, D.I.T. 93/12, ETSEIB-UPC, pp. 51-58.

Companys, R., 1999. "Note on an improved branch-and-bound algorithm to solve n/m/P/Fmax problems", *TOP* **7** (1), pp. 25-31.

Dannenbring, D. G., 1977. "An evaluation of flow-shop sequencing heuristics", *Management Science* **23**, pp. 1174-1182.

Dudek, R. A., Teuton Jr., O. F., 1964. "Development of m-stage decision rule for scheduling n jobs trough m machines", *Journal of the Operations Research Society of America* 12 (3), pp. 471-497.

Garey, M. R., Johnson, D. S., Sethi, R., 1976. "The complexity of flow shop and job shop scheduling", *Mathematics of Operations Research* **1**, pp. 117-129.

Giglio, R. J., Wagner, H. M., 1964. "Approximate solutions to the three-machine scheduling problem", *Operations Research* **12** (2).

Gilmore, P. C., Gomory, R. E., 1964. "Sequencing a one state-variable machine: a solvable case of the traveling salesman problem", *Operations Research* **12**, pp. 655-665.

Gilmore, P. C., Lawler, E. L., Shmoys, D. B., 1985. "Well-solved special cases" in Lawler, E.L., Lenstra, K. L.,Rinooy Kan, A. H. G., Shmoys, D. B., (eds.) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, pp. 87-143.

Gupta, J. N. D., 1971. “A functional heuristic algorithm for the flow-shop scheduling problem”, *Operational Research Quarterly* **22** (1).

Hall, N. G., Sriskandarajah, C., 1996. “A survey of machine scheduling problems with blocking and no wait in process”, *Operations Research* **44** (3), pp. 510-525.

Haouari, M., Ladhari, T., 2000. “Minimizing maximum lateness in a two-machine flowshop” *Journal of the Operational Research Society* **51**, pp. 1100-1106.

Haouari, M., Ladhari, T., 2003. “A branch-and-bound-based local search method for the flow shop problem” *Journal of the Operational Research Society* **54**, pp. 1076-1084.

Ignall, E., Schrage, L. E., 1965. “Application of the branch-and-bound algorithm to some flow shop problems”, *Operations Research* **13**, pp. 400-412.

Johnson, S. M., 1954. "Optimal two- and three-stage production schedules with set-up time included", *Naval Research Logistics Quarterly* **1** (1), pp. 61-68.

Ladhari, T., Haouari, M., 2005. “A computational study of the permutation flow shop problem based on a tight lower bound”, *Computers & Operations Research* **32**, pp. 1831-1847.

Lageweg, B. J., Lenstra, J. K., Rinnooy Kan, A. H. G., 1978. “A general bounding scheme for the permutation flow-shop problem”, *Operations Research* **26** (1), pp. 53-67.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B., 1993. “Sequencing and scheduling: algorithms and complexity”, *in* Graves, C. G., Rinnooy Kan, A. H. G., Zipkin, P. (eds) *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, North-Holland, Amsterdam, pp. 445-522.

Leisten, R., 1990. “Flowshop sequencing problems with limited buffer storage”, *Int. J. Prod. Res.* **28** (11) 2085-2100.

Lomnicki, Z. A., 1965. “A branch and bound algorithm for the exact solution of three-machine scheduling problems”, *Operational Research Quarterly* **16** (1), pp. 89-100.

McCormick, S. T., Pinedo, M. L., Shenker, S., Wolf, B., 1989. “Sequencing in an Assembly Line with Blocking to Minimize Cycle Time”, *Operations Research* **37**, pp. 925-935.

McMahon, G. B., Burton, P. G., 1967. “Flow-shop scheduling with the branch-and-bound method”, *Operations Research* **15** (3), pp. 473-481.

Nabeshima, I., 1967. “On the bound of makespans and its application to m machine scheduling problem”, *Journal of the Operations Research Society of Japan* **9**, pp. 98-136.

Nagano, M. S., Moccellin, J. V., 2002. "A hight quality solution constructive heuristic for flow shop sequencing", *Journal of the Operational Research Society* **53**, pp. 1374-1379.

Nawaz, M., Enscore E. E., Ham, I., 1983. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem", *OMEGA* **11**, pp. 91-95.

Palmer, D. S., 1965 "Sequencing jobs through a multi-stage process in the minimum total time – A quick method of obtaining a near optimum" *Operational Research Quarterly* **16** (1), pp. 101-107.

Papadimitriou, C. H., Kanellakis, P. C., 1980. "Flowshop scheduling with limited temporary storage", *Journal of the ACM* **27** (3) pp. 533-549.

Potts, C. N., 1980. "An adaptive branching rule for the permutation flow-shop problem", *European Journal of Operational Research* **5**, pp. 19-25.

Reddi, S. S., Ramamoorthy, C. V., 1972. "On the flow-shop sequencing problem with no wait in process", *Operational Research Quarterly* **23** (3), pp. 323-331.

Ronconi, D. P., 2004 "A note on constructive heuristics for the flowshop problem with blocking" *Int. J. Production Economics* **87** pp. 39-48.

Ronconi, D. P., Armentano, V. A., 2001. "Lower bounding schemes for flowshops with blocking in-process", *Journal of the Operational Research Society* **52**, pp. 1289-1297.

Taillard, E., 1993. "Benchmarks for basic scheduling problems", *European Journal of Operational Research* **44**, pp. 375-382.

Taillard, E., 2005. "Prof. Éric Taillard, problem instances, Flow shop sequencing, Summary of best known lower and upper bounds of Taillard's instances" [February 24, 2005]
http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/flowshop.dir/best_lb_up.txt

Vivó, R., 1994. *Estudio e Implementación del Algoritmo Lomnicki Pendular*, PFC (Engineer Thesis), ETSEIB-UPC.

Watson, J. P., Barbulescu, L., Howe, A. E., Whitney, L. D., 1999. "Algorithm performance and problem structure for flow-shop scheduling" *Proceedings 16th National Conference on AI*.

SOME COMMENTS TO THE EDITOR AND REFEREE'S REMARKS

Referee 1: *More importantly the paper does not appear to make reference to some recent and highly relevant work such as:*

MS Nagano and JV Moccellin "A hight quality constructive heuristic for flow shop sequencing" J. Ops. Res. Soc. 53 1374-1379 (2002)

HD Pour "A new heuristic for the n-job, m-machine flow-shop problem" Production Planning&Control 12 (7) 648-653 (2001)

JP Watson et al. "Algorithm performance and problem structure for flow-shop scheduling" Proceedings 16th National Conference on AI (1999)

The paper not compare the performance of its heuristic with those in the Nagano & Moccellin and Pour papers.

Editor: *Comparisons with algorithms Nagano & Moccellin and Pour must be done.*

We did not intend to participate on a heuristic competition. Nevertheless, we experimented with the Nagano & Moccellin (N&M) and Pour procedures. We have coded N&M and Pour heuristics for both $Fm \mid prmu \mid C_{max}$ and $Fm \mid block \mid C_{max}$ cases (before we have adapted Pour and N&M heuristics to the $Fm \mid block \mid C_{max}$ case). First we have run them on the 9 instance sets used on Experience_1. Our conclusion is that N&N, as indicated in the paper of Nagano and Moccellin, is as good as NEH and, as it is not indicated, NEH is as good as N&M. Pour heuristic gives worse solutions and spends more computational time. Our comments on suggested papers are the following.

Nagano and Moccellin: Our conclusion is that there is not advantage to change NEH+ by N&M+. The article of Nagano and Moccellin has some weak points:

- There is not indicated how ties are solved (there is a high probability of ties with processing times between 1 and 10).
- A bound, taken from Taillard (1993), is used to compute a discrepancy value. This bound can easily be improved, and we used this improved form for many years[1]. The N&M proposed bound is (in our own nomenclature):

  $$LBM = \max \{\max_j \{P_{1,j} + \sum_{i=1}^{n} p_{j,i} + P_{j,m}\}, \max_i \{\sum_{j=1}^{m} p_{j,i}\}\}$$

  where $P_{1,j} = \min_i \sum_{k=1}^{j-1} p_{k,i}$ and $P_{j,m} = \min_i \sum_{k=j+1}^{m} p_{k,i}$

  The improved bound is:

  $$LBM = \max \{\max_j \{Q_j + \sum_{i=1}^{n} p_{j,i}\}, \max_i \{\sum_{j=1}^{m} p_{j,i} + \sum_{h \neq i} \max\{p_{1,h}, p_{m,h}\}\}\}$$

  where $Q_j = \min_{h \neq i} \{\sum_{k=1}^{j-1} p_{k,h} + \sum_{k=j+1}^{m} p_{k,i}\}$
- The number of machines m is not used in the discussion of results.
- The main histogram is misleading. The floor (abscise axe) is placed on 60 %.
- There are some problems of nomenclature (P_i or $\Sigma\ TP_i$?)
  Nevertheless, in the new version we cite the Nagano and Moccellin paper.

[1] R. Companys (1983) *Problema del Taller Mecánico* CPDA-ETSEIB

Pour: Our conclusion is that the Pour procedure gets worse results than NEH or N&M heuristics. In a recent paper[2] Ruiz and Maroto compared 25 heuristics, using the Taillard sets of instances, and they arrived to the conclusion:

> "… The results obtained by the author (Davoud Pour) claiming to be a better heuristic than NEH, clearly do not hold with this benchmark… Davoud Pour's heuristic times grow exponentially with the size of the problem considered, … We think these time requirements are unbearable, especially considering this method's low performance."

There is not advantage to change NEH+ by Pour+. The article of Pour has some weak points:

- The formulation of CDS heuristic is incorrect.
- There is not indicated how ties are solved. The ties can appear in many steps of the algorithm.
- The size of the numerical example included, (m=3, n=3), does not allow to check all the algorithm features.
- The value choice of the couples (n,m) for the experimental data is surprising for us (for instance, m=2?).
- The index definition is ambiguous: EI = 100×H/F must be EI = H/F, EI and RE are related if they are referred to the same parameter; the values shown on the table are not coherent with the precedent observation, etc.
- The F_{max} scale of figure 1 attains the value limit 1000, but for m=65 and n=65 the average F_{max} value must be 6450.
- The impact of m is not discussed. In fact, the paper argues that the number of machines has not influence on the number of iterations and then the procedure is indicated for a great m. The increment of computational time when m increases is not taken into account.

JP Watson *et al*.: The main argument is "the processing times on real problems are not random as in Taillard instances, the performance that an algorithm obtains in a popular test benchmark instances is reduced if instances have a modest level of randomness, and the instances with a certain structure do not develop a topological space similar to the developed by the test benchmark instances". The authors built a PSP Generator where the processing time of instances was generated with a different kind of randomness (normal distributed processing times, machine correlation and job correlation). The authors do not justify why his instance sets are more realistic than Taillard instance sets. The authors choose 7 procedures (mainly introduced on papers presented to Artificial Intelligence conferences). They compare the solutions given by each algorithm to the best solution obtained for each instance. The NEH-RS algorithm (NEH with random starts) is the winner (we can suppose that NEH-RS+ would perform better). The article of JP Watson *et al*. has some weak points:

- There is no a wide spectrum of m values: m=20 with n=50,100,200
- The link between the α value and the overlapping of distributions is correct only in probability. If for each α value the instance number is 10, the possibility of estimation error in the conclusions has a very high risk.
- The heuristics chosen are, saying it in a polite form, peculiar.

[2] R. Ruiz and C. Maroto (2005) "A comprehensive review and evaluation of permutation flowshop heuristics" EJOR **165**, pp. 479-494.

- It is a lack of fair play. To compare heuristics, it is convenient to present computational times. Does NEH spend the same time than NEH-RS? If NEH-RS spends more time than NEH, we consider understandable a superior performance from NEH-RS.
  In the new version we cite the JP Watson *et al*. paper.

Our tests on Pour and N&M procedures are based on the following remarks. Applying a heuristic to an instance *I* we obtain a permutation *S* and a value Cmax(*S*). We call **symmetric heuristic** a heuristic such that applied to the inverse instance *I'* gives *S'* the inverse permutation of *S*, and **asymmetric heuristic** in the other cases. Palmer, trapezes, CDS, etc. are symmetric heuristics, NEH, Pour and N&M are asymmetric heuristics, and it is possible to define a new heuristic applying the original heuristic to the direct and inverse instances retaining the best of both solutions. With the 9 sets of instances used on Experience_1 we obtain the results that are shown on tables 1 and 2. NEH2, Pour2 and N&M2 are the names chosen for this application of heuristic to both instances, direct and inverse. NEH2+, Pour+ and N&M2+ correspond to the application to the solutions given on direct and inverse instances by NEH, Pour or N&M algorithms the local search procedure (NEDM-RCT), retaining the best solution attained.

m	n	*NEH2*	*POUR2*	*N&M2*	*NEH2+*	*POUR2+*	*N&M2+*
3	13	0.234	2.705	0.228	0.030	0.070	0.024
	14	0.243	2.492	0.253	0.035	0.088	0.032
	15	0.188	2.234	0.180	0.036	0.075	0.026
4	13	0.887	5.853	0.909	0.201	0.268	0.205
	14	0.829	4.727	0.852	0.209	0.302	0.211
	15	0.761	4.723	0.783	0.210	0.277	0.204
5	13	1.605	6.778	1.685	0.482	0.555	0.458
	14	1.626	6.765	1.648	0.489	0.600	0.503
	15	1.527	6.751	1.693	0.477	0.623	0.517

Table 1: Mean relative discrepancies (%) of solutions given by the six heuristics for the Fm | prmu | C_{max} case.

m	n	*NEH2*	*POUR2*	*N&M2*	*NEH2+*	*POUR2+*	*N&M2+*
3	13	3.444	10.403	3.438	1.382	2.298	1.422
	14	3.524	10.579	3.610	1.414	2.298	1.470
	15	3.818	10.674	3.818	1.573	2.383	1.556
4	13	3.671	11.668	3.764	1.681	2.653	1.707
	14	3.892	12.041	3.857	1.792	2.878	1.782
	15	4.262	12.265	4.334	1.893	2.934	1.939
5	13	3.666	11.564	3.606	1.733	2.742	1.722
	14	3.938	12.182	3.898	1.836	2.858	1.899
	15	3.051	12.349	3.004	0.827	1.917	0.917

Table 2: Mean relative discrepancies (%) of solutions given by the six heuristics for the Fm | block | C_{max} case.

We applied also the heuristics to Taillard sets. The comparison is less significant: the number of instances for each set is reduced (10) and the instances were selected to retain only hard instances. The maximum number of accepted ties, without improvement, on NEDM-RCT was fixed to 5×n. Some results are shown on tables 3 and 4. The heading "best" corresponds to the best solution given by heuristics NEH2, Pour2 and N&M2; the heading "best+" corresponds to the best solution given by heuristics NEH2+, Pour2+ and N&M2+. In the $Fm \mid block \mid C_{max}$ case the optimum solution is, in general, unknown and the discrepancy is referred to the known best solution (normally given by LOMPEN_5).

n×m	*NEH2*	*POUR2*	*N&M2*	*best*	*NEH2+*	*POUR2+*	*N&M2+*	*best+*
20×5	2.105	9.577	2.327	2.105	0.922	1.027	0.927	0.658
20×10	3.819	12.268	3.347	3.039	2.295	3.347	1.863	1.633
50×5	0.428	4.010	0.804	0.402	0.267	0.648	0.403	0.190
50×10	4.508	11.606	4.239	3.911	2.376	3.206	2.142	1.914
100×5	0.405	2.318	0.292	0.265	0.221	0.501	0.255	0.179
100×10	1.686	6.566	1.807	1.576	1.143	1.309	1.225	1.023
200×10	0.982	3.968	1.009	0.861	0.767	0.942	0.553	0.497

Table 3: Mean relative discrepancies (%) of solutions given by heuristics for the $Fm \mid prmu \mid C_{max}$ case.

n×m	*NEH2*	*POUR2*	*N&M2*	*best*	*NEH2+*	*POUR2+*	*N&M2+*	*best+*
20×5	4.847	14.096	5.433	4.271	2.596	3.333	2.495	1.777
20×10	5.281	13.978	4.850	4.500	3.159	5.468	3.046	2.565
20×20	3.308	10.040	3.458	2.373	1.744	2.798	1.926	1.478
50×5	6.146	17.924	5.843	5.617	2.242	3.918	2.666	2.066
50×10	5.357	14.943	5.349	4.959	2.263	3.934	2.226	1.755
50×20	5.120	12.820	3.664	3.526	2.332	3.563	1.900	1.771
100×5	6.612	19.530	6.540	6.120	1.583	1.978	1.779	1.069
100×10	5.627	15.656	5.069	5.063	1.579	2.649	1.688	1.451
100×20	4.168	12.372	3.414	3.342	1.844	2.524	1.568	1.420

Table 4: Mean relative discrepancies (%), referred to the best known solution, of solutions given by heuristics for the $Fm \mid block \mid C_{max}$ case.

P_{OUR} and $P_{OUR}2$ are less efficient than NEH and N&M, but more computational time consuming. Ruiz and Maroto[3], reached the same conclusion comparing NEH and Pour heuristics (besides other 23) using the Taillard instance sets also. NEH and N&M are so efficient as NEH2 and N&M2. The true improvement of the solution value is produced by the local search improvement used (NEDM-RCT). We do not think very interesting to change NEH+ by N&M+, but, perhaps, on hard instances, it could be interesting to use NEH2+ or both, NEH2+ and N&M2+, retaining the best solution.

We do not think pertinent to include these comparisons and these comments in the paper.

[3] R. Ruiz and C. Maroto (2005) *op. cit*.

Referee 1: *The standard no wait would have been preferable to block.*

We think that **block** and **no wait** are different. We found in Michael Pinedo (1995) *Scheduling: Theory, Algorithms and Systems*, Prentice Hall
(p. 11) "**Blocking** (***block***). Blocking is a phenomenon that may occur in flow shops. If a flow shop has a limited buffer in between two successive machines, it may happen that when the buffer is full the upstream machine is not allowed to release a completed job. This phenomenon is known as *blocking*: the completed job has to remain on the upstream machine preventing, or blocking, said machines from working on another job …"
(p. 12) "**No-wait** (***nwt***). The *no-wait* requirement is another phenomenon which may occur in flow shops. Jobs are not allowed to wait between two successive machines. This implies that the starting time of a job at the first machine has to be delayed to ensure that job can go though the flow shop without having to wait for any machine …"
(p. 112) "… In contrast to the blocking case where jobs are *pushed* down the line by machines upstream that have completed their processing, in this case the jobs are actually *pulled* down the line by machines that have become idle … It is easy to see that $F2 \mid block \mid Cmax$ is equivalent to $F2 \mid nwt \mid Cmax$. However when there are more than two machines in series, the two problems are different …"

Referee 2: *The versions presented use sequential, and not parallel computation. The algorithm should not be described as a parallel algorithm.*

We think an algorithm and its computational implementation are two related but different things.

However, in order to avoid misunderstandings, the co-authors decided to write the adjective "double" instead of "parallel".

0) Two different versions of the LOMPEN algorithm. Could one of this versions have been used to solve all the considered instances? Comparisons.

We said, "The old (Vivó, 1994) code was unable to solve other instances of greater dimension". There are more than three versions of LOMPEN*:*
LOMPEN_0 (1992) the first prototype.
LOMPEN_1 (1993) was used on the Carulla and Ibañez engineer thesis and on the first experiments with the algorithm. We used a bounding procedure, published in 1983, similar to the proposed by Taillard (1993) and Nagano and Moccellin (2002), but more efficient. Only the $Fm \mid prmu \mid C_{max}$ case was considered.
LOMPEN_2 (1994) done by Vivó; it was similar to LOMPEN_1 but it used SPLIT function systematically.
LOMPEN_2.5 (1997) was a parallel implementation of LOMPEN_2 using PVM 8, done by Minguez and Roselló (engineer thesis directed by A. M. Coves).
LOMPEN_3 (1996) introduced the two machines bounding procedure. Some results on sets of 5000 instance were summarized in the documentation of several seminars and courses (2000).
LOMPEN_4 (2001) introduces the consideration of $Fm \mid block \mid C_{max}$ case. The main objective was to evaluate the performance of 10 heuristics (Palmer, trapezes, Gupta, Teixido_1 and NEH, without and with the local search procedure). The instances used were that of Experience_1 (additionally, we consider sets of 1000 instances each for m=3,4,5 and n=10,11,12). Some results were presented to OR43 (Bath, 4-6 September

2001). The chosen treatement for the two-machine bound implies that LOMPEN_4, in its present form, is limited to instances with m≤6. This is an important reason for a new code development.
LOMPEN_4.5 was a parallel implementation of LOMPEN_4, done by J. Pereira, 2002. The parallelism model used is based on MIMD (Multiple Instruction stream Multiple Data stream) a system with shared memory. Two processes deal with the direct and inverse instances at same time and they refresh the data for the other process in the shared memory. LOMPEN computations are run in a Work–Station Sun Ultra Enterprise 450, with four 400 MHz. processors Sparc (only two were used) and with 1 Gb. RAM and operative system Solaris 7.
LOMPEN_5 (2004), done by A. Alemán in his engineer thesis, introduces 5 heuristics, which are applied, with diverse local search procedures, to direct and inverse instances to obtain the initial best solution. Also, it introduces the *razzia* procedure. LOMPEN_5 can solve, and solved, all proposed instances. The mean CPU time for the $Fm \mid prmu \mid C_{max}$ case, with the sets used in Experience_1, is lower than the LOMPEN_4 time, but higher for the $Fm \mid block \mid C_{max}$ case.

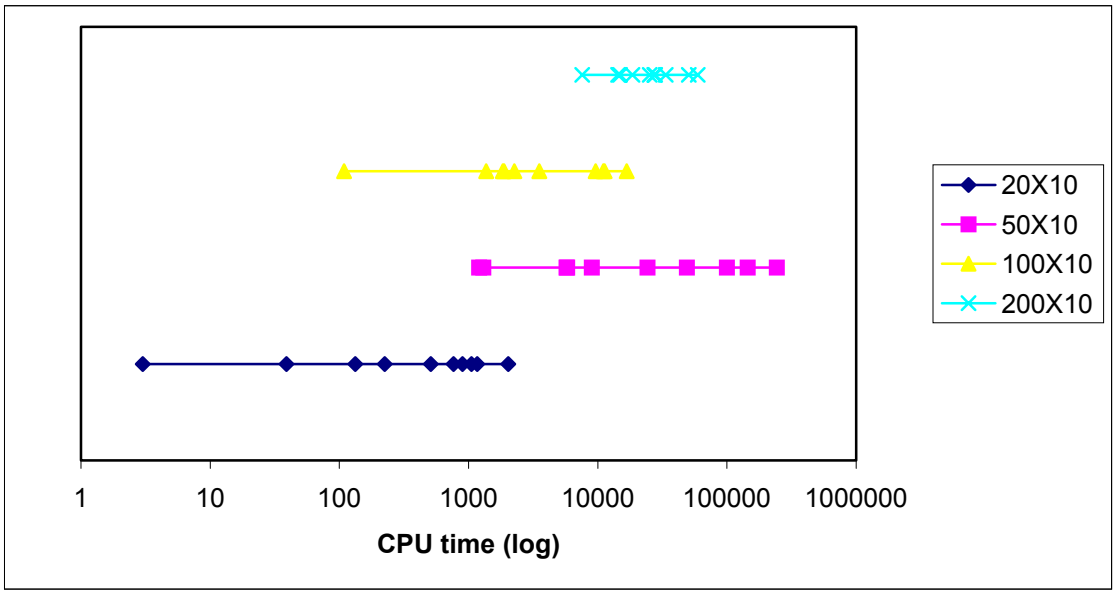
		Fm | prmu | Cmax		Fm | block | Cmax	
		LOMPEN_4	LOMPEN_5	LOMPEN_4	LOMPEN_5
m=3	n=13	0.048	0.004	0.705	1.085
	n=14	0.042	0.005	3.348	3.912
	n=15	0.036	0.004	7.772	12.504
m=4	n=13	0.135	0.007	2.026	2.275
	n=14	0.208	0.010	7.441	7.111
	n=15	0.290	0.009	28.321	33.129
m=5	n=13	1.208	0.023	3.639	4.828
	n=14	0.611	0.036	18.984	21.981
	n=15	1.250	0.050	70.737	94.487

Table 6: CPU time for LOMPEN_4 and LOMPEN_5.

Referee 2: *Page 20 (experience 2) and conclusion. The authors present two tables but the results are not discussed in the text.*

The behaviour of instance # 952 (n=13, m=5) on the $Fm \mid prmu \mid C_{max}$ case can indicate that 1000 random instances are not enough to deal with all the instance variability (the optimum of instance # 952 is 209 and initially the best bound is 201 and the best initial solution, given by NEH+, is 209).

The behaviour of the 20×50 Taillard instances suggests that the Taillard instances are not homogenously hard. We show the CPU time (on logarithmic scale) spent by LOMPEN_5 on the 40 Taillard instances with m=10 in the chart ($Fm \mid prmu \mid C_{max}$ case). To see it clearer, we use 4 different levels of horizontal values for n=20,50,100 or 200. The points corresponding to n=50 do not follow the general apparent distribution.



Referee 1: *Given the above comments and the limited scope of the article I recommend against publication.*

Obviously, we must accept the referee's opinion. Nevertheless, we think the article scope is not so limited. We show the impact of local search improvement on NEH and similar heuristics. We define a local search improvement procedure that takes into account the ties. We show the application of LOMPEN to Fm | block | C_{max} problem and the links between the direct and inverse explorations. We describe the differential behaviour of Fm | prmu | C_{max} and Fm | block | C_{max} problems, and the reasons that make, at the present level of knowledge, the second problem harder than the first. Our algorithm solved tail106 (m=20, n=200) in the Fm | prmu | C_{max} case and 7 instances (tail001, tail002, tail004, tail005, tail006, tail007, tail009) in the Fm | block | C_{max} case.

ANNEX 1: SOLUTIONS GIVEN BY ALGORITHMS

instance	NEH	POUR	N&M	NEH+	POUR+	N&M+	optimum
tail001	1286	1321	1286	1286	1297	1286	1278
tail002	1365	1412	1373	1365	1366	1366	1359
tail003	1132	1281	1132	1119	1098	1132	1081
tail004	1312	1459	1312	1302	1300	1304	1293
tail005	1305	1375	1305	1250	1250	1277	1235
tail006	1228	1313	1220	1224	1203	1210	1195
tail007	1270	1323	1267	1259	1249	1251	1234
tail008	1224	1407	1224	1206	1229	1214	1206
tail009	1265	1375	1295	1255	1259	1255	1230
tail010	1127	1236	1131	1108	1125	1127	1108

Table 7-a: n=20; m= 5 (Fm | prmu | C_{max} case)

instance	NEH2	POUR2	N&M2	NEH2+	POUR2+	N&M2+	optimum
tail001	1286	1321	1286	1286	1297	1286	1278
tail002	1365	1411	1370	1365	1366	1365	1359
tail003	1132	1281	1132	1098	1089	1088	1081
tail004	1308	1459	1312	1302	1300	1304	1293
tail005	1305	1360	1305	1250	1250	1243	1235
tail006	1210	1282	1210	1210	1203	1210	1195
tail007	1251	1323	1267	1251	1249	1246	1234
tail008	1221	1350	1224	1206	1206	1207	1206
tail009	1265	1349	1265	1255	1259	1255	1230
tail010	1127	1230	1127	1108	1125	1127	1108

Table 7-b: n=20; m= 5 (Fm | prmu | C_{max} case)

instance	NEH	POUR	N&M	NEH+	POUR+	N&M+	optimum
tail061	5514	5565	5495	5514	5499	5495	5493
tail062	5296	5403	5287	5284	5316	5284	5268
tail063	5196	5378	5206	5196	5211	5206	5175
tail064	5023	5122	5021	5023	5044	5021	5014
tail065	5261	5485	5306	5255	5255	5265	5250
tail066	5139	5248	5139	5139	5146	5139	5135
tail067	5284	5318	5281	5246	5284	5261	5246
tail068	5113	5339	5105	5101	5137	5105	5094
tail069	5489	5633	5487	5487	5504	5473	5448
tail070	5345	5358	5347	5328	5342	5342	5310

Table 8-a: n=100; m=5 (Fm | prmu | C_{max} case)

instance	NEH2	POUR2	N&M2	NEH2+	POUR2+	N&M2+	optimum
tail061	5514	5565	5495	5514	5499	5495	5493
tail062	5284	5403	5287	5284	5284	5284	5268
tail063	5196	5378	5206	5196	5211	5206	5175
tail064	5023	5122	5021	5023	5044	5021	5014
tail065	5261	5485	5261	5255	5255	5261	5250
tail066	5139	5248	5139	5139	5144	5139	5135
tail067	5283	5318	5261	5246	5284	5256	5246
tail068	5113	5238	5105	5100	5137	5098	5094
tail069	5489	5527	5465	5465	5495	5465	5448
tail070	5345	5358	5346	5328	5342	5342	5310

Table 8-b: n=100; m=5 (Fm | prmu | C_{max} case)

instance	NEH	POUR	N&M	NEH+	POUR+	N&M+	optimum
tail041	3142	3469	3106	3126	3126	3074	2991
tail042	3032	3341	2994	2953	2988	2975	2867
tail043	2998	3255	2953	2953	2969	2883	2839
tail044	3163	3336	3148	3122	3127	3120	3063
tail045	3149	3414	3108	3040	3073	3049	2976
tail046	3129	3393	3152	3128	3076	3114	3006
tail047	3207	3441	3271	3178	3237	3265	3093
tail048	3165	3281	3138	3058	3090	3082	3037
tail049	3073	3240	3050	2973	3025	2987	2897
tail050	3218	3400	3237	3195	3170	3163	3065

Table 9-a:n=50; m=10 (Fm | prmu | C_{max} case)

instance	NEH2	POUR2	N&M2	NEH2+	POUR2+	N&M2+	optimum
tail041	3142	3364	3106	3096	3126	3069	2991
tail042	3032	3226	2994	2953	2947	2975	2867
tail043	2981	3255	2953	2873	2969	2883	2839
tail044	3163	3336	3148	3112	3127	3099	3063
tail045	3135	3379	3108	3040	3031	3049	2976
tail046	3129	3364	3133	3128	3076	3083	3006
tail047	3207	3441	3271	3165	3237	3165	3093
tail048	3165	3281	3138	3058	3080	3055	3037
tail049	3017	3240	3012	2950	3025	2953	2897
tail050	3204	3400	3137	3169	3170	3140	3065

Table 9-b: n=50; m=10 (Fm | prmu | C_{max} case)

instance	NEH	POUR	N&M	NEH+	POUR+	N&M+	optimum
tail071	5867	6098	5892	5800	5814	5843	5770
tail072	5448	5783	5430	5430	5401	5384	5349
tail073	5767	6076	5785	5757	5702	5757	5676
tail074	6012	6306	5942	5905	5944	5885	5781
tail075	5620	6059	5608	5589	5658	5563	5467
tail076	5375	5678	5398	5335	5331	5331	5303
tail077	5668	5914	5708	5666	5711	5696	5595
tail078	5743	5945	5758	5713	5695	5695	5617
tail079	5979	6200	5946	5979	5979	5946	5871
tail080	5903	6240	5903	5903	5985	5903	5845

Table 10-a: n=100; m=10 (Fm | prmu | C_{max} case)

instance	NEH2	POUR2	N&M2	NEH2+	POUR2+	N&M2+	optimum
tail071	5867	6098	5892	5800	5814	5843	5770
tail072	5401	5728	5426	5384	5401	5384	5349
tail073	5759	6018	5774	5700	5702	5752	5676
tail074	5917	6306	5942	5879	5916	5882	5781
tail075	5620	5942	5608	5586	5658	5551	5467
tail076	5375	5678	5371	5328	5331	5331	5303
tail077	5668	5914	5708	5666	5711	5680	5595
tail078	5735	5945	5721	5695	5695	5695	5617
tail079	5979	6200	5946	5979	5979	5946	5871
tail080	5903	6140	5903	5903	5903	5903	5845

Table 10-b: n=100; m=10 (Fm | prmu | C_{max} case)

ANNEX 2: PAPER'S ABSTRACTS

Jean-Paul Watson, Laura Barbulescu, Adele E. Howe and L. Darrell Whitney (1999) "Algorithm performance and problem structure for flow-shop scheduling" *Proceedings 16th National Conference on AI*

El supuesto fundamental del artículo consiste en decir que los algoritmos se contrastan contra colecciones de ejemplares fabricados al azar mientras que los ejemplares del mundo real no son aleatorios[4]. Considera correlaciones entre tiempos de proceso respecto a máquina y respecto a pieza. Desea mostrar:

1) La eficiencia superior obtenida por un algoritmo en una colección de ejemplares popular en el campo de *test benchmarck* queda anulada cuando se aplica a una colección con un nivel modesto de aleatoriedad.
2) Los ejemplares reales con cierta estructura no generan un espacio topológico similar al asociado a ejemplares difíciles, lo que explica que los algoritmos que explotan la estructura, triunfadores en los primeros fallan en los segundos.

Total, *nihil novo sub sole*. Critican a Taillard (1993) por elegir ejemplares sin estructura y filtrarlos para quedarse con los difíciles. Construyen un "*PSP Generator*" y generan 6 colecciones, a partir de 3 dimensiones, con n=50,100,200 y m=20 y de la dualidad correlación en máquinas y correlación en piezas[5]. Para cada colección generan 100 ejemplares[6] y para obtener los tiempos de proceso utilizan leyes normales *ad hoc*.

Para la correlación en piezas consideran n distribuciones normales. Los tiempos de proceso de la pieza i se generan mediante la distribución normal i (suponemos, aunque no lo indican, que discretizan). Las desviaciones típo de las distribuciones normales se generan aleatoriamente mediante una ley uniforme entre 1 y 20 y las medias mediante una ley uniforme entre 35 y $35+\alpha \cdot H$, donde H (llamada *canonical width*) es igual a cuatro veces la suma de las n desviaciones tipo. Se supone que al crecer α se reduce la posibilidad de solapamiento de las distribuciones normales (claramente si $\alpha=0$ todas las distribuciones normales tienen la misma media). α adopta 10 valores entre 0.1 y 1, con paso 0.1.

Para la correlación en máquinas se procede de forma análoga. Parece deducirse que en *PSP Generator* los valores numéricos indicados están parametrizados y pueden modificarse.

Los algoritmos utilizados son aparentemente siete (dicho número es el que figura en los resultados) basados en tres metodologías de búsqueda:

1. *path relinking*: Se trata de una estrategia general de búsqueda en la cual el espacio de búsqueda se explora intentando encontrar óptimos adicionales

[4] "*However, real-world problems are not random*"
[5] No está muy claro, pero es lo que parece por los resultados.
[6] Aunque podrían ser 1000, si son 100 para cada colección y valor de α. Este aspecto no queda claro ya que en la cuarta página dicen "*Algorithm performances were measured on six problem classes ... For each problem class, we generate 100 problem instances ...*" y luego "*… we obtained a summary measure of algorithm performance at each level of α for each problem class by computing the average percent above best for the 100 problems*". Si cada clase consta de 100 ejemplares corresponden 10 para cada clase y valor de α.

próximos a dos óptimos locales conocidos[7] (Whitley, L. D., 1989 "The GENITOR algorithm and selective preasure: Why rank based allocation or reproductive trials is best" *Proceedings of the Third International Conference on Genetic Algorithms*).

2. *incremental construction*: Son heurísticas familiares tales como NEH, DDS (Walsh, T., 1996, "Deph-bounded discrepancy search" *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*), LDS (Harvey, W. D. and Ginsberg, M. L., 1995 "Limited discrepancy search" *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*), HBSS (Bresina, J. L., 1996 "Heuristic-biased stochastic sampling" *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*)
3. *iterative sampling*: Se caracterizan por muestrear iterativamente. NEH-RS (NEH con *random starts*) consiste en ordenar las piezas al azar en la primera fase. ITSAMPLS (*iterative random sampling with local search*) consiste en generar una permutación aleatoriamente y aplicarle búsqueda local. La búsqueda local no utiliza el procedimiento de intercambio sino el de intercalación y desplazamiento (Reeves, C. R. and Yamada, T. 1998 "Genetic algorithms, path relinking, and the flow shop sequencing problem" *Evolutionary Computation* **6**, 45-60). Aparentemente también consideran un octavo procedimiento, la generación de permutaciones al azar, que queda fuera de las comparaciones ya que siempre es el peor[8].

Definen, como unidad de tiempo para los algoritmos basados en NEH, el del bucle de la fase 2 del NEH[9]. A dicho tiempo lo llaman "evaluación" y asignan 100K de tales evaluaciones a todos los algoritmos basados en NEH. Para el muestreo aleatorio generan 100K soluciones. PATHRELINK se limita a 10^5 evaluaciones[10]. ITSAMPLS limitan el número de evaluaciones[11] a 100K.

Para cada algoritmo registran el mejor F_{max} obtenido para cada ejemplar. Dado que el F_{max} óptimo es desconocido miden la ineficiencia del algoritmo mediante el porcentaje por encima de la mejor solución hallada por alguno de los algoritmos. Finalmente obtienen una síntesis del rendimiento del algoritmo para cada nivel de α calculando el valor medio del porcentaje.

Correlación en máquinas

Los resultados, como cabría esperar, indican que los algoritmos estocásticos (ITSAMPLS, NEH-RS y HBSS) vencen a los deterministas (NEH, LDS y DDS). Para grandes dimensiones ITSAMPLS cae. La sorpresa la da PATHRELINK ya que no se comporta aceptablemente.

---

[7] "*At the highest level, pathrelink is a steady-state genetic algorithm*"

[8] No deja de ser lógico. El procedimiento en forma natural, es la primera fase de ITSAMPLS y por tanto sus resultados son peores que los de dicho procedimiento ya que han pasado por la mejora local.

[9] Suponemos que es "*Insert the k-th job into the location in the partial schedule, among k possible, which minimizes the partial makespan*"

[10] No sabemos si son las evaluaciones definidas antes, aunque PATHRELINK no es un método basado en NEH. El texto anterior a la indicación del límite dice "*Each local search or path projection involves 1000 steps, each requiring an evaluation*"

[11] Tenemos la misma duda que antes, aunque tampoco ITSAMPLS es un método basado en NEH.

Correlación en piezas

Pasa más o menos lo mismo. NEH-RS es el gran vencedor, aunque aquí PATHRELINK le sigue.

El artículo continúa con una digresión sobre qué significa un ejemplar duro, qué es la estructura de un ejemplar y analizan las distancias entre óptimos locales en los ejemplares de Taillard y en los ejemplares con correlación.

Concluyen que PATHRELINK, que es maravilloso en los ejemplares de Taillard, aquí no va tan bien, en cambio una cosa tan simple como NEH-RS va muy bien.

El artículo incluye 8 figuras (6 histogramas y 2 nubes de puntos).

Comentarios: El artículo proviene del campo de AI y se nota. No indica las razones por las que la aleatoriedad tal como la consideran se adapta más al mundo real que Taillard. No hay ningún análisis de ejemplares reales, dejando a un lado la consideración, puramente anecdótica, que ciertas operaciones pueden depender del tamaño de la pieza[12] y otras no. Se centran en los ejemplares con m=20. Discuten lo que ocurre al variar n (entre ciertos límites ya que sólo consideran tres valores) y el parámetro α, pero no les preocupa lo que puede pasar al variar m. La relación de α con el solape de las distribuciones es aceptable, pero las medias se determinan aleatoriamente, por tanto el valor $\alpha=0$ garantiza el solape, pero $\alpha=1$, por sí sólo, no garantiza la dispersión. Sobre todo si el número de ejemplares es 10, la realidad puede diferir de los deseos.

La elección de las heurísticas es, por lo menos, curiosa. Nos tememos que el experimento carece de *fair play*, no hablan estrictamente de tiempos de proceso. Nos cuesta mucho creer, por ejemplo, que NEH consuma el mismo tiempo de cálculo que NEH-RS; sólo es coherente comparar dos procedimientos tal como lo hacen si ambos consumen los mismos recursos. En caso contrario es honesto indicar el consumo de recursos de cada uno. La unidad elegida a partir de NEH es ambigua: los cálculos no son los mismos según el valor de k ya que aproximadamente son del orden de k^2 (k evaluaciones de F_{max} con k piezas), que es variable con k y tiene un máximo n^2 dependiente de n. Suponiendo que eligen un valor medio, podemos suponer para NEH-RS un tiempo concedido proporcional aproximadamente a $10^5 \times n^2/3$ mientras que para NEH normal sería $n \times (n+1) \times (2 \cdot n+1)/6 \approx n^3/3$. Por tanto el 10^5 no se compensa[13] ni con n=200. ¿Consumen 10^5 evaluaciones de PATHRELINK o ITSAMPLS lo mismo que 10^5 de NEH-RS?
Ya que utilizan un procedimiento de mejora local, ¿por qué no lo utilizan con la solución de NEH?

Si de verdad las escalas de las desviaciones relativas respecto a la mejor solución son porcentajes, las desviaciones máximas de los algoritmos, sin incluir el de muestreo aleatorio, son insignificantes (0.004, 0.002 y 0.016 para correlación en máquinas, 0.0008, 0.0009 y 0.0032 para correlación en piezas). Posiblemente el artículo llega a demostrar el supuesto 1 propuesto, pero dudamos mucho que haya demostrado el 2.

[12] Se refieren a paneles de circuitos (*circuit boards)*

[13] Lo cual es lógico ya que la gracia de NEH-RS es realizar muchas construcciones de solución y guardar la mejor. ¿Es honesto comparar con NEH que sólo hace una?

Hamid Davoud Pour (2001) "A new heuristic for the n-job, m-machine flow-shop problem" *Production Planning & Control* **12** (7) 648-653

Aunque el artículo habla siempre del problema $n/m/F/F_{max}$ entendemos que trata el $n/m/P/F_{max}$.

El procedimiento propuesto construye progresivamente una permutación. En un momento dado sea σ la permutación parcial de r piezas ya construida, **J** el conjunto de las piezas ya secuenciadas en σ y $\overline{\mathbf{J}}$ el conjunto de las no secuenciadas. Se prueban todas las piezas[14] $i \in \overline{\mathbf{J}}$ en la posición r+1, construyendo una permutación de n piezas colocando detrás de i todas las piezas de $\overline{\mathbf{J}}-\{i\}$ en cierto orden. Precisamente la determinación de este orden es el aspecto más complejo del procedimiento y lo describimos más adelante. Se determina F_{max} para cada posible pieza i en la posición r+1 y aquella que proporciona el menor valor se asigna definitivamente a dicha posición con lo que la nueva permutación parcial σ*i tiene r+1 posiciones definidas. No se indica cómo se resuelven los empates.

Las operaciones de las piezas de $\overline{\mathbf{J}}-\{i\}$ se ordenan independientemente en cada máquina por $p_{j,i}$ creciente y se determina el instante $\overline{c}_{j,h}$ en que cada pieza $h \in \overline{\mathbf{J}}-\{i\}$ terminaría su operación en la máquina j en el orden establecido y <u>comenzando la primera operación</u> en dicha máquina en el instante 0. Se determina $\overline{C}_h = \sum_{j=1}^{m} \overline{c}_{j,h}$. El orden de las piezas es el de $\overline{C}_h$ creciente. No se indica cómo resolver los empates en ninguna de las dos ordenaciones.

Presenta un ejemplo numérico con n=3 y m=3.

Para contrastar el algoritmo adopta otras tres heurísticas: CDS, Palmer, y NEH. La fórmulación de CDS no es correcta (probablemente error tipográfico) y en Palmer utiliza un cambio de signo y una división por 2 totalmente innecesarios.

La base experimental está formada por 2000 ejemplares, 100 de cada una de las 20 parejas (n,m) elegidas, distribuidos de la siguiente forma:

- 800 pequeños de 8 parejas (n,m) con $2 \le n \le 8$, $2 \le m \le 18$, $4 \le n \times m \le 80$
- 600 medianos de 6 parejas (n,m) con $10 \le n \le 30$, $2 \le m \le 50$, $39 \le n \times m \le 900$
- 600 grande de 6 parejas (n,m) con $40 \le n \le 65$, $34 \le m \le 65$, $1496 \le n \times m \le 3625$

Entre los pequeños aparecen las parejas (2,2) y (4,2) y entre los medianos (13,3). Aparentemente la clasificación está basada en el valor de n.

Los tiempos de proceso se han obtenido aleatoriamente de una distribución uniforme entre 1 y 99.

Como indicadores ha elegido:

[14] No entendemos la insistencia del autor en elegir la pieza a probar al azar.

- el índice de eficiencia: EI = 100×H/F, donde H es el valor del parámetro para el método propuesto (que denomina Hamid) y F para el algoritmo con el que contrasta (CDS, P o NEH) (aunque en el texto aparece la formulación indicada, a partir de los comentarios y los valores de la tabla suponemos que EI = H/F)

- el error relativo: RE = 100×(F – H)/H con el mismo significado anterior, aunque aquí el parámetro es claramente F_{max}.

La comparación con NEH es sorprendente. Los valores de síntesis sobre los 2000 ejemplares son RE = 5.8 e EI = 0.90; por tanto EI no se refiere a F_{max} ya que aproximadamente debería cumplirse RE = $100\times(EI^{-1} - 1)$

En un histograma (figura 1) muestra que F_{max}, para todas las parejas (m,n), es prácticamente igual con CDS y Palmer, y muy parecido con NEH y Hamid, existiendo una ligera ventaja de este último para los ejemplares grandes. Curiosamente en el gráfico la escala de F_{max} no sobrepasa el valor 1000 cuando para ejemplares con n=65 y m=65 serían esperables valores de F_{max} del orden de 6450 y superiores. Los resultados (RE y EI) se presentan para ejemplares pequeños, medianos y grandes, así como para el conjunto. No existe pues discusión del comportamiento de la heurística al variar el número de máquinas (que según nuestra experiencia puede influir decisivamente). Un histograma muestra el tiempo de cálculo de las diferentes heurísticas para cada una de las 20 colecciones definidas por la pareja (n,m); no nos es posible emitir un juicio comparativo sobre el tiempo de cálculo de Hamid respecto NEH (los valores se encuentran demasiado próximos[15]).

Concluye el autor que el número de máquinas no tiene efecto en el número de iteraciones a realizar y por tanto que su algoritmo es adecuado para gran número de máquinas. También insinúa que muchos cálculos pueden obviarse por haberse realizado con anterioridad, lo que implica cierta disciplina en guardarlos.

El artículo está acompañado de dos figuras (el histograma de F_{max} y el del tiempo de cálculo) y el ejemplo numérico. Las referencias comprenden 5 artículos, el más antiguo de 1965 (Palmer) y los dos más modernos de 1989.

[15] Lo que no está de acuerdo a nuestra experiencia,

M. S. Nagano and J. V. Moccellin (2002) "A hight quality solution constructive heuristic for flow shop sequencing" *Journal of Operational Research Society* **53**, 1374-1379

La heurística NEH se compone de dos fases: en la primera se ordenan las piezas de cierta manera mientras que en la segunda se construye una permutación mediante un procedimiento de incorporación de las piezas no consideradas en una secuencia ya formada con posible intercalación. El orden inicial en NEH es el de carga de la pieza ($JL_i = \Sigma_j\ p_{j,i}$) no creciente. La heurística N&M substituye JL_i por $JL_i - \max_h\{BT_{h,i}\}$ y mantiene el resto. $BT_{h,i}$ es el tiempo total de bloqueo de la máquinas si no hubiese pulmones y se procesaran únicamente dos piezas, h e i, en dicho orden[16]. No explica como se resuelven los empates.

Para demostrar que el orden inicial es importante en la heurística NEH desarrollan el "doble" NEH, es decir una vez aplicado NEH, vuelven a aplicar la segunda fase al orden resultante. En su experimento esta segunda aplicación empeora la solución. No entendemos por qué no aplican la segunda fase al orden JL_i no decreciente, al dado por una heurística directa (Palmer, Gupta, etc.) o incluso a un orden obtenido aleatoriamente.

Para la comparación NEH versus N&M utilizan dos índices. El porcentaje de éxitos, número de veces que la solución de una heurística no es peor que la de la otra (incluyendo, por tanto, los empates) y una especie de discrepancia determinada respecto a una cota y no respecto al óptimo. La cota, que atribuyen a Taillard (1993), coincide aproximadamente con lo que utilizábamos en LOMPEN_1 y LOMPEN_2. Dicha cota es el máximo de la tradicional cota longitudinal (ya usada por Lomnicki), sin imponer que la primera y la última pieza sean distintas y una especie de cota transversal. Como cota transversal utilizan JL_i sin añadir, como sería lógico[17], $\sum_{h\neq i} \max\{p_{1,h}, p_{m,h}\}$.

Presentan un ejemplo numérico n=10 y m=4, con el que obtienen un F_{max} de 66 con NEH, de 63 con N&M y una cota de 62 (es fácil obtener soluciones de valor 62).

La base experimental está formada 5700 ejemplares. Los valores de m son 4, 7 y 10, los de n toman la forma 5×k donde k es un entero entre 2 y 20; de cada combinación se han generado 100 ejemplares. Los tiempos de proceso se han tomado aleatoriamente de una ley uniforme entre 1 y 10.

Llegan a la conclusión que el porcentaje de éxitos es mayor con N&M que con NEH (aunque la figura es engañosa ya que el eje de abscisas se sitúa a la altura del 60 %). Sin embargo tanto la media como la desviación típica de la discrepancia son prácticamente idénticas. Al contrastar esto último con el porcentaje de éxitos una explicación que se nos ocurre es que cuando N&M es mejor que NEH lo es por poco y cuando es peor lo es por mucho. Los tiempos de cálculo son ligeramente superiores en N&M. Los autores

[16] En el artículo no lo definen así sino como "*lower bound for total waiting time for job i between the end of its operation on each machine and the beginning of the operation on the successive machine, when job h immediately precedes job i*". Tampoco está incluido en el artículo como calcular fácilmente dicho valor en el ejemplar inverso, que es lo que hemos hecho nosotros.

[17] R. Companys (1983) *Problema del Taller Mecánico* CPDA-ETSEIB

concluyen "*computational results lead us to an experimental conclusion that NEH is not superior to the N&M heuristic*".

El artículo está acompañado de 6 figuras (2 diagramas de Gantt y 4 histogramas), y el ejemplo numérico detallado.

Comentarios: Podemos formular dos críticas,

- El número de máquinas no es considerado en los resultados, los histogramas están dibujados en función de las piezas. Suponemos que los valores representados (tanto respecto a la solución como respecto al tiempo) son los medios de los 300 ejemplares, 100 con m=4, 100 con m=7 y 100 con m=10.

- Existen algunos problemas de nomenclatura, por ejemplo $\Sigma_j\ p_{j,i}$ es P_i en la página 1375, un poco más abajo P_{i} (que suponemos un error tipográfico) y $\Sigma\ TP_i$ en la página 1376.