Belgian Journal of Operations Research, Statistics and Computer Science Vol. 34 (1)

Minimization of the number of tool switches on a flexible manufacturing machine

Jean–Pierre Follonier

Ecole Polytechnique Fédérale de Lausanne
Département de Mathématiques
1015 Lausanne, Switzerland

Abstract

In this paper we consider the problem of sequencing a set of jobs on a single machine having a limited capacity tool magazine. Each job requires a subset of tools for its processing. If these tools are not present in the magazine, one or more tools must be removed and replaced by others. The problem is to find the best job sequence such that the number of tool switches is minimized. This NP–hard problem has been proposed by Tang and Denardo.

In this article we propose two new heuristics. The first one is based on an insertion method and the second one is an adaptation of the general tabu search techniques. These two new heuristics are compared with the previous ones and tested on problems of different sizes. They give better results while requiring less computation time.

Keywords : flexible manufacturing systems, tool management, sequencing, heuristics, tabu search

1. Introduction

The problem under investigation involves the processing of a serie of jobs that have been assigned to a numerically-controlled machine equipped with an automatic tool interchange device. Each job requires a specific set of tools. These tools must be placed on the limited capacity tool magazine of the machine before the job can be processed. If the requisite tools are not on the machine magazine, then one or more tool switches need to be performed; that is some tools must be transferred into the tool storage area and replaced by some others. The cost of this operation is proportional to the number of switches. It includes retrieval from storage, transportation, loading and calibration. The problem is to determine the processing sequence of the jobs and the tools to switch on the machine before each job is processed. We suppose that at the startup C tools are inserted into the machine magazine and at the end of the job sequence, the C tools are removed. So no switches are considered at startup and at the end.

The problem of minimizing the number of tool switches was first proposed by Tang and Denardo (1988). Later, it was revisited by several authors; Bard (1988) formulated the problem as a nonlinear integer program and solved it with a dual-based relaxation heuristic. Crama et al. (1991) established basic results concerning the computational complexity of the problem and implemented several heuristics.

In this paper, we describe some of the heuristics developed by these people and compare their results with those given by our two new heuristics. The first one is based on an insertion method and the second one on the well-known tabu search technique. They both give better results than the ones developed previously and need very little CPU time. Furthermore the tabu search heuristic allows us to get an accurate estimation of the optimum solution of the problem, if we do not care about the required amount of CPU time.

Section 2 contains a mathematical description of the problem and its possible decomposition into two subproblems : a tooling problem and a sequencing problem. In Section 3, the heuristics used to solve the second subproblem are presented. Section 4 reviews our computational results and compares the efficiency of the older heuristics with the new ones. Finally, this paper ends with a conclusion in Section 5.

2. Problem description

2.1 Mathematical formulation

Let N be the number of jobs to be processed on a single machine, and M be the number of tools required to process the whole set of jobs. We suppose that each tool occupies only one tool slot on the tool magazine and that the tool magazine has exactly C tool slots. We assume that no job requires more than C tools (otherwise the problem is unfeasible) and that there is no tool sharing with other machines.

Let A be a NxM matrix where each row represents a job and each column a tool. An element of A, a_{ij}, is equal to 1 if the processing of job i requires tool j and 0 otherwise. So A_i, the i-th row of the matrix A, denotes the tool requirement vector of job i.

The underlying problem – we will name it the *switching problem* – is to determine the job sequence and the corresponding tool loading that minimizes the number of tool switches. Therefore we introduce the following 0-1 variables. Let x_{in} be 1 if the job i is assigned to position n in the sequence and $x_{in} = 0$ otherwise (i, n = 1, 2, ..., N). Let W be a NxM matrix, where each row represents an instant of the sequence and each column a tool. An element of W, w_{nj}, is equal to 1 if tool j is on the magazine at instant n, and 0 otherwise. So W_n, the n-th row of W, describes the tools on the machine at instant n. We assume that at the startup C tools are inserted into the magazine. Since there is no point in removing a tool from the magazine unless we have to insert an other one, the tool magazine contains exactly C tools at any time.

With the above notation and assumptions, the constraints and the objective function of the problem can be expressed as follows :

$$\min \sum_{n=1}^{N-1} \sum_{j=1}^{M} w_{n+1,j} \quad (1 - w_{nj}) \qquad (0)$$

$$\sum_{i=1}^{N} x_{in} = 1 \qquad \text{for } n = 1, \ldots, N \qquad (1)$$

$$\sum_{n=1}^{N} x_{in} = 1 \qquad \text{for } i = 1, \ldots, N \qquad (2)$$

$$\sum_{j=1}^{M} w_{nj} = C \qquad \text{for } n = 1, \ldots, N \qquad (3)$$

$$a_{ij}\, x_{in} = w_{nj} \qquad \text{for } i, n = 1, \ldots, N \text{ and } j = 1, \ldots, M \qquad (4)$$

$$x_{in}, w_{nj} \in \{0,1\} \qquad \text{for } i, n = 1, \ldots, N \text{ and } j = 1, \ldots, M \qquad (5)$$

Constraints (1) and (2) ensure that each job is assigned to exactly one position in the sequence. Constraint (3) fixes the number of tools in the magazine at C at any instant. Constraint (4) guaranties that if job i requires tool j and is placed in the n-th position of the sequence, then tool j must be in the magazine at this moment. The objective function (0) simply counts the number of tool switches. Tang and Denardo tried to linearize it in order to apply different standard integer programming codes. The results they obtained were somewhat disappointing. For small problems (N=10, M=9, C=4), the programs needed huge amounts of CPU time and did not find the optimal solution. So they resorted to a heuristic.

The tool switching problem can be decomposed naturally into two subproblems : the tool replacement problem and the scheduling problem.

2.2 The tool replacement problem

In this case, we consider a given job sequence, i.e. x_{in} are fixed and satisfy constraints (1) to (5). Then the problem consists of finding the best tool replacement policy, that is determining which tools are to be placed on the machine at each instant. This can be done optimally using the "Keep Tool Needed Soonest" (KTNS) policy (see Tang and Denardo, 1988), which has the following properties :

a) At any instant insert all the tools that are required for the current job.

b) If there are no vacant slots when tools need to be inserted on the magazine, then keep the tools that are needed the soonest.

Here is a possible implementation of the KTNS policy; T(j,n) denotes the set of all instants, at or after instant n, at which tool j is needed and L(j,n) is the first instant at or after which tool j is needed; L(j,n) is equal to N+1 if tool j is no more used after instant n.

Step 0: For $j = 1, ..., M$ and $n = 1, ..., N$, determine
$T(j,n) = \{k : \exists\, i,\ k \geq n,\ a_{ij} = 1 \text{ and } x_{ik} = 1\} \cup \{N+1\}$
$L(j,n) = \min \{m : m \in T(j,n)\}$
Set $w_{ij} := 0 \quad \forall\ i,j$

Step 1: Set $w_{1j} := 1$ for C values having minimal values of L(j,1)
Set $n := 2$

Step 2: If $n \leq N$
then set $W_n := W_{n-1}$
else STOP.

Step 3: If $[L(j,n) = n] \Leftrightarrow [W_{nj} = 1]\ \forall\ j$
then set $n := n+1$ and go to step 2.

Step 4: Find j such that $L(j,n) = n$ and $w_{nj} = 0$; set $w_{nj} := 1$ and $w_{nk} := 0$ for a k that maximize $L(p,n)$ over $\{p : w_{np} = 1\}$; go to step 2.

The number of tool switches entailed by the job sequence $\mathcal{J}$, can then easily be calculated; it is equal to the following sum :

$$TS(\mathcal{J}) = \sum_{n=1}^{N-1} \sum_{j=1}^{M} \max \{ 0,\ (w_{n+1,j} - w_{n,j}) \}$$

Theorem 1 : Each KTNS policy minimizes the total number of tool switches for the tool replacement problem.

Proof : see Tang and Denardo (1988).

This theorem shows that the tool replacement subproblem can be solved easily and quickly. Let us just point out that the KTNS policy has complexity O(MN).

2.3 The scheduling problem

Theorem 1 indicates how to find an optimal tooling for a given job sequence. So the scheduling problem amounts of finding the optimal job schedule, minimizing the number of tool switches, among the N! possible sequences.

In some cases, the scheduling problem may be reduced. If the set of tools required to process a job J_2 is a subset of the tools required to process a job J_1 (let us denote this by $A_{J_2} \leq A_{J_1}$), then processing J_2 immediately after J_1 entails no additional tool switches. More formally, let $F(i) = \{k : k \neq i, A_k \leq A_i\}$. Then the jobs in F_i may be processed in any order immediately after job i, without having to load any additional tools.

The scheduling problem is NP-Hard as shown by Tang and Denardo (1988) and Crama et al. (1991). Therefore the use of a heuristic approach is essential. The next Section presents some of them.

3. Heuristics for the scheduling problem

3.1 Traveling salesman heuristics

Tang and Denardo (1988) transformed the problem of finding a job schedule into a traveling salesman problem. In fact, each schedule can be seen as an Hamiltonian path in a complete graph G, in which the vertices represent the N jobs. The length of an edge (i,j) denoted LB(i,j), is equal to the smallest possible number of tool switches needed if job j is positioned immediately after job i in the sequence, without taking into account the tooling decisions before job i or after job j. LB(i,j) can be defined as follows :

$$LB(i,j) = \max \left\{0, \left(\sum_{k=1}^{M} a_{jk} - \sum_{k=1}^{M} a_{ik}\, a_{jk}\right) - \left(C - \sum_{k=1}^{M} a_{ik}\right)\right\}$$

The first part counts the number of tools required by job j but not by job i, and the second part is equal to the number of tool slots available on the machine after the processing of job i. Of course, we have LB(i,j) = LB(j,i).

The job sequence is obtained by determining a good Hamiltonian path in the graph G, by using one of the two following greedy procedures :

3.1.1 Shortest Edge heuristic (SE)

This heuristic was applied by Tang and Denardo (1988). It constructs the path by choosing sequentially the shortest edges in the graph :

Step 1: Pick the shortest arc on the graph and put it on the list $\mathcal{L}$. Remove that arc from the graph.

Step 2: If $\mathcal{L}$ consists of N-1 arcs, then go to step 5.

Step 3: Pick the shortest arc on the remaining graph and remove it from the graph.

Step 4: Check the arc picked in Step 3 with all the arcs on the list $\mathcal{L}$. If that arc forms a tree (a node with degree 3) or a cycle with some of the arcs on the list $\mathcal{L}$, then go to step 3; otherwise, add that arc to the list $\mathcal{L}$ and go to step 2.

Step 5: Deduce from $\mathcal{L}$ the job sequence $\mathcal{J}$ corresponding to the Hamiltonian path.

The complexity of the Shortest Edge heuristic is $O(N^2 \log N)$.

3.1.2 Farthest Insertion heuristic (FI)

This heuristic is used to obtain an initial solution for the traveling salesman problem. The following variation was implemented (see Golden and Stewart (1985), pg. 226) :

Step 1: Start with the partial job sequence $\mathcal{J} := \{1\}$.
Let $\mathcal{Q} := \{2, 3, ..., N\}$ be the set of jobs not sequenced yet.

Step 2: Find job $k \in \mathcal{Q}$ and job $l \in \mathcal{J}$ such as $LB(l,k) = \max_j\{\min_i LB(i,j)\}$, where $i \in \mathcal{J}$ and $j \in \mathcal{Q}$. Insert job k in the best position in the partial sequence $\mathcal{J}$.
$\mathcal{Q} := \mathcal{Q} \setminus \{k\}$

Step 3: If $\mathcal{Q} \neq \emptyset$ then go to step 2.

This implementation of the Farthest Insertion heuristic has complexity $O(MN^4)$.

3.2 Simple Greedy heuristic (SG)

This greedy heuristic was proposed by Crama and al. (1991); the sequence $\mathcal{J}$ is constructed job by job:

Step 1: Start with the partial job sequence $\mathcal{J} := \{1\}$.
Let $\mathcal{Q} := \{2, 3, ..., N\}$ be the set of jobs not sequenced yet.

Step 2: Choose i such as $TS(\mathcal{J} \oplus i) = \min \{\mathcal{J} \oplus j, j \in \mathcal{Q}\}$ where $\mathcal{J} \oplus j$ denotes the partial job sequence $\mathcal{J}$ followed by the job j.
$\mathcal{J} := \mathcal{J} \oplus i$ and $\mathcal{Q} := \mathcal{Q} \setminus \{i\}$

Step 3: If $\mathcal{Q} \neq \emptyset$ then go to step 2.

At step (b), TS($\mathcal{J} \oplus j$) is evaluated by using the KTNS policy. Therefore the complexity of the Simple Greedy heuristic is $O(MN^3)$.

3.3 Multiple-Start Greedy (MSG)

This heuristic runs N times the Simple Greedy heuristic, starting with every one of N jobs and retains the best complete sequence found (see Crama et al. (1991)). Hence its complexity is $O(MN^4)$.

3.4 Best Position Insertion heuristic (BPI)

The above greedy heuristics (SG and MSG) try to complete a partial sequence by adding at the end the job that minimizes the number of tool switches entailed by the resulting partial job sequence. The Best Position Insertion heuristic sorts the jobs according to a certain rule. Starting from the partial job sequence containing only the first job, it inserts the others, one after the other, at the best position in the sequence.

Step 1: Sort the jobs by weight decreasing order : $c_1 \geq c_2 \geq ... \geq c_N$

Step 2: $\mathcal{J}$:= {1} and Q := {2, 3, ..., N}

Step 3: For i:=2 to N do
Choose the best sequence S among the following ones :
{ (i,1,2,...,i-1), (1,i,2,...,i-1), (1,2,i,3,...,i-1), ... ,
(1,2,...,i-1,i) }
$\mathcal{J}$:= S and Q := $Q \setminus \{i\}$

Different weight functions were experimented :

1) c_i := uniform(0,1), i.e. the jobs are ordered randomly.

2) c_i := number of tools required to process job i

3) c_i := – (number of tools required to process job i)

4) each tool j receives a weight e_j equal to the number of jobs requiring it and the weight of a job i is obtained by summing the weights of the tools it requires :

$$c_i := \sum_{j=1}^{M} a_{ij}\, e_j = \sum_{j=1}^{M} a_{ij} \sum_{k=1}^{N} a_{kj}$$

These weight functions were experimented on different problem instances which are described in Section 4. The average number of tool switches of ten instances are presented in Table 1. The best results are given by the second and the forth weight functions. Finally the second function was selected because it gives the best results and is the simplest one. So the Best Position Insertion heuristic inserts the jobs into the partial sequence starting with those requiring the most tools. These jobs have to be treated carefully since they are likely to lead to large increase in the number of to tool switches. The complexity of the Best Position Insertion heuristic is $O(MN^3)$ like the Simple Greedy one.

Weight function	Problem 1 (10,10)	Problem 2 (15,20)	Problem 3 (30,40)	Problem 4 (60,90)
1	7	16	78	184
2	7	15	72	177
3	8	17	83	192
4	7	15	73	177

Table 1: Comparison of the results obtained by BPI with different weight functions

3.5 Tabu Search heuristic (Tabu)

3.5.1 Sketch of tabu search techniques

Tabu search techniques are mainly used to solve combinatorial optimization problems. These methods suggested by Glover (1989, 1990) can be sketched as follows :

> *starting from an initial feasible solution, at each step we choose a move to a neighboring solution in such a way that we move towards a solution giving hopefully the optimum value of some objective function f.*

For this purpose, each solution is represented by a point in some space and we define a neighborhood $N(s)$ of each point s ($s \notin N(s)$). The basic step of the procedure consists in starting from a feasible point s, generating a subset V^* of $N(s)$ and then moving to a neighbor s^* in V^*, usually to the one that optimizes $f(s)$ over V^*.

Up to this point, this is close to a local improvement technique except the fact that a move to a solution s^* worse than the current solution s may be accepted.

The interesting feature of the tabu search technique is the tabu status that can be given to certain moves in order to rule them out. The objective of this status is to exclude moves which would bring the algorithm back where it was at some previous iteration and keep it trapped in a local minimum. So at each move $s \rightarrow s^*$, the opposite move $s^* \rightarrow s$ is declared tabu; this status is attributed only for *tabulength* iterations to avoid cycling to some extent. However it may also act too violently and prevent us from including in V^* solutions which were not visited earlier. Therefore, for relaxing the action of the tabu status, an aspiration function A is introduced : a solution s' in $N(s)$ which would be forbidden because of the tabu status can nevertheless be included into V^* if it is associated with a threshold value (given by A) that is greater than $f(s')$.

In conclusion, the basic step consists in moving from a point s to the best non tabu neighbor s^*. At each step the best solution s^+ found is stored and updated; so are the tabu status and the aspiration function A.

A stopping rule must also be defined : in general, we may give a number *nbmax* limiting the number of consecutive iterations which can be performed without providing any improvement of the best value of the objective function. An estimation f' of the minimum value of the objective

function f can also be used. As soon as the solution is close enough to *f'* or *nbmax* steps without improvement have been performed, the whole procedure stops.

Tabu search techniques have been applied successfully to solve many different problems such as graph coloring (Hertz, 1987; Dubois, 1992), timetabling (Costa, 1990, Hertz, 1992) or tool management (Follonier, 1992).

3.5.2 Tabu search for the switching problem

We used the following implementation of the tabu search techniques. A solution is any permutation of the N jobs. The neighborhood is the set of all permutations that can be performed by moving a job chosen randomly from its position to an other one in the sequence. So the size of V^* is equal to N-1. The objective function simply counts the number of tool switches entailed by the job sequence (= TS($\mathcal{J}$)). Following our experiments, the parameter *tabulength* does not seem to have a revealing influence on the behavior of the tabu search; due to our definition of a move, the probability of cycling is minuscule also when *tabulength* is set to 0. Therefore after many experiments we set arbitrarily the length of the tabu list to 4.

In order to reduce the computational time of the method, the examination of the neighborhood is stopped as soon as we find a move improving the current value of the objective function. We also end the search as soon as the CPU time used is equal to the maximum of 2 seconds and the CPU time required by the MSG heuristic. This allows to evaluate the performance of the tabu search heuristic according to the CPU time it required.

Two different initial solutions were compared. First, the tabu search started from the solution proposed by the Best Position Insertion heuristic, and then from five randomly generated job sequences.

3.6 Improvement strategies

Once a job schedule is found, it is sometimes possible to obtain a better schedule, by altering it slightly. For example, Tang and Denardo (1988) enumerated explicitly the set of the *perturbed* job sequences of the best schedule obtained so far in order to find a better one; a *perturbed* sequence of a job sequence $\mathcal{J}$ is defined as a job sequence Q which can be performed by the same sequence of tools that is used to process the job $\mathcal{J}$, that is W($\mathcal{J}$). Applying the KTNS policy to the job sequence Q may reduce the number of tool switches.

Crama et al. (1991) have tested another strategy : the 2-Opt strategy. Starting from a job sequence $\mathcal{J}$, the idea is to produce a better sequence $\mathcal{J}'$ by exchanging the positions of two jobs in $\mathcal{J}$. This can be repeated until no improving exchange is possible :

Step 1: Find two jobs i and j whose exchange results in an improved sequence; if there are no such jobs, then STOP, else go to step 2.

Step 2: Exchange jobs i and j and go to step 1.

Other improvements strategies are described in Crama et al. (1991).

4. Computational experiments

The various tests have been made on a Silicon Graphics Iris Workstation (33 Mhz). Different problem instances were generated as proposed by Crama et al. (1991).

4.1 Generation of problem instances

The input data for the tool switching problem is :

- the number of jobs (N)
- the number of tools (M)
- the capacity of the tool magazine (C)
- the NxM job-tool matrix (A)

Given N,M and C, the number of tools required by a job is set randomly to an integer value belonging to the interval $[\tau_1, \tau_2]$. Matrices A are generated in the following way : for each job i, an integer t_i was drawn from the uniform distribution over $[\tau_1, \tau_2]$; this number denotes the number of tools needed for processing job i, i.e. the number of 1's in the i-th row of A. Next, a set T_i of t_i distinct integers are drawn over [1,M]. These integers denote the tools required by job i, i.e. a_{ij} = 1 if and only if j is in T_i. This process for generating the i-th line of A was repeated until we had $T_i \not\subset T_k$ and $T_k \not\subset T_i$ for all $k < i$. So the scheduling problem cannot be reduced (see Section 2.3).

Five problems were created, using the following parameters, and for each of them 10 matrices A were generated randomly :

Problem id	N	M	τ_1	τ_2
Pb 1 (10,10)	10	10	2	4
Pb 2 (15,20)	15	20	2	6
Pb 3 (30,40)	30	40	5	15
Pb 4 (40,60)	40	60	7	20
Pb 5 (60,90)	60	90	10	25

Table 2: Size of the generated problems

For each problem size, four different capacities were used, and for each of them 10 matrices A were generated randomly :

Problem id	C_1	C_2	C_3	C_4
Pb 1 (10,10)	4	5	6	7
Pb 2 (15,20)	6	8	10	12
Pb 3 (30,40)	15	17	20	25
Pb 4 (40,60)	20	22	25	30
Pb 5 (60,90)	25	27	30	37

Table 3: Capacities of the tool magazines

4.2 Detailed results

Because it is impossible to compute the optimal solution of the problem in a reasonable amount of time, we measure the performance of a heuristic as the relative difference in percent between the objective function given by the heuristic and the best value found during all our experiments. If we note $F_H(I)$ the value of the objective function given by the heuristic H applied to the problem instance I, $F_{Best}(I)$ the minimum objective function known for the problem instance I, then the performance of the heuristic H can be defined by the ratio :

$$\Delta_H(I) := \frac{F_H(I) - F_{Best}(I)}{F_{Best}(I)} * 100$$

In order to have $F_{Best}(I)$ as close as possible to the value of the optimal solution of the problem, we ran the tabu search heuristic 5 times, starting from different random job sequences, and set $F_{Best}(I)$ to the minimal value obtained; the heuristic was stopped after it has performed *nbmax* iterations without any improvement of the best value of the objective function, where *nbmax* was set respectively to 800, 1200, 1400, 1600 and 2800 for the five problems.

In the following tables, each line corresponds to a heuristic :

- Shortest Edge (SE) : with Tang and Denardo's improvement strategy.
- Farthest Insertion (FI)
- Simple Greedy (SG)
- Multiple-Start Greedy (MSG)
- Best Position Insertion (BPI)
- Tabu Search (BPI) : the sequence found by BPI is used as initial solution.
- Tabu Search (Random) : the initial solution is choosen randomly and the values are averages over 5 runs.

The given values are averages of $\Delta_H(I)$ over the ten instances generated for each size of the problem. The CPU time is the one used to solve the problem with the smallest capacity of the tool magazine. It gets smaller as the capacity grows except for the Shortest Edge heuristic (see remark below). For example, the Multiple-Start Greedy heuristic solves problem Pb 4 (40x60) in 85.0 seconds when the capacity is 20, and in 75.9 seconds when the capacity is set to 30. Mean $F_{Best}(I)$ gives an indication of the number of tool switches required to process the jobs.

As Tang and Denardo's improvement strategy is a complete enumeration of the *perturbed* job sequences, the CPU time required by the Shortest Edge heuristic can be huge; this is particularly the case when the tool magazine capacity is large. Therefore, we did not run SE with the tool magazine capacity set to C_3 or C_4.

Heuristic	Capacity of the tool magazine 4	5	6	7	CPU time (seconds)
Shortest Edge	66.9	64.3			< 0.1
Farthest Insertion	16.4	21.4	35.0	36.7	< 0.1
Simple Greedy	17.4	29.3	43.5	40.0	< 0.1
Multiple-Start Greedy	4.2	6.2	2.5	0.0	0.1
Best Position Insertion	16.5	10.7	32.5	23.3	< 0.1
Tabu Search (BPI)	0.0	0.0	0.0	0.0	2.0
Tabu Search (Random)	0.0	0.0	0.0	0.0	2.0
Mean $F_{Best}(I)$	9.9	6.7	4.3	3.0	

Table 4 : Problem 1 (10x10)

Heuristic	Capacity of the tool magazine 6	8	10	12	CPU time (seconds)
Shortest Edge	61.0	57.3			< 0.1
Farthest Insertion	28.5	23.5	17.0	10.0	0.1
Simple Greedy	22.5	23.1	25.0	12.5	< 0.1
Multiple-Start Greedy	8.8	10.6	6.0	0.0	0.5
Best Position Insertion	18.6	16.2	16.0	10.0	0.1
Tabu Search (BPI)	3.8	4.6	1.0	0.0	2.0
Tabu Search (Random)	1.2	1.5	0.0	0.0	2.0
Mean $F_{Best}(I)$	17.9	13.1	10.0	8.0	

Table 5 : Problem 2 (15x20)

Heuristic	Capacity of the tool magazine 15	17	20	25	CPU time (seconds)
Shortest Edge	56.7	59.5			< 0.1
Farthest Insertion	6.9	16.6	24.5	32.5	1.2
Simple Greedy	20.6	23.4	29.3	35.7	0.5
Multiple-Start Greedy	13.1	15.1	17.7	22.1	16.0
Best Position Insertion	3.7	8.6	15.1	31.8	1.3
Tabu Search (BPI)	2.9	6.1	9.0	11.8	16.0
Tabu Search (Random)	5.8	8.0	8.6	13.3	16.0
Mean $F_{Best}(I)$	88.8	67.3	46.9	24.7	

Table 6 : Problem 3 (30x40)

Heuristic	Capacity of the tool magazine 20	22	25	30	CPU time (seconds)
Shortest Edge	40.0	48.2			0.1
Farthest Insertion	5.3	8.7	12.6	20.1	4.7
Simple Greedy	11.3	13.2	16.5	21.0	2.1
Multiple-Start Greedy	6.5	7.6	8.9	12.9	84.8
Best Position Insertion	1.4	3.4	8.2	14.4	5.2
Tabu Search (BPI)	1.3	2.7	4.4	8.9	84.8
Tabu Search (Random)	3.6	4.7	5.3	7.0	84.8
Mean $F_{Best}(I)$	205.2	172.2	135.6	92.6	

Table 7 : Problem 4 (40x60)

Heuristic	Capacity of the tool magazine 20	22	25	30	CPU time (seconds)
Shortest Edge	4.9	6.6			74.7
Farthest Insertion	2.3	6.1	7.9	13.6	21.2
Simple Greedy	4.0	8.0	10.9	14.5	43.0
Multiple-Start Greedy	4.2	4.9	7.4	9.7	114.6
Best Position Insertion	1.1	2.4	6.9	11.6	15.1
Tabu Search (BPI)	0.8	1.6	3.7	6.7	130.7
Tabu Search (Random)	1.2	2.5	3.9	6.1	137.3
Mean $F_{Best}(I)$	205.2	172.2	135.6	92.6	

Table 8 : Problem 4 (40x60) with Global-2-Opt improvement strategy

Heuristic	Capacity of the tool magazine 25	27	30	37	CPU time (seconds)
Shortest Edge	36.0	38.9			34.7
Farthest Insertion	3.8	4.7	7.5	13.4	26.2
Simple Greedy	8.8	9.0	9.9	14.6	11.6
Multiple-Start Greedy	5.5	6.1	6.7	8.5	701.2
Best Position Insertion	0.8	1.8	3.9	10.2	28.8
Tabu Search (BPI)	0.7	1.0	2.5	3.4	701.2
Tabu Search (Random)	2.8	3.0	3.3	3.8	701.2
Mean $F_{Best}(I)$	435.5	390.7	335.0	240.0	

Table 9 : Problem 5 (60x90)

First of all we can see that there are great differences between the solutions obtained for the small problems (1 and 2) and those obtained by the others.

The bigger the size of the problem is, the smaller the difference between the performances of the heuristic is. This can be seen in Figure 1, where the capacities were set to their smallest values. The most difficult problem seems to be the second one. We can also see that the Best Position and the Farthest Insertion heuristics behave badly when they are applied to small problems (Pb 1 or Pb 2), but their performance increases with the size of the problem.

The use of Global-2-Opt improvement strategy (see Table 7 and 8) can improve considerably the quality of the solution but it requires a huge amount of CPU time. This is the case for the "bad" heuristics Shortest Edge and Simple Greedy. It seems to be interesting to use this improvement strategy combined with Best Position Insertion especially when the tool magazine capacity is large.

We can also see that it is worth to start the Tabu Search with the solution of BPI especially when the problem has a large size and when the tool magazine capacity is rather small.

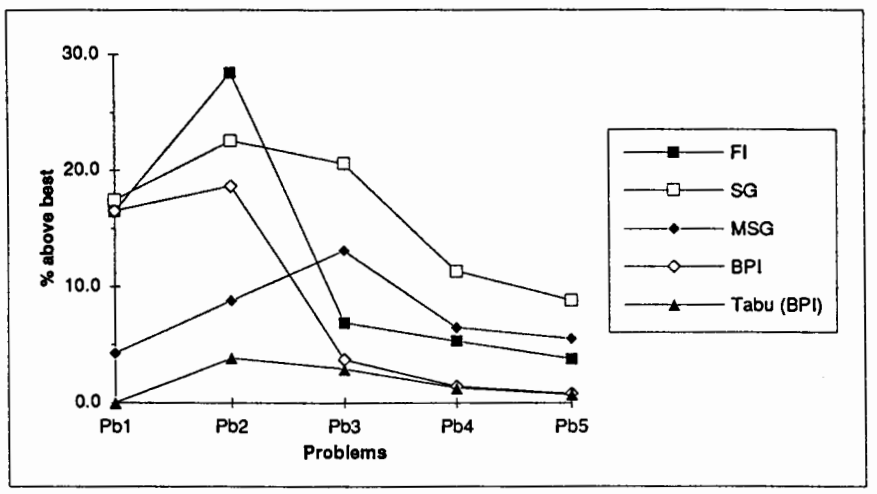

Figure 1 : Performance of the heuristics on the five problems with the smallest capacities (C_1)

Contrary to the observations of Crama et al., we can see that $\Delta_H(I)$ grows for all heuristics when the capacity of the tool magazine is increased. This is shown in Figure 2 for problem 4 (40x60). This conclusion can be drawn because we have obtained a better estimation of the optimal value of the problem due to the use of the tabu search techniques. Of course the behaviour of the different heuristics is not the same : MSG seems to be more stable than the other heuristics.

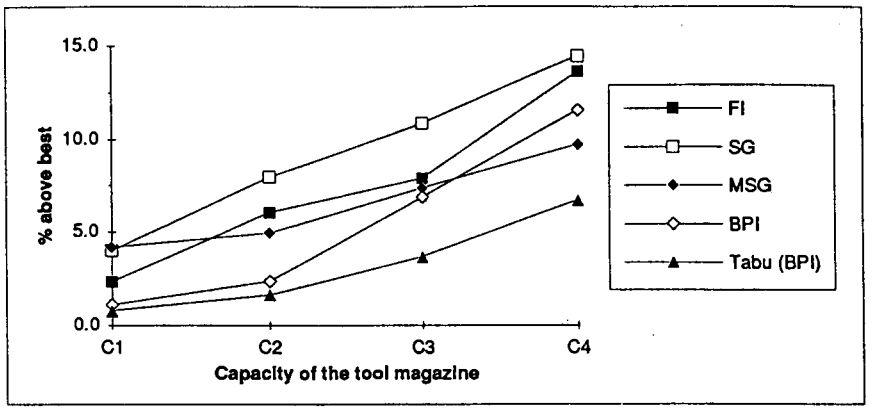

Figure 2 : Performance of the heuristics on problem 4 (40x60) according to the capacity of the tool magazine

When we take into account the required CPU time, the conclusions are totally different. Simple Greedy is a very fast heuristic, but it gives rather poor results. On the contrary, BPI gives rapidly very good solutions except for small problems. Tabu Search is very efficient but it requires more CPU time. As we stopped this heuristic as soon as the elapsed CPU time was equal to the CPU time required by MSG, Tabu Search (BPI) could not improve significantly the solution given by BPI and Tabu Search (Random) wastes time because the initial solutions are too far from the optimal one.

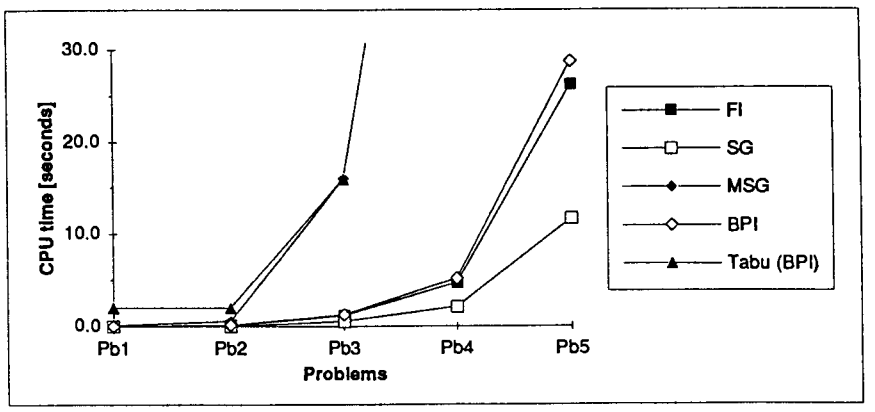

Figure 3 : CPU time required by the heuristics on the five problems

5. Conclusion

In this paper we have given an overview of the problem of minimizing the number of tool switches on a flexible manufacturing machine and have presented two new heuristics : the Best Position Insertion heuristic and a tabu search heuristic. They turn out to be efficient. BPI is a very fast insertion method which gives better results than the existing algorithms especially when the problems are of medium or large size (more than 20 jobs) and the tool magazine capacity rather small. On the contrary, tabu search is slower but gives the best results. A lot of CPU time can be saved if the search is started from a good initial solution.

The model presented is limited to the situation where jobs are to be scheduled on a single machine. It should be interesting to consider the case where more than only one machine are involved and where the tools have to be shared by the machines. A new tool management strategy should be determined since the Keep Tool Needed Soonest policy is no more optimal.

Acknowledgements

The author would like to thank Dr Alain Hertz and Daniel Costa for valuable comments about earlier versions of the paper and Prof. D. de Werra who offered him the opportunity of doing this research.

References

Bard J.F., "A heuristic for minimizing the number of tool switches on a flexible machine", *IIE Transactions*, vol 20, 1988, pp 382-391.

Costa D., "A Tabu Search Algorithm for computing an operational timetable", *ORWP 90/19*, DMA–EPFL, Lausanne, Switzerland,1990. To appear in *EJOR*.

Crama Y., Kolen A.W.J., Oerlemans A.G., Spieksma F.C.R., "Minimizing the number of tool switches on a flexible machine",.*Research Memorandum 91-010,* University of Limburg, Maastricht, The Netherland, 1991.

Dubois N., "EPCOT : an efficient procedure for coloring optimally with tabu search". To appear in *Computers & Mathematics with Applications*, 1992.

Follonier J.-P., "On grouping parts and tools and balancing the workload on a flexible manufacturing system", *ORWP 92-05*, DMA–EPFL, Lausanne, Switzerland, 1992.

Glover F., "Tabu Search-Part I", *ORSA Journal on Computing*, vol 1(3), 1989, pp 190-206.

Glover F., "Tabu Search-Part II", *ORSA Journal on Computing*, vol 2(1), 1990, pp 4-32.

Golden B.L., Stewart W.R., "Empirical analysis of heuristics", in *The traveling salesman problem*, E.L. Lawler & al. (eds), John Wiley, 1985, pp 207-249.

Hertz A., de Werra D., "Using tabu search for graph coloring", *Computing*, vol 39, 1987, pp 345-351.

Hertz A., "Tabu search for large scale timetabling problems", European Journal of Operational Research, vol 54, 1992, pp 39-47.

Tang C.S., Denardo E.V., "Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches", *Operations Research*, vol 36, no 5, 1988, pp 778-784.