Journal of Intelligent Manufacturing (1995) **6**, 87–94

Modelling a tool switching problem on a single NC-machine

CAROLINE PRIVAULT and GERD FINKE

Laboratoire Artemis-Imag, BP 53, 38041 Grenoble Cedex 09, France

The problem addressed in this paper is a tool switching problem on a single numerically controlled machine. In the first part, the sequence of jobs is given and the tooling aspect is dealt with: the uniform case where all switching times are equal is described, and a model for the general nonuniform case is proposed. This problem is reduced to the problem of finding a minimum cost flow of maximum value in an acyclic network. The second part is devoted to the tool management scheduling problem. Some heuristics are presented and computational results are given and analyzed.

Keywords: Tool management, scheduling, mini-cost flow, heuristics

1. Introduction

This problem occurs in flexible manufacturing systems where numerically controlled machines are used to manufacture different types of parts: a great variety of part families but small batch sizes. In this case, machines have to be very flexible and each of them performs different operations, using a set of special tools. To give an idea of the number of tools required, consider for example Case's workshop (at St Dizier, France). In this shop, 27 000 transmissions per year are produced. All the part families (transmissions, crankcases, cast-iron pieces, ...) require the use of 1800 different tools. In this context, a lot of time can be wasted with tool handling, tuning and switching, and efficient tool management appears to be a deciding factor in the total completion time. The single machine problem arises especially in the metalworking industry, and for cutting tool management. For example, Bard (1988) studied a problem of tool setup and volume removal time minimization on a single flexible machine which performs cutting operations; and Kiran and Krason (1988) described several methods for tooling automation on milling centres. In the same way, Kusiak and Finke (1987) described an NC-forging machine, for which effective tool management is vital.

In this paper, a single NC-machine equipped with an individual tool magazine of limited capacity is considered. Tool magazines may be of different types: disks, drums or turrets, and chains. They usually contain an average of 30 tools, but drums or chains may have larger capacity (more than 100 tools). A set of N jobs has to be scheduled on the machine. Each job requires a number of tools that does not exceed the capacity of the magazine. All tools are kept in a general tool storage area. They may be stored on a rack, or a disk stocker, located close to the machine. If this is not the case, automated guided vehicles may be used to transfer tools from the tool room to the tool magazine of the machine. Most of the time, an automated tool changer is located on the machine (gantry loaders or robot arms), which contributes to reducing setup times (Kusiak, 1990). However, even with all these automatic devices, a lot of machine processing time can be wasted on the loading and unloading operations of tools, because tools still require fine tuning during changes. Before each job is processed, the corresponding tools have to be placed in the tool magazine; if no place is available, unused tools must be unloaded. The choice of these tools depends on the next jobs to be processed. The problem is then to find a schedule of the N jobs in order to minimize the total number of tool changes. Tang and Denardo (1988) were the first to study the problem in this form.

As a related field of application one can mention the production of circuit boards in the electronic industry: components are placed automatically on printed circuit boards by NC-machines before the soldering phase. A special type of machine is used to place (or insert) a

0956–5515 © 1995 Chapman & Hall

particular type of electronic component (Surface Mounted Components for example). This machine for example is equipped with a limited number of SMC component feeders, and feeder switchings may be necessary when new types of circuits are assembled (with other types of components). Here tools can be identified with feeders, and then we have to find a sequence of circuit types on the machine, in order to minimize feeder exchange times which are time-consuming. Bard (1988) described a similar problem. These case studies show that the tool switching problem on single machining centres is a real problem in itself.

Initially the complexity of the problem is described. Then, the tooling problem in the uniform case is studied, and a model for the nonuniform case is given using a network flow algorithm for its solution. In Section 4, the sequencing problem is studied. Four heuristic methods are described and compared on a set of randomly generated instances.

2. An NP-hard problem

This problem has two aspects: firstly, it is a scheduling problem and secondly, one has a tool management problem. At each instant, it is necessary to choose which tools are unloaded to make room for the requested tools. These two problems are of course related, which increases the complexity. Moreover, Crama *et al.* (1991) have established that this is already an NP-hard problem for the capacity of two tools. They showed that the decision problem of the Hamiltonian path in an edge-graph (given an edge-graph H, does H contain a Hamiltonian path?), polynomially reduces to the scheduling problem with tool management (as defined). The Hamiltonian path in an edge-graph is an NP-complete problem (see Bertossi, 1981). On the contrary, if we have a fixed sequence of jobs, it is necessary only to manage tools in the magazine, in order to minimize the number of tool switches in the sequence. As will be seen in the next section, this problem can be solved in polynomial time.

3. The tool management problem

3.1. *Tool switching in uniform time*

This is the case described in the first section: it is supposed that all setup times are equal. It is assumed that we always begin with an empty tool magazine. Then after the first jobs, and during subsequent jobs in the sequence, the machine uses a full magazine. Given the order of the jobs on the machine, the KTNS rule (Keep Tools Needed Soonest), of Tang and Denardo is applied to manage the tools replacements in the sequence. This means that a tool is inserted only if it is directly requested by the current job, and that to make room for it, we choose the unrequested tool that will be needed the latest.

The algorithm gives an optimal loading of the magazine and runs in polynomial time of order $O(MN)$, with N the number of jobs in the sequence, and M the total number of tools requested for the whole sequence.

Tang and Denardo first proved the optimality of the KTNS rule. They established the following: given a loading plan of the magazine, they showed with tool interchange arguments that this loading plan can be modified in order to follow the KTNS principle, and without increasing the number of tool changes. Then, Crama *et al.* gave another proof, using a linear programming algorithm of Hoffman, Kolen and Sakarovitch (see Crama *et al.*, 1991). It can be seen that although the KTNS principle appears to be very clear and simple, it is still laborious to prove its optimality. This remark motivated the present authors to find a different model for the tooling management, which also allows the generalization to nonuniform set-up times.

3.2. *A more general case: nonuniform set-up time*

For our general tool switching problem, some tools may require more time for their manipulation and tuning operations than others. The exchange time of a tool may also depend on the previous tool at that position. We assign a general weight d_{ij} to the removal of tool i and insertion of tool j into the magazine. This defines a weight matrix (d_{ij}) with M rows and M columns, where $d_{ij} = 0$ if $i = j$ (in the previous section, we had $d_{ij} = 1$ for all $i \neq j$). In this case, the goal is to minimize the sum of the tool switching times. In the next section, we formulate this problem as a problem of finding a flow of maximum value and minimum cost in an acyclic graph.

3.2.1. *Formulation of the general tooling problem*

The graph consists of a source (s), a sink (p) and three levels of vertices (see Fig. 1): the first one contains C vertices, where C is the capacity of the tool magazine. This gives the initial configuration of the magazine. Each of them contains one unit of flow, and is connected to each vertex of the second level. This second level corresponds to groups of requested tools, one group for each job in the sequence. It is composed of vertices r_j where each r_j corresponds to a requested tool. These requests are numbered according to their appearance in the sequence. Tools belonging to the same group (or job) are arranged in increasing order of their tool number 1, 2, ..., M. For example, consider the following sequence of five jobs and their tools:

job1: 2,3
job2: 2,1
job3: 3
job4: 1
job5: 3,1

We obtain a sequence of tools:

$$r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8$$
$$(2,3)\ (1,2)\ (3)\ (1)\ (1,3)$$

We continue the construction of the graph which is illustrated in Fig. 1 for our example. From each vertex (v) of the first level, there is an arc to vertex (r_j) of the second level. The cost of this arc is equal to the switching time between tool r_j and tool v in the magazine (if we begin the production with a full magazine), and zero otherwise. The third level consists of copies r_j' of all vertices r_j of the second level. With this construction, we find again the same sets of vertices of level two which correspond to the groups of tools of each job in the sequence. Between each r_j and r_j', there is a 'horizontal' arc of large negative weight $(-k)$ where k is a large number. Within a same group and between a group and its copy, there are no other arcs.

There are arcs between groups of the third level, and groups of the second one. They always start at a copy r_i' and terminate at a vertex r_j, situated in any one of the succeeding groups (jobs that are requested later). Their weight is the switching time d_{ij}. In Fig. 1, only a few of these arcs are displayed, for instance, the arc from tool 2' of the second job to tool 3 of the third job. These arcs guarantee that tools will be inserted in the right order (i.e. they will be in the magazine when requested), and that no cycle can exist in the network. Finally, there is also an arc from vertex r_j' to the sink (p) at zero cost. All arc capacities are equal to one.

On this graph, the tooling problem reduces to the problem of computing a flow of maximum value and minimum cost. Since no arc exists between two vertices of the same set, each unit of flow will pass on only one vertex of a set. Tools of the same set will be in magazine simultaneously to execute the corresponding job. Therefore, the path followed by each flow unit describes the successive loadings of the corresponding slot in the tool magazine. On our network, the max-flow value is equal to the sum of the arc capacities starting at the source (s). This value is equal to the magazine capacity C.

Since k is a very large number, each minimum cost flow must saturate all the 'horizontal' arcs (r_j, r_j'), i.e. each tool will be inserted when requested. This construction is illustrated with an example from Crama *et al.* (1991), whose incidence matrix (tools/jobs) is the $M \times N$ matrix $\boldsymbol{T}$ defined by:

$$\boldsymbol{T}_{ij} = 1 \text{ if tool } i \text{ is required by job } j$$
$$= 0 \text{ otherwise.}$$

(here $M = 3$ tools and $N = 5$ jobs)

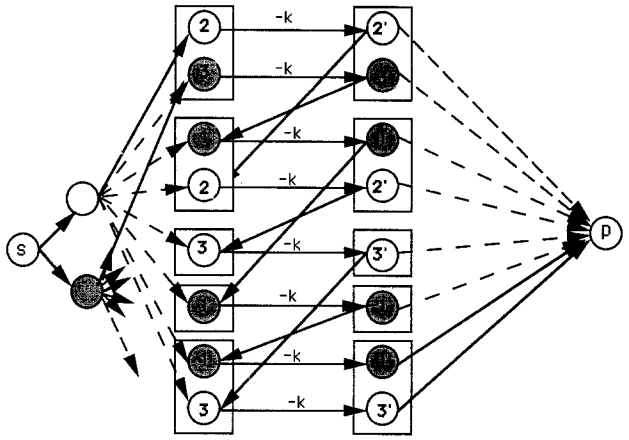

Fig. 1. Network construction

$$\boldsymbol{T} = \begin{bmatrix} 0\,1\,0\,1\,1 \\ 1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,1 \end{bmatrix}$$

and we have a weight matrix:

$$\boldsymbol{D} = \begin{bmatrix} d_{11}\,d_{12}\,d_{13} \\ d_{21}\,d_{22}\,d_{23} \\ d_{31}\,d_{32}\,d_{33} \end{bmatrix}$$

It is supposed that columns are ordered according to the job sequence and that the machine has a two-slot magazine. In Fig. 1, not all arcs were drawn, only those taken by the min-cost flow.

It is supposed that we have computed a flow of value 2, and that this flow attains the minimum cost corresponding to the weight matrix of tools {1, 2, 3}. On the network, the 'light grey' vertices define the path followed by one unit of the max-flow, and the 'dark grey' vertices define the path of the other. This flow can be visualized in the form of the classical replacement table:

jobs:	1	2	3	4	5
tools:	2	2→3	3	3	
	3→1	1	1	1	

with a total switching time of $d_{31} + d_{23}$

3.2.2. *Solving the general tooling problem*

One solution to the nonuniform tooling problem has been suggested by Oerlemans (1992), in the form of a linear programming formulation. This seems to be the only existing method. The min-cost flow model is an extension of the model given by Chrobak *et al.* (1991) for solving the single tool problem; it means that each job in the sequence requests only one tool. In fact it is interesting to notice that this problem is a particular off-line application of a more general problem: the k-server problem (see Section 4) (McGeogh and Sleator, 1989). The k-server problem has several applications, especially for operating systems where

paging is used for memory management (Fiat *et al.*, 1991; Mattson *et al.*, 1970).

To obtain an optimal tool management, we apply the minimum cost augmentation method of Tarjan (1983): the flow is always augmented along a minimum cost path in the residual graph. This algorithm runs in polynomial time because the network is acyclic and all capacities are integers. At each step, one unit of flow is pushed through the network and since it is an acyclic network, the algorithm will compute the flow in at most C augmentations.

Let t_j be the number of tools required by job j. The network has a total of $2*\sum(t_j,\ j=1,\ \ldots\ ,\ N) + C + 2$ nodes. We have $t_j \leq C$ for each j, so that the number of vertices is less than $2C*N$. This implies that an optimal tool loading will be computed in at most C^3*N^2 operations. This result can be improved by computing successive minimum cost augmenting paths using a good single source shortest-path algorithm (Tarjan, 1983).

3.2.3. *Application to the uniform case*

The model can also be applied to the uniform case if we set all weights equal to 1 in the network. It gives an alternative method to KTNS, which is in $O(N^2)$ whereas the KTNS procedure is in $O(MN)$. As the number of tools requested by all jobs is most of the time greater than the number of jobs, we have $M > N$. Consequently, minimizing the number of tool switches by computing a maximum flow of minimum cost can be a rather competitive method.

3.2.4. *Remark*

When the tools required by a job to be processed next, have to be inserted in the tool magazine, the requested tools already in place are obviously kept there, whereas the others are inserted. This simple rule should be followed by the max-flow min-cost algorithm. Let us suppose that it is not the case for a flow of value C at minimum cost. One unit flow placed on a vertex (corresponding to a tool numbered 1) in the first set is going to a vertex (tool) numbered 2 in the next set, whereas one unit is leaving vertex 3 in the first set (or a previous set) for tool 1 in the next set. The local cost is $d_{12} + d_{31}$ (see Fig. 2).

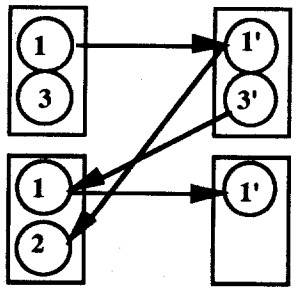

Fig. 2. Non-optimal flow configuration

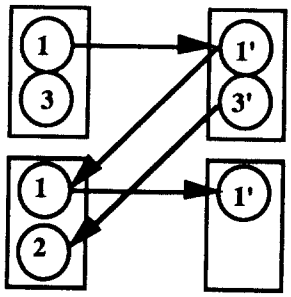

Fig. 3. Optimal flow configuration

If the flow unit placed on 1 in the first set is kept on 1 in the next set (see Fig. 3) and if the other unit goes from 3 to 2, then the new local cost is $d_{32} + d_{11} = d_{32}$, and the flow is the same anywhere else. Now if the triangular inequality is verified, we have $d_{12} + d_{31} \geq d_{32}$ and therefore the flow could not be of minimum cost.

Let us now return to the uniform case, and study the scheduling problem with tool management, when all setup times are equivalent. In the following section, the main ideas of the methods will be outlined. More details can be found in Crama *et al.* (1991), Golden and Stewart (1985), and Privault and Finke (1993).

4. Scheduling jobs in order to minimize tool switches

It is assumed that we have to construct the job sequence and manage tool switches altogether. Since it is an NP-hard problem, we will focus now on heuristics to solve it. Several methods have been proposed. The most important and efficient ones are described. Among all of them, we can distinguish two main types: the first class gathers heuristics which first construct the whole sequence, and then manage the tool switches with the KTNS procedure. It includes also all improvement methods that follow this strategy (alteration of a sequence, and evaluation of the new sequence by KTNS). In the second class, we find heuristics that construct the sequence and manage tools altogether. Most of the methods found in the literature belong to the first class.

4.1. *First type heuristics*

4.1.1. *The farthest insertion method*

To construct the sequence, the classical heuristic methods of the travelling salesman problem have been tested by Crama *et al.* (1991). They tried the shortest edge heuristic, nearest neighbour, farthest insertion, and a branch and bound algorithm. These heuristics are applied to a complete graph whose vertices are jobs, and whose edges are

weighted by an approximation of the number of tool switches between each pair of jobs. This approximation called *Lb* is in fact a lower bound for the number of tool changes for a given capacity C:

$$Lb\ (i,j) = \max\ \{0, |T_i| + |T_j| - |T_i \cap T_j| - C\}$$

with T_j the set of tools requested by job j, and $|T_j|$ the cardinality of the set.

According to their computational experiments, farthest insertion seems to give the best results. However, the main problem for such methods is the reliability of the approximation of the number of tool switchings between two jobs. This number is sequence-dependent, because all jobs do not fully occupy the magazine, and free slots may be held by other tools requested before and during the sequence. Then the quality of the only approximation we have depends strongly on the type of the instances. Most of the time, the bound *Lb* gives values that are too optimistic. Among improvement methods, the 2-Opt strategy (Golden and Stewart, 1985) has also been tested and seems to be efficient, but it requires too many computations (Crama *et al.*, 1991).

The authors have suggested a restricted 2-Opt method which can be used for improvement of the solution given by another heuristic. It consists in improving the sequence by interchanging only consecutive jobs. This reduces the computations, but the results still depend on the quality of the starting sequence.

4.1.2. *The 'Super task' method*

The procedure called 'super task' is the first proposed by Privault and Finke (1993). The principle is to gather several jobs that use common tools, until the 'big job' obtained nearly requires the whole tool magazine. This 'grouping' stage is repeated with the new tasks, and the single remaining jobs, until no more jobs can be added without exceeding the capacity of the magazine. It is then easier to approximate the number of tool switchings between tasks: the lower bound *Lb* nearly gives the exact number of tool switches. Afterwards all the 'big jobs' (and single remaining jobs) are ordered using the shortest edge method and the *Lb* approximation. Jobs within the same task are arbitrarily placed in order of their addition to the task. To avoid undesired effects due to these partial orders, restricted 2-Opt is finally applied to improve the whole sequence. According to the authors' computational experiments, this approach appears to give rather good solutions rapidly (see last section).

4.1.3. *The best insertion method*

Jobs are sorted according to the following priority rule: for each job, we calculate $m(j) = \min\ \{Lb(i,j),\ i \neq j\}$, and then jobs are ordered such that $m(j_k) \geq m(j_{k+1})$ for $k = 1\ (N-1)$. The idea is to place in the sequence, with priority, jobs that induce the most tool switches before they are processed even in the most favourable case. This is the meaning of $m(j)$. Then the first two jobs are selected to constitute a first partial sequence $S = \{j_1, j_2\}$. The next job in the priority list is inserted at the three possible places in S: first, between j_1 and j_2, and last. For each place, the corresponding sequence is evaluated by the KTNS procedure, and the best position is retained. The next jobs are inserted in the same way, and at the end, $[N(N+1)/2] - 3$ sequences have been tested. This method is nearly as competitive as the *super task* method, but it requires many more computations.

4.2. *The second class methods*

4.2.1. *Next best*

In the second class of methods, is featured the *next best* heuristic which consists in constructing the sequence step by step, choosing the next job to be processed as the one which has the most tools already loaded in the last magazine of the partial sequence (this is in fact the nearest neighbour principle). Then the required tools are loaded on the machine, and if space is needed, the unused tools, which will be the less frequently used by all jobs that have not been processed yet, are unloaded.

This last 'unload criterion' is not really efficient, but it is precisely the main difficulty of the second class strategies: how to make a good choice between unused tools to make room in the machine magazine if we do not know exactly which job will be processed next and which tools will be needed again. This problem can also occur when jobs are processed as soon as they are ordered by customers in a workshop.

4.2.2. *A new approach: online tool management*

Generally speaking, the question is how to take decisions whose quality depends on the future development of the instance. This is typically an online problem, and this is why the link with the k-server problem was made (McGeogh and Sleator, 1989; Privault and Finke, 1993). In this problem, there is a collection of k mobile servers residing on a set of n vertices. One has to decide how to move the servers in response to each request of a sequence by considering only the current and past requests in the sequence. We have single online requests whereas for the tooling problem, we have online sets of requests.

This motivated the authors to adapt McGeoch and Sleator's competitive algorithm for the uniform single request problem (the so-called partitioning algorithm), to serve sets of requests. Using this online management, and the nearest neighbour principle, a heuristic called 'part' is suggested: each job is chosen to be first in the sequence (each in its turn). Then the sequence is constructed as the next best sequence, but for each job, tools are replaced according to a new choice criterion given by the adapted

version of the online partitioning algorithm. We obtain N sequences, evaluated by the KTNS rule, and the best one is selected: more details can be found in Privault and Finke (1993) and Privault (1994).

5. Computational results

Crama *et al.* (1991) gave several other interesting heuristics, but they suggested the use of farthest insertion or simple greedy (which is nearly the same principle as next best) when good solutions have to be computed quickly, and the 2-Opt strategy when time does not matter. Among these methods and the one described by the authors, it was decided to compare those which are known to give good results in a reasonable computing time. 2-Opt for example can take 30 min to solve a problem of 60 tools, 40 jobs and for a capacity of 20, whereas other heuristics will take a few seconds on the same type of instance.

In the same way, the authors selected the single-start version of the next best method rather than the multiple-start one: the first job in the sequence is chosen almost arbitrarily as the job requiring the greatest number of most frequently used tools. In fact, the authors' previous computational experiments showed that the multiple-start version of course gives better results than the single-start version, but it is really time consuming (more than 2 h for some 100 tool/50 job problems).

Finally, *farthest insertion, super task, next best*, and the *partitioning method* (named *part* in Tables 1, 2, and 3) were selected. One-hundred-and-twenty random instances were generated, varying the parameters M, N, C and the filling rates of the jobs compared to the magazine capacity: this ratio is the number of tools per job compared to the number of places in the magazine. It is known that this parameter could make some problems easier to compute. For example, if all jobs saturate the magazine, there is no need for tooling management any more.

Twelve classes of problems were considered, and for each one, 10 random instances were generated. For each problem the solution given by each heuristic was computed, and for the four methods the percentage above the best solution found was calculated. This means that on a problem, one method (or more) is achieving 0%; and then the average percentage of each method on the ten problems is computed for each type of instance. In the same way, the computing times are averages on ten runs. The authors chose to test the following parameters (N, M): (20, 50), (40, 60), (50, 100), and tried two capacities C_1 and C_2 for each couple.

Tables 1 and 2 show the good results obtained by the *partitioning method* on the 120 matrices. These performances are almost independent of the parameters. This is not the case for the *farthest insertion* method. If the filling rate is 50% up to 100%, that is on dense incidence matrices, it gives nearly the smallest number of tool switches, whereas on sparse matrices, the score is as bad as the score given by any random sequence. This phenomenon is due to the use of the *Lb* approximation of the tool switches. In the first case, it gives almost the exact number of tool replacements, but for filling rates between 30% and 60%, it is rather too optimistic: all lower bounds are close

Table 1. (20, 50) Matrices (type: N = 20 jobs: M = 50 tools)

		Heuristic method					*Running times (s)*				
Capacity	*Filling rates of jobs %*	*Part*	*Next best*	*Super task*	*Farthest insertion*	*Random sequence*	*Part*	*Next best*	*Super task*	*Farthest insertion*	*Random sequence*
C_1 = 15	(30, 60)	0	11.7	5	25.2	32.5	3.9	16.27	0.315	0.508	–
	(50, 100)	1.3	8.9	6.3	0.7	31	5.48	23.11	0.347	0.605	–
C_2 = 26	(30, 60)	0	17.9	7.9	31.1	36.3	3.85	22.92	0.347	0.517	–
	(50, 100)	2.1	11.2	6.7	0.8	35.6	6	35.23	0.493	0.649	–

Table 2. (40, 60) Matrices (type: N = 40 jobs; M = 60 tools)

		Heuristic method					*Running times (s)*				
Capacity	*Filling rates of jobs %*	*Part*	*Next best*	*Super task*	*Fartherst insertion*	*Random sequence*	*Part*	*Next best*	*Super task*	*Fathest insertion*	*Random sequence*
C_1 = 20	(30, 60)	0.34	18.06	5.54	40.24	47.17	21.6	33.8	1.03	3.79	–
	(50, 100)	0.73	5.88	6.46	0.326	36.4	31.9	52	1.22	4.4	–
C_2 = 35	(30, 60)	0	16.5	8.04	41.44	50.82	20.9	52.28	1.17	3.77	–
	(50, 100)	0.7	7.62	3.08	0.73	39.94	33	86	2.08	4.5	–

Table 3. (50, 100) Matrices (type N = 50 jobs; M = 100 tools)

		Heuristic method					*Running times (s)*				
Capacity	*Filling rates of jobs %*	*Part*	*Next best*	*Super task*	*Farthest insertion*	*Random sequence*	*Part*	*Next best*	*Super task*	*Farthest insertion*	*Random sequence*
C_1 = 30	(30, 60)	0	11.16	5.6	28.6	29.85	104.8	79.4	2.4	10.8	–
	(50, 100)	0.45	5.50	6	0.62	26.15	152.9	121.5	2.9	13.5	–
C_2 = 60	(30, 60)	0	15.6	6.34	33.49	37.09	90.8	142.1	2.8	10.9	–
	(50, 100)	0.04	5.88	3.2	1.09	31.33	145.5	231.9	5.9	16.3	–

or equal to zero, and *farthest insertion* gives random sequences. It can also be seen that between the two second class methods, *next best* seems to be less efficient than the *part* method, which allows one to say that the partitioning algorithm may be an interesting way of solving online tool management.

Concerning the running times, the four heuristics have been implemented in Pascal, and tested on a Sun (sparc 1), but optimization of the code was not seriously pursued. However, it is still interesting to give a rough idea of the average times for comparison. Anyway they are rather short, especially for the *super task* method, which also provides good solutions. *Next best* and *part* require of course the longest computing times. Table 3 gives results on 40 larger matrices. It confirms our conclusions on their quality and running times.

6. Concluding remarks

In summary, we have seen that the two related aspects of this scheduling problem (sequencing and tooling) result in a difficult problem. For the sequencing problem, *super task* can be used when good solutions have to be found quickly, and the *partitioning method* can be chosen to compute high quality solutions. These two methods offer the main advantage of being independent of the instance type, and especially of the sparsity of incidence matrices. Sparse incidence matrices constitute the most difficult cases: a great variety of different tools requested for global production, but a small number of tools used per job.

As far as general tooling management is concerned, the max-flow min-cost model that was described provides a general optimal method for solving problems with special different setup times. This algorithm can be applied to the classical uniform case, as a competitive alternative method to the KTNS procedure when the total number of tools largely exceeds the number of jobs.

In general, flexible machines may belong to a system of several (identical or dedicated) machines. For dedicated machines (which could be a part of a flow-line), machines usually require special sets of tools. Therefore, our tooling management model may be applied independently to each of these specialized machines. In the case of a cell of identical machines, one has the additional problem of the allocation of jobs — with the necessary tools — to the machines. This leads to the scheduling theory of parallel machines under resource constraints (Blazewicz *et al.*, 1988), which was not considered in this research. Very few papers analyse explicitly such general cases. Oerlemans (1992) studied identical machines with the same magazine capacities, as a job grouping problem. A tabu search technique has been applied by Widmer (1991) to solve a jobshop problem with tooling constraints.

References

Bard, J. F. (1988) A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, **20**(4), 382–391.

Bertossi, A. A. (1981) The edge Hamiltonian path is NP-complete. *Information Processing Letters*, **13**(4, 5), 157–159.

Blazewicz, J., Finke, G., Haupt, R. and Schmidt, G. (1988) New trends in machine scheduling. *European Journal of Operational Research*, **37**, 303–317.

Chrobak, M., Karloff, H., Payne, T. and Vishwanathan, S. (1991) New results on server problems. *SIAM Journal on Discrete Mathematics*, **4**(2), 172–181.

Crama, Y., Kolen, A. W. J., Oerlemans, A. G. and Spieksma, T. C. R (1991) Minimizing the number of tool switches on a flexible machine, Research Memorandum 91.010, Limburg University.

Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D. and Young, N. E. (1991) Competitive paging algorithms. *Journal of Algorithms*, **12**, 685–699.

Golden, B. L and Stewart, W. R. (1985) Empirical analysis of heuristics, in *The Traveling Salesman Problem*, Lawler, E. L. *et al.* (eds), John Wiley & Sons, Chichester, pp. 207–249.

Kiran, A. S. and Krason, R. J. (1988) Automated tooling in a flexible manufacturing system. *Industrial Engineering*, April, 52–57.

Kusiak, A. (1990) *Intelligent Manufacturing Systems*, International Series in Industrial and Systems Engineering, Prentice Hall, New Jersey, pp. 35–45.

Kusiak, A. and Finke, G. (1987) Modeling and solving the flexible forging module scheduling problem. *Engineering Optimization*, **12**, 1–12.

Mattson, R., Gecsei, J., Slut, D. and Traiger, I. (1970) Evaluation techniques for storage hierarchies. *IBM System Journal*, **2**, 78–117.

McGeoch, L. A. and Sleator, D. (1989) A strongly competitive randomized paging algorithm, Carnegie Mellon University, Computer Science technical report, CMU-CS-89-122.

Oerlemans, A. G. (1992) Production planning for flexible manufacturing systems, Ph.D. Thesis, University of Limburg, Maastricht.

Privault, C. (1994), Modèles mathématiques pour la gestion off-line et on-line des changements d'outils sur une machine flexible, Thèse de Doctorat, Université Joseph Fourier, Grenoble.

Privault, C. and Finke, G. (1993) Gestion on-line des changements d'outils sur une machine flexible, in *Actes du Colloque 'Aide à la Décision et Recherche Opérationnelle', Congrès AFCET '93*, Versailles, **1**, pp. 175–184.

Tang, C. S. and Denardo, E. V. (1988) Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. *Operations Research*, **36**(5), 767–777.

Tarjan, R. E. (1983) Data structures and network algorithms. *BMS-NSF Regional Conference Series in Applied Mathematics*, **44**, 109–111.

Widmer, M. (1991) Job shop scheduling with tooling constraints: a tabu search approach. *Journal of the Operational Research Society*, **42**(1), 75–82.