The International Journal of Flexible Manufacturing Systems, 6 (1994): 33-54

Minimizing the Number of Tool Switches on a Flexible Machine

YVES CRAMA, ANTOON W.J. KOLEN AND ALWIN G. OERLEMANS
Department of Quantitative Economics, University of Limburg, P.O. Box 616, 6200 MD Maastricht, The Netherlands

FRITS C.R. SPIEKSMA
Department of Mathematics, University of Limburg, P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract. This article analyzes a tool switching problem arising in certain flexible manufacturing environments. A batch of jobs have to be successively processed on a single flexible machine. Each job requires a subset of tools, which have to be placed in the tool magazine of the machine before the job can be processed. The tool magazine has a limited capacity, and, in general, the number of tools needed to produce all the jobs exceeds this capacity. Hence, it is sometimes necessary to change tools between two jobs in a sequence. The problem is then to determine a job sequence and an associated sequence of loadings for the tool magazine, such that the total number of tool switches is minimized. This problem has been previously considered by several authors; it is here revisited, both from a theoretical and from a computational viewpoint. Basic results concerning the computational complexity of the problem are established. Several heuristics are proposed for its solution, and their performance is computationally assessed.

Key Words: computational complexity, heuristics, sequencing, tool management.

1. Introduction

The importance of tool management for the efficient use of automated manufacturing systems has been recently stressed by several authors; we refer for instance to Gray, Seidmann and Stecke (1988) or Kiran and Krason (1988) for a thorough discussion of this issue. In particular, a central problem of tool management for flexible machines is to decide how to sequence the parts to be produced, and what tools to allocate to the machine, in order to minimize the number of tool setups. The problem becomes especially crucial when the time needed to change a tool is significant with respect to the processing times of the parts, or when many small batches of different parts must be processed in succession. These phenomena have been observed in the metalworking industry by Hirabayashi, Suzuki and Tsuchiya (1984), Finke and Kusiak (1987), Bard (1988), Tang and Denardo (1988), Bard and Feo (1989), etc. Blazewicz, Finke, Haupt and Schmidt (1988) describe for instance an NC-forging machine equipped with two tool magazines, each of which can handle eight tools. The tools are very heavy, and exchanging them requires a sizable fraction of the actual forging time. Another situation where minimizing the number of tool setups may be important is described by Förster and Hirt (1989, p. 109). These authors mention that, when the tool transportation system is used by several machines, there is a distinct possibility

that this system becomes overloaded. Then, minimizing the number of tool setups can be viewed as a way to reduce the strain on the tool transportation system. Bard (1988) mentions yet another occurrence of the same problem in the electronics industry. Suppose several types of printed circuit boards (PCBs) are produced by an automated placement machine (or a line of such machines). For each type of PCB, a certain collection of component feeders must be placed on the machine before boards of that type can be produced. As the machine can only hold a limited number of feeders, it is usually necessary to replace some feeders when switching from the production of one type of board to that of another type. Exchanging feeders is a time-consuming operation and it is therefore important to determine a production sequence for the board types which minimizes the number of "feeder setups." Identifying the feeders with tools, we see that this constitutes again an instance of the "job-sequencing and tool loading" problem evoked above.

The present article deals with a particular formulation of this problem, due to Bard (1988) and Tang and Denardo (1988). Suppose that N jobs have to be successively processed, one at a time, on a single flexible machine. Each job requires a subset of tools, which have to be placed in the tool magazine of the machine before the job can be processed. The number of tools needed to produce all the jobs in the batch is denoted by M. We represent the data by an $M \times N$ *tool-job* matrix A, with:

$$\begin{aligned} a_{ij} &= 1 \quad \text{if job } j \text{ requires tool } i, \\ &= 0 \quad \text{otherwise,} \end{aligned}$$

for $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. Without loss of generality, A has no zero row. The tool magazine has a limited *capacity*: it can accommodate at most C tools, each of which fits in one slot of the magazine. To ensure feasibility of the problem, we assume that no job requires more than C tools. We also assume that, while the jobs are in process, the tool magazine is always loaded at full capacity (as will be explained below, this is in fact a nonrestrictive assumption for our problem). We thus call any subset of C tools a *loading* of the magazine.

A *job sequence* is a permutation of $\{1, 2, \ldots, N\}$, or, equivalently, of the columns of A. As the number of tools needed to produce all jobs is generally larger than the capacity of the tool magazine (i.e., $M > C$), it is sometimes necessary to change tools between two jobs in a sequence. When this occurs, one or more tools are removed from the tool magazine and are replaced by a same number of tools retrieved from a storage area. We call *setup* the insertion of a tool in the magazine. A *switch* is the combination of a tool setup and a tool removal. Since each tool has to be set up at least once in order to process the whole batch of jobs, we will also pay attention to the *extra setups* of a tool, that is, to all setups of the tool other than the first one.

The *tool switching problem* is now defined as follows: determine a job sequence and an associated sequence of loadings for the tool magazine, such that all tools required by the jth job are present in the jth loading, and the total number of tool switches is minimized. In matrix terms, the tool switching problem translates as follows: determine an $M \times N$ 0-1 matrix $P = (p_{kj})$, obtained by permuting the columns of A according to some job sequence, and an $M \times N$ 0-1 matrix $T = (t_{kj})$ containing C ones per column (each

column of T represents a tool loading), such that $t_{kj} = 1$ if $p_{kj} = 1$ (i.e., tool k is placed in the jth loading if it is needed for the jth job in the sequence; $k = 1, \ldots, M$; $j = 1, \ldots, N$), and the following quantity is minimized:

$$\sum_{j=2}^{N} \sum_{k=1}^{M} (1 - t_{k,j-1})t_{kj}.$$

(This quantity is exactly the number of switches required for the loading sequence represented by T). Observe that minimizing the number of tool switches is equivalent to minimizing the number of setups or of extra setups, since the following relations hold:

$$\begin{aligned} \text{number of setups} &= \text{number of switches} + C \\ &= \text{number of extra setups} + M. \end{aligned}$$

Let us now briefly discuss some of the (explicit and implicit) assumptions of the tool switching model.

(1) As mentioned before, the assumption that the tool magazine is always fully loaded does not affect the generality of the model. Indeed, since no cost is incurred for tools staying in the magazine, one may consider that the first C tools to be used are all incorporated in the very first loading; thereafter, a tool only needs to be removed when it is replaced by another one.

(2) Each tool is assumed to fit in one slot of the magazine. Removing this assumption would create considerable difficulties. For instance the physical location of the tools in the magazine would then become relevant, since adjacent slots would need to be freed in order to introduce a tool requiring more than one slot.

(3) The time needed to remove or insert each tool is constant, and is the same for all tools. This assumption is particularly crucial for the correctness of the KTNS procedure (see section 2.2), which determines the optimal tool loadings for a given job sequence. Many of our heuristic procedures, however, can easily be adapted in the case where switching times are tool dependent.

(4) Tools cannot be changed simultaneously. This is a realistic assumption in many situations, e.g., for the forging or for the PCB assembly applications mentioned above.

(5) The subset of tools required to carry out each job is fixed in advance. This assumption could be relaxed by assuming instead that, for each job, a list of subsets of tools is given, and that the job can be executed by any subset in the list; [i.e., several process plans are given for each job; see e.g., Finke and Kusiak (1987)]. Choosing the right subset would then add a new dimension (and quite a lot of complexity) to the problem.

(6) Tools do not break down and do not wear out. This assumption is justified if the tool life is long enough with respect to the planning horizon. Otherwise, one may want to lift the assumption "deterministically," e.g., by assuming that tool k is worn out after the execution of w_k jobs, for a given value of w_k. Alternatively, breakdowns and wear may also be modeled probabilistically. This would obviously result in a completely new model.

(7) The list of jobs is completely known. This assumption is realistic if the planning horizon is relatively short.

This article deals with various aspects of the tool switching problem. Section 2 contains some basic results concerning the computational complexity of this problem; in particular, we establish that the problem is already NP-hard for $C = 2$, and we present a new proof of the fact that, for each fixed job sequence, an optimal sequence of tool loadings can be found in polynomial time. In section 3, we describe several heuristics for the tool switching problem, and the performance of these heuristics on randomly generated problems is compared in section 4. Section 5 contains a summary of our results and presents perspectives for future research. The Appendix contains some graph-theoretic definitions.

2. Basic results

We present in this section some results concerning the computational complexity of the tool switching problem. We assume that the reader is familiar with the basic concepts of complexity theory [see, e.g., Nemhauser and Wolsey (1988)]. Let us simply recall here that, loosely speaking, a problem is NP-hard if it is at least as hard as the traveling salesman problem (see the Appendix).

2.1. NP-hardness results

Tang and Denardo (1988) claim that the tool switching problem is NP-hard. They do not present a formal proof of this assertion, but rather infer it from the observation that the problem can be modeled as a traveling salesman problem with variable edge lengths. Our immediate goal will be to establish the validity of two slightly stronger claims.

Consider first the following restricted version of the tool switching problem:

Input: An $M \times N$ matrix A and a capacity C.
Problem P1: Is there a job sequence for A requiring exactly M setups (i.e., no extra setups)?

Theorem 1. Problem P1 is NP-hard.

Proof. It is straightforward to check that P1 is precisely the decision version of the so-called *matrix permutation problem*, which has been extensively investigated in the VLSI design literature [see Möhring (1990) and references therein]. Several equivalent versions of the matrix permutation problem have been shown to be NP-hard [see Kashiwabara and Fujisawa (1979) and Möhring (1990)], and hence P1 is NP-hard. □

In the description of problem P1, both A and C are regarded as problem data. But, from the viewpoint of our application, it may also be interesting to consider the situation where a specific machine, with fixed capacity, has to process different batches of jobs. The matrix A can then be regarded as the sole data of the tool switching problem. This observation leads us to define the following problem, where C is now considered as a fixed parameter:

Input: An $M \times N$ matrix A.
Problem P2: Find a job sequence for A minimizing the number of setups required on a machine with capacity C.

Theorem 2. Problem P2 is NP-hard for any fixed $C \geq 2$.

Proof. Let $G = (V, E, d)$ be a graph and $H = (E, I, \delta)$ be its edge-graph (see Appendix). We consider the problem of finding a minimal-length *TS* path in H (problem P3 in the Appendix). We are now going to prove Theorem 2 by showing that this NP-hard problem can be formulated as a special case of problem P2, for any fixed $C \geq 2$. For simplicity, we first concentrate on a proof of Theorem 2 for $C = 2$.

Let $V = \{1, 2, \ldots, M\}$ and $E = \{e_1, e_2, \ldots, e_N\}$. Define an $M \times N$ matrix A, with rows associated to the nodes of G, columns associated to the edges of G, and such that:

$$a_{ij} = 1 \quad \text{if edge } e_j \text{ contains node } i,$$
$$= 0 \quad \text{otherwise.}$$

Consider now A as an instance of the tool switching problem, with capacity $C = 2$. A job sequence for this problem corresponds to a permutation of E, and hence to a *TS* path in the edge-graph of G. Also, it is easy to see that the number of tool switches between two jobs j and k, corresponding to the edges e_j and e_k of G, is:

— equal to 1 if e_j and e_k share a common node, that is, if $\delta(e_j, e_k) = 1$ in H;
— equal to 2 if e_j and e_k do not share a common node, that is, if $\delta(e_j, e_k) = +\infty$ in H.

This discussion immediately implies that an optimal job sequence for A (with capacity 2) always corresponds to a minimal-length *TS* path in H. Hence, we can solve P3 by solving P2, and this entails that P2 is NP-hard.

To see that Theorem 2 is also valid for $C > 2$, it suffices to adapt the definition of A in the previous argument, by adding C-2 rows of 1s to it; that is, A now has $(M + C\text{-}2)$ rows, and $a_{ij} = 1$ if $i \geq M + 1$. The reasoning goes through with this modification. □

2.2. *Finding the minimum number of setups for a fixed job sequence*

The tool switching problem naturally decomposes into two interdependent issues, namely:

(1) *Sequencing:* Compute an (optimal) job sequence, and
(2) *Tooling:* For the given sequence, determine what tools should be loaded in the tool magazine at each moment, in order to minimize the total number of setups required.

In their article, Tang and Denardo (1988) proved that the sequencing subproblem actually is the hard nut to crack, since the tooling problem can be solved in $O(MN)$ operations by applying a so-called Keep Tool Needed Soonest (KTNS) policy. A KTNS policy prescribes that, whenever a situation occurs where some tools should be removed from the magazine so as to make room for tools needed for the next job, then those tools which are needed the soonest for a future job should be removed last [we refer to Tang and Denardo (1988) or Bard (1988) for a more precise description].

Tang and Denardo's proof of the correctness of KTNS relies on ad hoc interchange arguments and is rather involved [as observed by Finke and Roger—see Roger (1990)—the correctness of KTNS was already established by Mattson, Gecsei, Slutz and Traiger (1970) in the context of storage techniques for computer memory, in the case where each job requires exactly one tool; their proof is similar to Tang and Denardo's]. We now look at the tooling subproblem from a different angle, and show that the problem can be modeled as a specially structured 0–1 linear programming problem, which can be solved by a greedy algorithm due to Hoffman, Kolen and Sakarovitch (1985) [see also Nemhauser and Wolsey (1988), pp. 562–573; Daskin, Jones and Lowe (1990) present another application of the same greedy algorithm in a flexible manufacturing context]. When translated in the terminology of the tool switching problem, this algorithm precisely yields KTNS. Thus, this argument provides a new proof of correctness for KTNS.

The bulk of the work in our derivation of the KTNS procedure will simply consist of reformulating the tooling problem in an appropriate form. With this goal in mind, we first introduce some new notations and terminology. For the remainder of this section, assume that the job sequence σ is fixed. Let the $M \times N$ (0, 1)-matrix P be defined by:

$$\begin{aligned} p_{ij} &= 1 \quad \text{if tool } i \text{ is required for the } j\text{th job in } \sigma, \\ &= 0 \quad \text{otherwise} \end{aligned}$$

(that is, P is obtained by permuting the columns of A according to the job sequence at hand). A tooling policy can now be described by flipping some entries of P from 0 to 1, until each column of P contains exactly C ones. If we denote by c_j the *remaining capacity* of column j, that is the quantity:

$$c_j = C - \sum_{i=1}^{M} p_{ij}$$

then a tooling policy must flip c_j entries from 0 to 1 in the jth column of P.

Let us next define a *0-block* of P as a maximal subset of consecutive zeroes in a row of P. More formally, a 0-block is a set of the form $\{(i, j), (i, j + 1), \ldots, (i, j + k)\}$, for which the following conditions hold:

(1) $1 < j \leq j + k < N$,
(2) $p_{ij} = p_{i,j+1} = \ldots = p_{i,j+k} = 0$,
(3) $p_{i,j-1} = p_{i,j+k+1} = 1$.

Intuitively, a 0-block is a maximal time interval before and after which tool i is needed, but during which it is not needed. It is easy to see that each 0-block of P is associated with an extra setup of tool i. Thus, flipping an element of P from 0 to 1 can only reduce the number of extra setups if this element belongs to a 0-block, and if all other elements of this 0-block are also flipped. In other words, only flipping *whole* 0-blocks can help reducing the number of setups.

Example 1. The matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

contains three 0-blocks, namely $\{(1, 3), (1, 4)\}$, $\{(3, 2)\}$, and $\{(3, 5)\}$. They correspond to an extra setup of tool 1 in period 5, and two extra setups of tool 3, in periods 3 and 6. Assume that the capacity is $C = 2$. Then, the number of extra setups can be minimized by flipping the first and the third 0-blocks to 1, thus resulting in the matrix:

$$T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \qquad \square$$

From the previous discussion it should now be clear that the tooling problem can be rephrased as follows: flip to 1 as many 0-blocks of P as possible, while flipping at most c_j entries in column j ($j = 1, 2, \ldots, N$).

Denote by B the number of 0-blocks in P, and, for $k = 1, 2, \ldots, B$, introduce the decision variables:

$$\begin{aligned} x_k &= 1 \quad \text{if the } k\text{th 0-block is flipped to 1,} \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

For $j = 1, 2, \ldots, N$ and $k = 1, 2, \ldots, B$, let also:

$$\begin{aligned} m_{jk} &= 1 \quad \text{if the } k\text{th 0-block ``meets'' column } j \text{ in } P, \\ &= 0 \quad \text{otherwise} \end{aligned}$$

[formally, a 0-block meets column j if it contains an element of the form (i, j), for some i; for instance, in Example 1, the first 0-block meets columns 3 and 4].

Now, the tooling problem admits the following 0–1 linear programming formulation:

$$\text{(TP)} \quad \max \sum_{k=1}^{B} x_k$$

$$\text{s.t.} \sum_{k=1}^{B} m_{jk} x_k \leq c_j, \qquad (j = 1, 2, \ldots, N)$$

$$x_k \in \{0, 1\}, \qquad (k = 1, 2, \ldots, B)$$

Assume now that the 0-blocks of P have been ordered in nondecreasing order of their "end-points": that is, the 0-blocks of P have been numbered from 1 to B in such a way that

the index of the last column met by the kth 0-block is smaller than or equal to the index of the last column met by the $(k+1)$-st 0-block, for $k = 1, \ldots, B-1$. Then, the matrix (m_{jk}) is a so-called *greedy matrix*, i.e., it does not contain the matrix $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ as a submatrix. Hoffman et al. (1985) considered the following, more general problem on an $N \times B$ greedy matrix:

$$\text{(GP)} \quad \max \sum_{k=1}^{B} b_k x_k$$

$$\text{s.t.} \sum_{k=1}^{B} m_{jk} x_k \leq c_j, \qquad (j = 1, 2, \ldots, N)$$

$$0 \leq x_k \leq d_k, \; x_k \text{ integer}, \qquad (k = 1, 2, \ldots, B)$$

where b_k, d_k $(k = 1, 2, \ldots, B)$ and $c_j (j = 1, 2, \ldots, N)$ are integers with $b_1 \geq b_2 \geq \ldots \geq b_B$. They proved that, when the matrix (m_{jk}) is greedy, problem (GP) can be solved by a greedy algorithm, in which each x_k $(k = 1, 2, \ldots, B)$ is successively taken as large as possible while respecting the feasibility constraints. Reformulating this algorithm for (TP), we see that we should successively flip 0-blocks to 1, in order of nondecreasing endpoints, as long as the remaining capacity of all columns met by the 0-block is at least one. We leave it to the reader to check that this procedure is precisely equivalent to a KTNS policy.

Remark. In a more general situation where the setup times are not identical for all tools, the tooling subproblem can still be formulated as a problem of the form (GP), where b_k is now the time required to set up the tool associated with the kth 0-block. Since the condition $b_1 \geq b_2 \geq \ldots \geq b_B$ does not generally hold for these setup times, the greedy algorithm of Hoffman et al. (1985) and KTNS are no longer valid. However, the matrix (m_{jk}), being an interval matrix, is totally unimodular [see section 3.4 and Nemhauser and Wolsey (1988) for definitions]. It follows that the tooling subproblem can still be solved in polynomial time in that case, by simply solving the linear programming relaxation of the formulation (GP).

3. Heuristics

The tool switching problem being NP-hard, and hence probably difficult to solve to optimality, we concentrate in the sequel on heuristic techniques for its solution. We propose here six basic approaches, falling into two main categories (we adopt the terminology used by Golden and Stewart (1985) for the traveling salesman problem):

— *Construction strategies*, which exploit the special structure of the tool switching problem in order to construct a single (hopefully good) job sequence (subsections 3.1 to 3.4 below);

— *Improvement strategies*, which iteratively improve a starting job sequence (subsections 3.5 and 3.6 below).

Composite strategies will be obtained by combining construction and improvement procedures. A computational comparison of the resulting procedures will be presented in section 4.

As explained in section 1, the data of our problem consist of an $M \times N$ tool-job matrix A and a capacity C. We focus on the solution of the sequencing subproblem (see subsection 2.2), since we already know that the tooling subproblem is easy to solve. Whenever we speak of the *cost* of a (partial) job sequence, we mean the minimal number of tool switches required by the sequence, as computed using KTNS.

3.1. Traveling salesman heuristics

These heuristics are based on an idea suggested by Tang and Denardo (1988). They consider a graph $G = (V, E, lb)$ (see Appendix 1 for definitions), where V is the set of jobs, E is the set of all pairs of jobs, and the length $lb(i, j)$ of edge $\{i, j\}$ is an underestimate of the number of tool switches needed between jobs i and j when these jobs are consecutively processed in a sequence. More precisely:

$$lb(i, j) = \max(|T_i \cup T_j| - C, 0),$$

where T_k is the set of tools required by job k ($k = 1, 2, \ldots, N$). Notice that, if each job requires exactly C tools (i.e., $|T_k| = C$ for all k), then $lb(i, j)$ is equal to the number of tool switches required between jobs i and j in any schedule.

Each traveling salesman (TS) path of G corresponds to a job sequence for the tool switching problem. So, as suggested by Tang and Denardo (1988), computing a short TS path in G constitutes a reasonable heuristic for the generation of a good sequence. As a matter of fact, when all jobs use full capacity, then the tool switching problem is precisely equivalent to the TS problem on G.

In our computational experiments, we have considered the following procedures for constructing a short TS path in G:

(1) *Shortest edge* heuristic: This is the heuristic used by Tang and Denardo (1988), and called "greedy feasible" in Nemhauser and Wolsey (1988); complexity: $O(N^2 \log N)$;
(2) *Nearest neighbor* heuristic with all possible starting nodes: See Golden and Stewart (1985), Johnson and Papadimitriou (1985); complexity: $O(N^3)$;
(3) *Farthest insertion* heuristic with all possible starting nodes: See Golden and Stewart (1985), Johnson and Papadimitriou (1985); complexity: $O(N^4)$;
(4) *B&B* algorithms: This is a state-of-the-art branch and bound code, which solves TS problems to optimality: see Volgenant and Jonker (1982); complexity: exponential in the worst case.

Procedures (1), (2), and (3) are well-known heuristics for the traveling salesman problem. In addition to the complexity mentioned for each procedure, an overhead of $O(MN^2)$ operations has to be incurred for the computation of the edge lengths $lb(i, j)$.

3.2. Block minimization heuristics

We describe now another way of associating a traveling salesman instance to any given instance of the tool switching problem. We first introduce a directed graph $D = (V^*, U, ub)$. Here, V^* is the set of all jobs, plus an additional node denoted by 0. Each ordered pair of nodes is an arc in U. The length $ub(i, j)$ of arc (i, j) is given by:

$$ub(i, j) = |T_i \backslash T_j|,$$

where T_k is the set of tools required by job k ($k = 1, 2, \ldots, N$); and T_0 is the empty set. In other words, $ub(i, j)$ is the number of tools used by job i but not by job j; hence, $ub(i, j)$ is an upper bound on the number of tool switches between jobs i and j, for any sequence in which i and j must be consecutively processed. If every job requires exactly C tools, then $ub(i, j) = ub(j, i) = lb(i, j)$ is equal to the number of switches between i and j. But in general, $ub(i, j)$ differs from $ub(j, i)$.

Each TS path of D finishing at node 0 defines a sequence of jobs, and the length of the path is an upper bound on the total number of switches entailed by the sequence. For reasons explained below, we refer to heuristics which attempt to construct a short TS path in D as *block minimization* heuristics. We have implemented two such heuristics:

(1) *NN block minimization*, based on a nearest neighbor heuristic with all possible starting nodes; complexity: $O(N^3)$;
(2) *FI block minimization*, based on a farthest insertion heuristic with all possible starting nodes; complexity: $O(N^4)$.

Let us mention another interesting interpretation of the block minimization approach. As in section 2.2, consider the matrix P obtained after permuting the columns of A according to a job sequence σ. We define a *1-block* of P as a set of entries, of the form $\{(i, j), (i, j + 1), \ldots, (i, j + k)\}$, for which the following conditions hold:

(1) $1 \leq j \leq j + k \leq N$,
(2) $p_{ij} = p_{i,j+1} = \ldots = p_{i,j+k} = 1$,
(3) either $j = 1$ or $p_{i,j-1} = 0$,
(4) either $j + k = N$ or $p_{i,j+k+1} = 0$.

(This definition does not exactly mimic the definition of 0-blocks, but the difference is irrelevant here). Notice that, were it not for the possibility to carry out KTNS on P, then each 1-block of P would induce a tool setup in the job sequence σ. Thus, the number of 1-blocks of P is an overestimate of the number of setups required by σ.

We leave it to the reader to check that the number of 1-blocks in P is also equal to the length of the TS path associated with σ in D (and finishing at node 0). So, finding a shortest TS path in D is equivalent to determining a permutation of the columns of A which minimizes the number of 1-blocks in the permuted matrix. This observation is essentially due to Kou (1977). Kou (1977) also proved that finding a permutation which minimizes the number of 1-blocks is NP-hard (our proof of Theorem 2 establishes the same result). This justifies the use of heuristics in our block minimization approach.

3.3. Greedy heuristics

One of the obvious drawbacks of the heuristics described in sections 3.1 and 3.2 is that they do not take a whole job sequence into account when estimating the number of tool switches required between pairs of jobs. For instance, $lb(i, j)$ is in general only a lower bound on the actual number of switches between jobs i and j, and this lower bound can sometimes be a quite poor estimate of the actual value. An extreme case would arise when no job requires more than $C/2$ tools; then, $lb(i, j) = 0$ for each pair (i, j), and any traveling salesman heuristic based on these edge-lengths picks a random job sequence! Similarly, $ub(i, j)$ can also be a rough upper bound on the number of switches required. In order to alleviate this difficulty, we propose now the following (*simple*) *greedy* heuristic:

Step 1. Start with the partial job sequence $\sigma = (1)$; let $Q = \{2, 3, \ldots, N\}$.
Step 2. For each job j in Q, let $c(j)$ be the cost of the partial sequence (σ, j) (i.e., the number of tool switches entailed by this partial sequence, disregarding the remaining jobs).
Step 3. Let i be a job in Q for which $c(i) = \min_{j \in Q} c(j)$; let $\sigma := (\sigma, i)$ and $Q := Q \backslash \{i\}$.
Step 4. If Q is not empty, then repeat Step 2; else, stop with the complete sequence σ.

Greedy runs in time $O(MN^3)$, since it requires $O(N^2)$ applications of the KTNS procedure (in Step 2). Its empirical performance can be slightly improved by taking advantage of the fact that all the partial sequences considered in Step 2 share the same initial segment.

Of course, there is no mandatory reason to select job 1 first in Step 1 of greedy, rather than any other job. This observation suggests to consider the following, more elaborate *multiple-start greedy* heuristic: run N times greedy, once for each initial sequence $\sigma = (j)$ $(j = 1, 2, \ldots, N)$, and retain the best complete sequence found. This heuristic clearly dominates greedy, in terms of the quality of the job sequence that it produces. Its worst-case complexity is $O(MN^4)$.

As a final note on this approach, it may be interesting to observe that, if each job requires exactly C tools, then multiple-start greedy is identical to the TS nearest neighbor heuristic (section 3.1) or to the *NN* block minimization heuristic (section 3.2).

3.4. Interval heuristic

In order to motivate our next heuristic, let us first consider a special situation: Assume that the matrix P arising by permuting the columns of A according to some sequence σ

has precisely one 1-block in each row. In other words, the ones in each row of P occur consecutively. When this is the case we say that A is an *interval matrix* [or that A has the *consecutive ones property*; see e.g., Fulkerson and Gross (1965), Booth and Lueker (1976), Nemhauser and Wolsey (1988)]. Then, the job sequence σ requires only one setup per tool, and is obviously optimal.

Thus, every $M \times N$ interval matrix admits an optimal sequence with M setups. Moreover, given an aribtrary matrix A, one can decide in time $O(MN)$ whether A is an interval matrix, and, in the affirmative, one can find within the same time bound a sequence entailing M setups for A [Booth and Lueker (1976)] (notice that this does not contradict Theorem 1; by applying KTNS, a sequence with m setups can sometimes be found for noninterval matrices]. On the other hand, it is by no means clear that any of the heuristics described in sections 3.1, 3.2, or 3.3 would find an optimal job sequence for an interval matrix.

These observations suggest the implementation of the following *interval heuristic*. The heuristic simultaneously builds a "large" interval submatrix of A, and computes an optimal job sequence for the submatrix. This sequence is the solution returned by the heuristic. More precisely:

Step 1. Initialize $I = \{\}$, $i = 1$.
Step 2. Determine whether the submatrix of A consisting of the rows with index in $I \cup \{i\}$ is an interval matrix; if so, then let $I := I \cup \{i\}$ and let σ be an optimal job sequence for the submatrix; else, continue.
Step 3. If $i < M$, then let $i := i + 1$ and go to Step 2; else, continue.
Step 4. Return the last job sequence found; stop.

The interval heuristic has the attractive property that it produces an optimal job sequence for every interval matrix. The complexity of the heuristic is $O(MN)$ if the algorithm by Booth and Lueker (1976) is used. In our implementation, we have used a slower, but simpler recognition algorithm for interval matrices, due to Fulkerson and Gross (1965).

In the following subsections, we concentrate on improvement strategies. The input for each procedure is some initial job sequence σ, which we subsequently attempt to improve in an iterative way.

3.5. 2-opt strategies

This class of strategies is based on an idea that has been widely used for other combinatorial optimization problems: Given a sequence σ, try to produce a better sequence by exchanging two jobs in σ (if i is the kth job and j is the pth job in σ, then *exchanging* i and j means putting i in pth position and j in kth position). We have considered two versions of this basic approach. The first one, called *global 2-opt*, can be described as follows:

Step 1. Find two jobs i and j whose exchange results in an improved sequence; if there are no such jobs, then return σ and stop; else, continue.
Step 2. Exchange i and j; call σ the resulting sequence; repeat Step 1.

Global 2-opt has been proposed by Bard (1988) for the tool switch problem. Notice that each execution of Step 1 requires $O(N^2)$ applications of KTNS, i.e., $O(MN^3)$ operations.

But the number of potential executions of this step does not appear to be trivially bounded by a polynomial in N and M [contrary to what is claimed by Bard (1988)]. In order to reduce the computational effort by iteration of global 2-opt, the following *restricted 2-opt* procedure can also be considered:

Step 1. Find two *consecutive* jobs in σ, say the kth and $(k + 1)$-st ones, whose exchange results in an improved sequence; if there are no such jobs, then return σ and stop.
Step 2. Exchange the jobs found in Step 1; call σ the resulting sequence; repeat Step 1.

The complexity of Step 1 in restricted 2-opt is $O(MN^2)$. This exchange strategy has also been proposed by Finke and Roger [see Roger (1990)].

3.6. Load-and-optimize strategy

Consider again a job sequence σ and the matrix P obtained by permuting the columns of A according to σ. Applying KTNS to P results in a new matrix T, each column of which contains exactly C ones (the jth column of T describes the loading of the tool magazine while the jth job in σ is being processed). Suppose now that we look at T as defining a new instance of the tool switching problem (with capacity C). If we can find for T a better sequence than σ, then this sequence will obviously be a better sequence than σ for the original matrix A as well. On the other hand, the problem instance (T, C) is a little bit easier to handle than the instance (A, C). Indeed, since each column of T contains C ones, the tool switching problem (T, C) can be reformulated as a TS problem, as explained in sections 3.1, 3.2, and 3.3. These observations motivate our *load-and-optimize* strategy:

Step 1. Permute the columns of A according to σ and apply KTNS; call T the resulting matrix.
Step 2. Compute an optimal sequence σ' for the tool switching instance (T, C).
Step 3. If σ' is a better sequence than σ for A, then replace σ by σ' and repeat Step 1; else return σ and stop.

From a practical viewpoint, we have found it easier to slightly alter this basic strategy, in the following way. In Step 2, rather than computing an optimal sequence for T (which is computationally demanding), we simply use the farthest insertion heuristic to produce a good sequence σ' (as in section 3.1). On the other hand, in Step 3, we accept the new sequence σ' even if it entails the same number of setups as σ. We only stop when 10 iterations of the procedure have been executed without producing a strictly improved sequence. In the sequel, we also refer to this variant as "load-and-optimize."

4. Computational experiments

4.1. Generation of problem instances

We tested our heuristics on 160 random instances of the tool switching problem. Of course, tool-job matrices occurring in practice may have characteristics not present in the ones

we generated. For instance, as pointed out by an anonymous referee, realistic matrices are likely to display interrow and intercolumn correlations, as well as "tool clusters." However, in the absence of real-world data or even of detailed statistical information about these, we decided to follow a procedure similar to the one proposed by Tang and Denardo (1988) in generating our test problems.

Each random instance falls into one of 16 *instance types*, characterized by the size (M, N) of the tool-job matrix and by the value C of the capacity. Accordingly, we denote the type of an instance by a triple (M, N, C). There are 10 instances of each type. The tool-job matrices are $M \times N$ matrices, where (M, N) is either (10, 10), (20, 15), (40, 30), or (60, 40). For each size (M, N), we also define a pair (min, max) of parameters with the following interpretation:

— min = lower bound on the numbr of tools per job,
— max = upper bound on the number of tools per job.

The specific values of these parameters are displayed in table 1.

For each problem size (M, N), 10 random matrices A were generated. For each $j = 1, 2, \ldots, N$, the jth column of A was generated as follows. First, an integer t_j was drawn from the uniform distribution over [min, max]: this number denotes the number of tools needed for job j, i.e., the number of 1s in the jth column of A. Next, a set T_j of t_j distinct integers were drawn from the uniform distribution over [1, M]: these integers denote the tools required by job j, i.e., $a_{kj} = 1$ if and only if k is in T_j. Finally, we checked whether $T_j \subseteq T_i$ or $T_i \subseteq T_j$ held for any $i < j$. If any of these inclusions was found to hold, then the previous choice of T_j was cancelled, and a new set T_j was generated [Tang and Denardo (1988) and Bard (1988) have observed that any column of A contained in another column can be deleted without affecting the optimal solution of the problem; thus, we want to make sure that our problem instances actually involve N columns, and cannot be reduced by this simple trick]. Notice that this generation procedure does not a priori prevent the occurrence of null rows in the matrix. In practice, only two of the 40 matrices that we generated contained null rows [these were two (20, 15) matrices, containing, respectively, one and three null rows].

A problem instance of type (M, N, C) is now obtained by combining an $M \times N$ tool-job matrix A with one of the four capacities C_1, C_2, C_3, and C_4 displayed in table 2.

We will see that the performance of some heuristics strongly depends on the value of the ratio max/C. We call *sparse* those problem instances for which max/C is small, and *dense* those for which the ratio is close to 1. Notice, in particular, that all instances of

Table 1. Parameter values min and max.

Problem Size	Min	Max
(10,10)	2	4
(20,15)	2	6
(40,30)	5	15
(60,40)	7	20

Table 2. Problem sizes and tool magazine capacities.

Problem Size	C_1	C_2	C_3	C_4
(10,10)	4	5	6	7
(20,15)	6	8	10	12
(40,30)	15	17	20	25
(60,40)	20	22	25	30

type (M, N, C_1) have $\max/C_1 = 1$. Varying the capacity as indicated in table 2 will allow us to examine the behavior of our heuristics under different sparsity conditions. Let us mention here that, according to the empirical observation of many real-world systems described by Förster and Hirt (1989), sparse instances are probably more "realistic" than dense ones. But of course, this conclusion is very much system-dependent.

4.2. Computational results

All heuristics described in section 3 have been implemented in Turbo Pascal and tested on the problem instances described above. The experiments were run on an AT personal computer equipped with an 80286 microprocessor and an additional 80287 coprocessor. Since our primary goal was to compare the quality of the solutions produced by the heuristics, no systematic attempts were made to optimize the running time of the codes. Accordingly, we will not report here on precise computing times, but simply give some rough indication of the relation between the times required by the various methods.

The performance of heuristic H on problem instance I is measured in terms of "percentage above the best solution found," namely, by the quantity:

$$\delta_H(I) = \left(\frac{H(I) - \text{best}(I)}{\text{best}(I)} \right) * 100,$$

where $H(I)$ is the number of tool setups required by the job sequence produced by heuristic H; and best(I) is the number of setups required by the best sequence found by *any* of our heuristics.

For information, table 3 indicates the evolution of best(I) as a function of the problem type [average of best (I) over all 10 instances of each type]. All subsequent tables (tables 4, 5, and 6) report averages and (in brackets) standard deviations of $\delta_H(I)$ over all instances I of a given type.

Table 4 compares the behavior of the four traveling salesman heuristics described in section 3.1. We will see later that TS heuristics perform best on dense instances, and tend to behave very badly on sparse instances. Therefore, we limit ourselves here to a comparison of these heuristics on the densest instances, that is, those instances where $C = C_1 = \max$.

From table 4, it appears that on average, and mostly for large instances, farthest insertion yields better solutions than the other TS heuristics. Farthest insertion is also a very fast heuristic, which produces solutions in a matter of seconds (about 30 seconds for the largest instances). The shortest edge and nearest neighbor heuristics are even faster, but

Table 3. Average of best(I).

Problem Size	Tool magazine capacity			
	C_1	C_2	C_3	C_4
(10,10)	13.2	11.2	10.3	10.1
(20,15)	26.5	21.6	20.0	19.6
(40,30)	113.6	95.9	76.8	56.8
(60,40)	211.6	189.7	160.5	127.4

Table 4. Average (and standard deviation) of $\delta_H(I)$ for the traveling salesman procedures.

Heuristic	(10,10, C = 4)	(20,15, C = 6)	(40,30, C = 15)	(60,40, C = 20)
Shortest edge	12.4 (6.8)	23.9 (9.8)	20.3 (3.1)	18.8 (3.4)
Farthest insertion	12.1 (9.8)	15.5 (8.6)	9.4 (5.3)	6.9 (2.7)
Nearest neighbor	13.7 (7.8)	19.8 (7.7)	21.0 (6.0)	18.9 (3.5)
Branch-and-bound	12.6 (4.6)	16.2 (5.8)	12.4 (4.3)	10.9 (2.9)

farthest insertion presents in our view the best quality versus efficiency trade-off. Thus, we will select farthest insertion as our "winner" among TS heuristics, and no longer report on the other TS heuristics in the sequel.

A similar comparison between the two block minimization heuristics presented in section 3.2 would lead to similar conclusions. Here again, FI is slightly better and slightly slower than NN. In the remainder of this section, we only report on the performance of FI, and no longer of NN.

Table 5 displays the performance of "constructive" and "improvement" heuristics over our complete sample of problem instances. The results (averages and standard deviations) for each heuristic are given in different columns. The results presented under the labels "2-opt" or "load-and-optimize" have been obtained by first picking a random job sequence, and then applying the corresponding improvement strategies to it. The columns labeled "random" provide, for the sake of comparison, the number of tool setups entailed by the initial random job sequence.

Let us now try to sketch some of the conclusions that emerge from this table. Consider first the case of dense instances, that is, the instances of type (10, 10, 4), (20, 15, 6), (40, 30, 15), and (60, 40, 20). As the size of these instances increases, the ranking of the solutions delivered by the various heuristics seems to become more or less stable. Namely, multiple-start greedy and global 2-opt produce (on the average) the best results. Next comes a group made up of farthest insertion, FI block minimization and simple greedy, which usually yield solutions of slightly lower quality. Finally, the worst solutions are produced by load-and-optimize, restricted 2-opt, and interval (and, as expected, the random procedure).

We get a somewhat different ranking of the heuristics when we look at sparse instances. Consider, e.g., the instances of type (10, 10, 7), (20, 15, 12), (40, 30, 25), and (60, 40, 30). Multiple-start greedy, global 2-opt, simple greedy, and FI block minimization remain, in that order, the best heuristics. But farthest insertion performs now almost as badly as the random procedure! As a matter of fact, for larger instances, it appears that the performance of farthest insertion deteriorates very systematically as sparsity increases. This

Table 5. Average (and standard deviation) of $\delta_H(I)$.

(M,N,C)	Farthest Insertion	FI Block Minimization	Simple Greedy	Multiple-Start Greedy	Interval	Restricted 2-opt	Global 2-opt	Load-and-Optimize	Random
(10,10,4)	12.1 (9.8)	14.3 (7.7)	12.3 (6.3)	4.6 (3.8)	22.6 (12.2)	26.0 (7.7)	8.7 (4.7)	5.8 (5.3)	41.2 (18.9)
(10,10,5)	19.0 (7.8)	13.6 (7.6)	8.1 (6.0)	3.7 (4.6)	14.1 (14.1)	24.3 (10.1)	7.4 (7.1)	10.1 (7.2)	33.8 (16.2)
(10,10,6)	17.8 (10.8)	9.7 (6.4)	5.7 (4.7)	2.9 (4.4)	9.7 (11.8)	18.3 (7.7)	3.0 (4.6)	6.7 (4.4)	26.3 (9.1)
(10,10,7)	11.7 (10.3)	3.9 (4.8)	1.0 (3.0)	0.0 (0.0)	3.0 (6.4)	9.8 (7.5)	0.0 (0.0)	3.0 (6.4)	13.8 (7.9)
(20,15,6)	15.5 (8.6)	12.0 (4.2)	13.7 (7.0)	4.6 (3.5)	25.7 (9.7)	33.6 (7.2)	10.0 (4.3)	12.3 (6.8)	45.9 (8.8)
(20,15,8)	37.3 (10.8)	13.9 (8.4)	11.0 (7.3)	4.6 (3.0)	20.4 (9.2)	35.7 (10.8)	9.7 (4.1)	23.8 (8.5)	42.2 (11.8)
(20,15,10)	30.5 (5.8)	8.3 (6.2)	5.6 (4.3)	1.5 (2.3)	10.4 (8.2)	24.3 (9.2)	6.4 (7.3)	25.6 (11.7)	30.1 (12.3)
(20,15,12)	15.3 (5.5)	2.1 (3.5)	1.0 (2.1)	0.0 (0.0)	3.5 (5.0)	13.6 (8.3)	1.0 (2.0)	16.6 (9.6)	18.1 (11.3)
(40,30,15)	9.4 (5.3)	8.8 (4.4)	11.4 (4.8)	6.2 (3.1)	30.5 (4.3)	30.3 (5.0)	6.0 (4.0)	16.6 (5.3)	42.9 (6.1)
(40,30,17)	16.3 (7.5)	9.4 (3.8)	9.8 (3.5)	5.5 (2.2)	31.2 (5.4)	31.0 (4.6)	4.5 (3.3)	27.5 (4.3)	44.6 (6.4)
(40,30,20)	33.8 (9.1)	12.1 (3.6)	9.8 (4.2)	3.2 (2.0)	30.4 (6.0)	33.0 (6.6)	6.0 (2.9)	35.1 (6.4)	45.5 (8.9)
(40,30,25)	39.4 (6.6)	15.0 (2.7)	8.3 (4.9)	2.6 (2.3)	27.8 (6.6)	34.5 (7.4)	6.1 (3.7)	37.8 (7.0)	40.5 (7.1)
(60,40,20)	6.9 (2.7)	9.7 (2.4)	10.2 (2.6)	5.8 (1.5)	30.6 (2.7)	25.8 (3.8)	4.8 (2.4)	20.0 (3.8)	37.1 (3.6)
(60,40,22)	9.9 (2.7)	8.7 (2.6)	7.9 (3.1)	3.3 (1.7)	29.3 (4.1)	25.4 (2.9)	3.7 (2.6)	25.4 (4.1)	36.5 (3.5)
(60,40,25)	21.8 (5.7)	10.5 (3.1)	8.2 (2.8)	2.8 (2.0)	30.2 (3.6)	29.7 (3.0)	2.1 (1.9)	35.5 (4.3)	38.0 (3.6)
(60,40,30)	36.7 (4.0)	13.1 (3.7)	6.5 (2.4)	1.7 (1.4)	28.8 (3.4)	30.1 (3.3)	4.5 (2.7)	36.7 (4.4)	37.6 (3.8)

behavior is matched by all other TS heuristics (shortest edge, nearest neighbor, and B&B). It can be explained by observing that, for sparse instances, the bounds $lb(i, j)$ tend to be poor estimates of the number of switches required between jobs i and j (see sections 3.1 and 3.3).

Our conclusion at this point would be that, if we are only concerned with the quality of the solution produced by each heuristic, then multiple-start greedy and global 2-opt come out the winners, while simple greedy and FI block minimization are good contenders. For dense problems, farthest insertion also is a very good technique.

This first picture becomes more nuanced when we also take computing time into account. Indeed, the various heuristics run at very different speeds. For instance, solving an instance of type (10, 10, 4) takes about 0.30 seconds by farthest insertion, FI block minimization, or by simple greedy; 2 seconds by global 2-opt; and 3 seconds by multiple-start greedy. More strikingly, the instances of type (60, 40, 20) require about 30 seconds by farthest insertion or by FI block minimization; 1.5 minutes by simple greedy; 30 minutes by global 2-opt; and 1 hour by multiple-start greedy (these times are rather stable, for a given method, over all instances of the same type). Even though some of these procedures could certainly be accelerated by implementing them more carefully, it is probably safe to say that the first three heuristics are fast, while the latter two are computationally more demanding. Therefore, for those applications where a solution of high quality has to be found quickly, FI block minimization and simple greedy seem to be perfectly adequate procedures (as well as farthest insertion, for dense instances). On the other hand, when computing time does not matter too much, and the thrust is instead on the quality of the solution, multiple-start greedy and global 2-opt could be considered.

Table 6 contains the results of our experiments with composite heuristics. The idea here is to quickly compute a good job sequence using one of the constructive heuristics, and

Table 6. Average (and standard deviation) of $\delta_H(I)$ for composite heuristics.

(M,N,C)	Farthest Insertion	FI Block Minimization	Simple Greedy	Interval	Global 2-opt
(10,10,4)	5.0 (5.5)	8.7 (6.9)	5.4 (3.6)	6.9 (4.5)	8.7 (4.7)
(10,10,5)	8.3 (5.3)	7.3 (7.1)	5.3 (5.5)	3.6 (4.4)	7.4 (7.1)
(10,10,6)	4.9 (4.9)	2.9 (4.4)	1.9 (3.8)	2.0 (4.0)	3.0 (4.6)
(10,10,7)	2.0 (4.0)	1.0 (3.0)	0.0 (0.0)	1.0 (3.0)	0.0 (0.0)
(20,15,6)	6.3 (5.1)	6.4 (3.8)	6.6 (3.8)	4.7 (2.9)	10.0 (4.3)
(20,15,8)	12.3 (6.4)	6.2 (4.9)	7.1 (3.4)	8.9 (5.5)	9.7 (4.1)
(20,15,10)	5.0 (5.5)	3.6 (4.0)	3.9 (3.0)	3.9 (5.2)	6.4 (7.3)
(20,15,12)	1.5 (3.2)	0.0 (0.0)	0.5 (1.5)	1.0 (3.0)	1.0 (2.0)
(40,30,15)	2.5 (3.1)	2.8 (2.0)	5.3 (4.3)	5.3 (3.1)	6.0 (4.0)
(40,30,17)	3.1 (1.3)	3.0 (2.5)	5.0 (2.4)	6.5 (2.6)	4.5 (3.3)
(40,30,20)	6.6 (4.1)	3.4 (2.1)	5.3 (2.7)	6.6 (2.9)	6.0 (2.9)
(40,30,25)	7.7 (3.0)	3.9 (2.2)	4.6 (3.4)	9.1 (5.1)	6.1 (3.7)
(60,40,20)	1.5 (1.6)	2.2 (1.8)	5.2 (1.5)	5.0 (1.5)	4.8 (2.4)
(60,40,22)	2.0 (2.4)	2.6 (2.1)	2.5 (2.3)	2.7 (2.0)	3.7 (2.6)
(60,40,25)	3.7 (1.7)	2.7 (2.0)	2.3 (2.5)	4.1 (3.4)	2.1 (1.9)
(60,40,30)	3.2 (2.7)	1.6 (2.0)	2.4 (2.0)	3.7 (1.5)	4.5 (2.7)

to subsequently improve it by relying on some improvement strategy. In view of our previous experiments, we consider five ways to produce an initial solution (namely, by farthest insertion, FI block minimization, simple greedy, interval, and by a random procedure), and we choose global 2-opt as improvement strategy.

We see from table 6 that, for dense instances, farthest insertion usually provides a very good initial solution, while FI block minimization always performs among the best for sparser instances. But in fact, surprisingly enough, all initialization procedures for global 2-opt (including the random one) come extremely close to each other, in terms of the quality of the solution produced. Also, their running times do not differ significantly.

5. Summary and conclusions

In this article we analyze a problem occurring in certain flexible manufacturing environments. This problem was described by Tang and Denardo (1988) and Bard (1988), and is here referred to as the tool switching problem. Links between this problem and other well-studied combinatorial optimization problems are established here for the first time: The matrix permutation problem (Möhring 1990), optimization with greedy constraint matrices (Hoffman et al. 1985), the block minimization problem (Kou 1977), recognition of interval matrices (Fulkerson and Gross 1965), etc.

We prove that the tool switching is NP-hard already for a fixed capacity $C \geq 2$. On the other hand, when the job sequence is fixed, we show that the problem of determining the optimal sequence of tool loadings can be modeled as a specially structured 0–1 linear programming problem which can be solved in polynomial time. This provides an alternative proof of the correctness of the KTNS procedure.

In view of the complexity of the tool switching problem, several heuristic solution approaches are introduced. For instance, by modeling the tool switching problem as a shortest Hamiltonian path problem, well-known heuristics for the latter problem become available. This was already noticed by Tang and Denardo (1988), but the connection is here exploited in a systematic way by our traveling salesman and block minimization heuristics. Other new approaches include a greedy-type heuristic and a heuristic based on the recognition of interval matrices. In addition, some improvement strategies are also proposed.

The performance of these heuristics is tested by applying them to a number of randomly generated problem instances. It turns out that the density of the tool job matrix affects the quality of the solutions obtained by some heuristics. In particular, the TSP-based heuristics of the type proposed by Tang and Denardo (1988) perform poorly for sparse instances. The simple greedy heuristic performs well (in comparison with the other heuristics) on the sparse instances considered here, both in terms of quality of the solution found and of running time by the heuristic. The reason for this relatively good behavior may be sought in the fact that greedy uses more information than, e.g., the traveling salesman heuristics (see section 3.3).

In order to formualte a more accurate judgment on the quality of the heuristics tested it would have been desirable to know tight and easily computed lower bounds on the cost of an optimal job sequence. The knowledge of such lower bounds would also be a prerequisite for the development of an exact optimization procedure (e.g., of the branch-and-

bound type) for the tool switching problem. At this moment, unfortunately, we do not have very good lower-bounding procedures for the problem. Some of the research directions which may be worth pursuing in this regard are described in Crama, Kolen, Oerlemans and Spieksma (1991). There, we identify some special structures, whose presence in the tool-job matrix implies the necessity of extra setups in any sequence. Also, some valid inequalities are given which may improve the mixed 0–1 linear programming formulation of the tool switching problem proposed by Tang and Denardo (1988) (the continuous relaxation of this formulation provides extremely weak bounds). The possibility of deriving good lower bound using Lagrangian relaxation is also worth exploring [see Bard (1988)].

Acknowledgments

The first author was partially supported in the course of this research by AFOSR grants 89-0512 and 90-0008 and an NSF grant STC88-09648 to Rutgers University.

Appendix. Graph-theoretic definitions

In this article, a *graph* G is a triple of the form (V, E, d), where:

- V is a finite set; the elements of V are the *nodes* of G;
- E is a set of pairs of nodes, called *edges*;
- d is a function which assigns a nonnegative *length* to each pair of nodes; we assume that $d(u, v) = +\infty$ when $\{u, v\}$ is not an edge.

A *path* in a graph is a sequence of nodes, i.e., a permutation of a subset of V. A *traveling salesman path* (or *TS path*) is a permutation of V. The length of a path $(u_1, \ldots, u_k)$ is by definition:

$$d(u_1, u_2) + d(u_2, u_3) + \ldots + d(u_{k-1}, u_k).$$

Notice, in particular, that the length of such a path is infinite if some pair $\{u_i, u_{i+1}\}$ is not an edge of the graph.

The *traveling salesman problem* on a graph G can be stated as follows: Find a TS path of minimal length in G.

With a graph $G = (V, E, d)$, we can associate another graph $H = (E, I, \delta)$, called the *edge-graph* of G, and defined as follows:

- Each node of H is an edge of G;
- A pair $\{e, f\}$, with $e, f \in E$, is an edge of H if and only if the edges e and f share a common node in G;
- $\delta(e, f) = 1$ if $\{e, f\}$ is an edge of H, and $\delta(e, f) = +\infty$ otherwise.

Observe that, in an edge-graph, every TS path has length either $|E| - 1$ or $+\infty$. Consider now the restriction of the traveling salesman problem to edge-graphs, that is:

Input: A graph G.
Problem P3. Find a TS path of minimal length in the edge-graph of G.

Equivalently, P3 asks whether there exists a TS path of finite length in the edge-graph of G. Bertossi (1981) proved that this problem is NP-hard .

We also deal in this article with *directed graphs*. A directed graph is a triple (V, U, d), where V is defined as for a graph, and:

— U is a set of ordered pairs of nodes, called *arcs*; i.e., $U \subset V \times V$;
— d is a (nonnegative) length function defined on $V \times V$, with the property that $d(u, v) = +\infty$ when (u, v) is not an arc.

So, in a directed graph, $d(u, v)$ may differ from $d(v, u)$. The definitions of a TS path and of the TS problem extend in a straightforward way for directed graphs.

References

Bard, J.F. and Feo, T.A., "The Cutting Path and Tool Selection Problem in Computer Aided Process Planning," *Journal of Manufacturing Systems*, Vol. 8, No. 1, pp. 17–26 (1989).

Bard, J.F., "A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine," *IIE Transactions*, Vol. 20, No. 4, pp. 382–391 (1988).

Bertossi, A.A., "The Edge Hamiltonian Path Problem is NP-Complete," *Information Processing Letters*, Vol. 13, No. 4,5, pp. 157–159 (1981).

Blazewicz, J., Finke, G., Haupt, R. and Schmidt, G., "New Trends in Machine Scheduling," *European Journal of Operational Research*, Vol. 37, pp. 303–317 (1988).

Booth, K.S. and Lueker, G.S., "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms," *Journal of Computer and System Sciences*, Vol. 13, pp. 335–379 (1976).

Crama, Y., Kolen, A.W.J., Oerlemans, A.G. and Spieksma, F.C.R., "Minimizing the Number of Tool Switches on a Flexible Machine," Research Memorandum RM 91.010, University of Limburg, Maastricht, The Netherlands (1991).

Daskin, M., Jones, P.C. and Lowe, T.J., "Rationalizing Tool Selection in a Flexible Manufacturing System for Sheet-Metal Products," *Operations Research*, Vol. 38, No. 6, pp. 1104–1115 (1990).

Finke, G. and Kusiak, A., "Models for the Process Planning Problem in a Flexible Manufacturing System," *International Journal of Advanced Manufacturing Technology*, Vol. 2, pp. 3–12 (1987).

Förster, H.-U. and Hirt, K., "Entwicklung einer Handlungsanleitung zur Gestaltung von Produktionsplanungs- und -Steuerungskonzepten beim Einsatz Flexibler Fertigungssysteme," Schlußbericht zum Forschungsvorhaben, Nr. S 172, Forschungsinstitut für Rationalisierung, Rheinisch-Westfälischen Technischen Hochschule, Aachen (1989).

Fulkerson, D.R. and Gross, D.A., "Incidence Matrices and Interval Graphs," *Pacific Journal of Mathematics*, Vol. 15, No. 3, pp. 835–855 (1965).

Golden, B.L. and Stewart, W.R., "Empirical Analysis of Heuristics," in *The Traveling Salesman Problem*. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.), John Wiley & Sons, Chichester, U.K., pp. 207–249 (1985).

Gray, A.E., Seidmann, A. and Stecke, K.E., "Tool Management in Automated Manufacturing: Operational Issues and Decision Problems," Working Paper CMOM 88-03, Simon Graduate School of Business Administration, University of Rochester, Rochester, New York (1988).

Hirabayashi, R., Suzuki, H. and Tsuchiya, N., "Optimal Tool Module Design Problem for NC Machine Tools," *Journal of the Operations Research Society of Japan*, Vol. 27, No. 3, pp. 205–228 (1984).

Hoffman, A.J., Kolen, A.W.J. and Sakarovitch, M., "Totally Balanced and Greedy Matrices," *SIAM Journal on Algebraic and Discrete Methods*, Vol. 6, No. 4, pp. 721–730 (1985).

Johnson, D.S. and Papadimitriou, C.H., "Computational Complexity," in *The Traveling Salesman Problem*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.), John Wiley & Sons, Chichester, U.K., pp. 37–85 (1985).

Kashiwabara, T. and Fujisawa, T., "NP-Completeness of the Problem of Finding a Minimum-Clique-Number Interval Graph Containing a Given Graph as a Subgraph," in *Proceedings of the 1979 International Symposium on Circuits and Systems*, pp. 657–660 (1979).

Kiran, A.S. and Krason, R.J., "Automated Tooling in a Flexible Manufacturing System," *Industrial Engineering*, pp. 52–57 (April 1988).

Kou, L.T., "Polynomial Complete Consecutive Information Retrieval Problems," *SIAM Journal on Computing*, Vol. 6, pp. 67–75 (1977).

Mattson, R., Gecsei, J., Slutz, D.R. and Traiger, I.L., "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, Vol. 9, No. 2, pp. 78–117 (1970).

Möhring, R.H., "Graph Problems Related to Gate Matrix Layout and PLA Folding," in *Computational Graph Theory*, G. Tinhofer et al. (eds.), Springer-Verlag, Wien, pp. 17–51 (1990).

Nemhauser, G.L. and Wolsey, L.A., *Integer and Combinatorial Optimization*, John Wiley & Sons, New York (1988).

Roger, C., "La Gestion des Outils sur Machines à Commande Numérique," Mémoire DEA de Recherche Opérationnelle, Mathématiques et Informatique de la Production, Université Joseph Fourier, Grenoble (1990).

Tang, C.S. and Denardo, E.V., "Models Arising from a Flexible Manufacturing Machine, Part I: Mimimization of the Number of Tool Switches," *Operations Research*, Vol. 36, No. 5, pp. 767–777 (1988).

Volgenant, T. and Jonker, R., "A Branch and Bound Algorithm for the Symmetric Traveling Salesman Problem Based on 1-Tree Relaxation," *European Journal of Operational Research*, Vol. 9, pp. 83–89 (1982).