IIE Transactions (2004) **36**, 95–110
Copyright © “IIE”
ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170490245351

Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes

MICHAL TZUR[1]* and AVRI ALTMAN[2]

[1]*Industrial Engineering Department, Tel Aviv University, Tel Aviv, Israel*
E-mail: tzur@eng.tau.ac.il
[2]*E-mail: avri@email.com*

Received March 2002 and accepted May 2003

The problem addressed in this paper is the tool switching problem for an automated manufacturing environment, when each tool may occupy more than one slot of the tool magazine. A machine processes parts automatically by using a limited capacity tool magazine. Providing a tool that is needed for a certain processing operation is not in the magazine, a tool switch must occur before the job can be processed, a time/cost consuming operation. To solve this problem, one has to decide three types of decisions, namely, how to select the jobs’ sequence (machine loading), which tools to switch before each processing operation (tool loading) and where to locate each tool in the magazine (slot loading). We present an integer programming formulation for the problem and suggest a heuristic procedure to obtain a solution. Our heuristic is partly a generalization of previously suggested approaches to the first two decision types, but it is mainly oriented towards answering the third decision type. The unified problem has not been addressed previously in the literature. We present a numerical study that demonstrates the efficiency of our procedure.

1. Introduction and literature review

The concept of flexibility has become a key consideration in the design, operation and management of manufacturing systems. In a survey by Sethi and Sethi (1990) they define flexibility in manufacturing as being able to reconfigure manufacturing resources in order to efficiently produce different products of acceptable quality. In addition, they note that at least 50 different terms for various types of flexibility can be found in the manufacturing literature.

A FMS (Flexible Manufacturing System) consists of a number of numerically controlled machines, linked by an automated material handling device, that perform the operations required to manufacture parts. The tools that are required by these operations are stored in a limited capacity tool magazine attached to each machine. An automated tool interchanging device enables the machine to interchange tools very quickly. The *Tool Management* problem is a generic name for various problems concerned with tool sequencing/loading/switching and setups minimization.

FMS environments are very common in the printed circuit board (PCB) assembly industry and in metal-based industries. Bard (1988) mentions a problem in the electronics industry in which an automated placement machine produces several types of PCBs. 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 another board type. Exchanging feeders is a time consuming operation and it is therefore important to determine a production sequence for the board types that minimizes the number of “feeder setups”. Identifying the feeders with tools constitutes an instance of the *job sequencing and tool loading problem* described below.

An example of a FMS in a metal-based industry can be found in Tatikonda and Stietz (1994). The FMS consists of three machining centers, each of which has a tool magazine with a capacity of 320 tools, which achieves a time of 4.5 seconds for the tool selection and 3 seconds for the tool changes. The capacity of 320 tools allows for tool redundancy thereby minimizing downtime due to tool unavailability.

An essential feature of a FMS is the fast tool interchanging capability. This capability allows us to reduce the number of costly setup changes while producing with the tools available in the magazine. When it becomes necessary to add tools to the tool magazine to allow new operations, the machine sometimes has to be shut down while the tools are replaced, after which the machine may resume production. The performance of a FMS may therefore be considerably improved by reducing the occurrences of these setups. The

*Corresponding author is currently visiting the IE/MS Department at Northwestern University.

problem becomes especially crucial when the time needed to change a tool is significant with respect to the processing time of the parts, or when many small batches of different parts must be processed in succession. Jain *et al.* (1996) observed that for high mix PCB shops at Hewlett-Packard, it was not unusual to find that over 50% of the production time was spent in setup operations. The setup times typically ranged from 1 to 5 hours, with an estimate of line downtime cost of over $1000 an hour. Moreover, if capacity limitations of the line limited sales of high demand products, this cost could rise to over $10 000 per hour.

Crama and Van de Klundert (1999) identified four basic tool management problems, which were derived from the general tool management model. In the tool switching problem, given a collection of jobs, a processing sequence for the jobs and a loading strategy for the tool magazine has to be found so as to minimize the total number of switches, see for example, Bard (1988), Tang and Denardo (1988a), Crama *et al.* (1994) and Hertz *et al.* (1998). In the tool loading problem, given a collection of jobs and a processing sequence for these jobs, one needs to find a loading strategy for the tool magazine so as to minimize the total number of switches, see for example, Crama *et al.* (1994). The problem of finding the weighted minimum number of switches, i.e., when there is a weight given to each tool, can be solved efficiently, see Privault and Finke (1995). The batch selection problem is concerned with a collection of jobs for which one needs to find the largest subset (batch) of jobs that can be processed without tool switches, see for example, Goldschmidt *et al.* (1994) and Crama (1997). The *job grouping problem* is concerned with sequencing the jobs that are scheduled to be processed and loading tools in the magazine in order to minimize the total number of switching instants, see for example, Tang and Denardo (1988b) and Crama and Oerlemans (1994).

An additional related problem is the physical placement of tools in the magazine, known as the DSA, the *Dynamic Storage Allocation* problem, see for example, Chen *et al.* (2002). The question addressed in this case is how to allocate blocks so that they do not intersect with each other while the used storage size is as small as possible.

A common assumption in the literature on the tool loading and switching problems is that each tool occupies one slot in the tool magazine. Yet, it is common for a tool to occupy several slots. In metal-based industries, this property is applicable to big tools (Stecke, 1983) or to tools that are kept as a kit (Matzliach and Tzur, 2000). In PCB assembly, this property is applicable to components of different sizes, which occupy more than one feeder slot (Jain *et al.*, 1996; Gronalt *et al.*, 1997; Günther *et al.*, 1998).

Previous work on the tool loading or switching problems with tools that may occupy more than one slot in the magazine is relatively limited. The problem is mentioned in Stecke (1983), Shanker and Tzen (1985) and Jain *et al.* (1996) but none of them includes a comprehensive treatment of the problem. Günther *et al.* (1998) discussed the machine loading problem (sequencing the jobs). They used component commonality between any pair of jobs as an estimate for the set-up effort incurred when switching between the job pair, which allowed them to model the problem in a standard approach derived for the traveling salesman problem. Gronalt *et al.* (1997) discussed the complementary problem, i.e., given a job sequence, they addressed the tool loading problem (which they call component set-up) and the slot loading problem (which they call feeder assignment). Their heuristic applies a recursive approach between the component set-up and the feeder assignment problems. More recently, Matzliach and Tzur (2000) addressed the tool loading problem (i.e., when the job sequence is given) with tools that may occupy more than one slot, but with no consideration to their physical location. They proved that the problem is NP-complete and suggested two heuristic procedures.

The problem addressed in this paper is the tool switching problem, in which each tool may occupy more than one slot in the magazine. Machines process parts automatically by using a limited capacity tool magazine. Tools that are not in the magazine are kept in the tool storage area. If a tool that is needed for a certain processing operation is not in the magazine, a tool switch must occur before the job can be processed, a time/cost consuming operation. Thus, one has to decide on three types of decisions, namely, how to select the jobs' sequence (machine loading), which tools to switch before each processing operation (tool loading) and where to locate each tool in the magazine (slot loading). The objective is to minimize the number of tool switches, where planning and scheduling are done off-line, prior to production. Since each tool can consume several (unrestricted) magazine slots, a tool placement in the magazine takes into account the current physical location of the other tools in the magazine. We refer to the latter consideration as *slot assignment*. We present an integer programming formulation for the problem, and suggest a heuristic procedure for its solution. The concept and focus of our heuristic is completely new, however, some procedures within it are based on previously proposed approaches. We then conduct a numerical study in which we solve a collection of tool switching problems, both via our suggested heuristic, as well as via previously suggested approaches from the literature, adjusted by us to handle the case of tools that may occupy more than one slot.

The paper is organized as follows: in Section 2 we specify the problem's assumptions and formally state the problem. In Section 3 we describe our solution procedure and in Section 4 we report on a numerical study. We conclude in Section 5.

2. Problem assumptions and formulations

In this section we state the assumptions and specify the parameters that define the problem. The complete integer programming formulation is given in Appendix A. The assumptions are:

1. There is a set of jobs to be processed. Each job requires a specific set of tools.
2. No job requires a set of tools that occupies more slots than available in the magazine.
3. Before a certain job can be processed, its required set of tools has to be placed in the magazine. If not all the tools are in the magazine, then a tool switch must occur.
4. Planning is done off-line, for the fixed set of jobs, ignoring the effect of the rolling horizon caused by the initial and final tool and slot assignments.
5. The cost/time associated with the placement or removal (rearrangement of slots) of tools is independent of the next job scheduled for processing.
6. The cost/time associated with the placement or removal (rearrangement of slots) of tools is independent of their size.
7. The time needed for start-up at the beginning of processing and for shutdown at the end of processing is fixed. Therefore, tool switches that occur before or after the processing of the set of jobs are ignored.
8. No tool maintenance operation is required during the process, in particular not one that requires a tool switch.
9. The secondary storage site for tools that are not currently in the tool magazine has an unlimited capacity.
10. Each tool occupies a given number of slots, independent of the presence of other tools. Thus, the possibility of tools that partly overlap is ignored.

2.1. *Parameters*

N	$=$	number of jobs that need to be processed; jobs are designated by the index j;
M	$=$	number of tools; tools are designated by the index i;
C	$=$	capacity (number of slots) of the tool magazine; Slots are designated by the index k.

The index n designates instants. Instant n is the point in time at which the nth job has completed processing, but before any tool switches occur. **B** is a vector of length M, whose ith entry b_i represents the number of magazine slots required by tool i **A** is an $M \times N$ matrix whose $(i,\ j)$th entry $a_{ij} = 1$ (0) if tool i is (not) required by job j.

The problem is to determine the sequence of the jobs that need to be processed, the tools that occupy the tool magazine at every instant, and the location of the slots that each tool in the magazine occupies. The objective of the problem is to minimize the total number of tool switches.

As mentioned in the Introduction, we mainly refer to environments found in PCB assembly and metal-based industries. The magazine used in PCB assembly is of the straight type, whereas the magazine used in the metal-based industries is of the round type (slot 1 is adjacent to slot C). In Appendix A we present integer programming formulations for the problem, referring to both cases.

The integer programming formulations, while clarifying the problem and formalizing it, are not useful for obtaining an optimal solution. We used the AMPL Plus software with the CPLEX solver, in an attempt to solve several instances to optimality, using these formulations. However, this attempt was not successful. The largest instance size that the AMPL managed to solve was a non-practical problem with $N = 5$ and $M = 5$. Hence, in the rest of the paper we focus on heuristics.

3. Solution procedure

In this section we describe our suggested heuristic solution procedure for the problem. An overview of the procedure is given in Section 3.1. Then, in Sections 3.2–3.3 we describe how we modified existing procedures from the literature, in order that they can be used for our problem, either as part of our suggested procedure or for comparison purposes. In Section 3.4 we develop a new procedure for arranging the physical placement of tools in the magazine. Finally, in Section 3.5 we describe the overall algorithm, based on the previously described procedures.

3.1. *Overview*

Crama *et al.* (1994) showed that the tool switching problem is NP-hard for any fixed $C \geq 2$. Hence, it is unlikely that a polynomial-time algorithm for it will be found. Many heuristic procedures have been developed for the problem, and most of them fall into two main categories: (i) *construction strategies*, which exploit the special structure of the tool switching problem in order to construct a single job sequence; and (ii) *improvement strategies*, which iteratively improve a starting job sequence. Crama *et al.* (1994) proposed and examined six basic approaches. Two of them, although not decisively, were superior to the others: the *multiple-start-greedy* (labeled as MSG), a constructive strategy, and the *global 2-opt* (labeled as G2OPT) an improvement strategy. The tool switching problem can be modeled as finding a minimum *traveling salesman tour*, where the cost (number of switches) of processing job j subsequent to job i is equivalent to traversing arc $(i,\ j)$. Although, none of the *Traveling Salesman Problem* (TSP)-based procedures that were proposed by Crama *et al.* (1994) exhibited a good performance (mostly due to sensitivity to the density of the tool/job matrix), Hertz *et al.* (1998) proposed several TSP-based heuristic procedures. Their main objective was to overcome the local, narrow view that accounted for interactions of two jobs at a time. They intended to accomplish this by defining a more holistic, global view of "distances" between pairs of jobs. Among the proposed heuristics, a constructive procedure called GENIUS performed best, albeit used a simple, natural estimation of distances. All of the above performance rankings are of solution quality, disregarding the computation time. The above three heuristics, namely MSG, G2OPT and GENIUS, will be referred to from now on as the sequencing heuristics.

Each of the heuristics discussed above, both constructive as well as improving, incorporates the KTNS (Keep Tools Needed Soonest) procedure when computing the cost of a (partial) job sequence. The KTNS procedure was shown by Tang and Denardo (1988a) to be optimal for the tool loading problem in which all tools occupy one slot, see Section 3.2 for more details. When we consider the case of different tool sizes, not to mention slot assignment, the optimality of this procedure is no longer valid. Matzliach and Tzur (2000) showed that the *tool loading problem* is NP-complete when considering different tool sizes, even without considering slot assignment. Therefore, an alternative heuristic procedure is needed to estimate the number of tool switches that are required by a given job sequence. Such a procedure, KSTNS (Keep Smaller Tools Needed Soonest), is discussed in the following section.

The general form of the tool switching problem considered in this work, i.e., with slot assignment, has not yet been dealt with in the literature as one unified problem. The closest work to ours are the two papers by Gronalt *et al.* (1997) and Günther *et al.* (1998), discussed in the Introduction, who together treat the same problem as ours, albeit in two separate steps, where the result of the first step is the input of the second step. In Section 3.5 we present our suggested heuristic procedure called *Aladdin*. For the sake of comparison, we modify the above three sequencing heuristic procedures in order to account for different tool sizes and slot assignments. The modification includes two considerations: (i) using the KSTNS procedure instead of the KTNS procedure, on account of different tool sizes; and (ii) considering physical placement by using a heuristic procedure, called the Block Submersion Procedure (BSP) (discussed in Section 3.4). The three sequencing heuristics mentioned above, are reviewed in Section 3.3.

Given the above mentioned procedures, a heuristic for the tool switching problem with slot assignment, which uses previous approaches, is defined as follows; create a complete job sequence and a tool placement using one of the three sequencing heuristics while using the KSTNS whenever a cost estimate is needed. Once the tool presence in the magazine at each processing stage is given, incorporate the BSP procedure to physically place tools into the magazine slots. We use this framework to modify the three sequencing heuristics, in order to compare them to Aladdin.

Thus, our main contribution consists of two major aspects. One is the adjustment and comparison of previously proposed heuristics, while considering the general form of the tool switching problem. The second contribution is the presentation and analysis of a new heuristic that is based upon novel concepts.

3.2. *KSTNS*

As mentioned in the previous section, the KTNS was shown to be optimal for the tool loading problem in which all tools occupy one slot. According to this policy, a tool is removed from the tool magazine only when it is full and another tool (currently not in the tool magazine) is required. Furthermore, the tools that remain in the tool magazine, are those for which the requirement is soonest. The KTNS policy has been widely used in literature.

We define and use a slight variation of the KTNS, the KSTNS. According to the KSTNS we keep the tools that we need soonest, and among those we prefer the smaller tools. The reason for this variation is to create more space in the magazine once we have to perform a tool switch, instead of choosing arbitrarily among the tools that are needed at the same time. Thus, we may gain more flexible tool placement options, when the problem of the physical placement of tools is addressed. When employing the KSTNS, the physical placement of tools in the magazine is done arbitrarily. The procedure always delivers a feasible tool placement, as will be demonstrated in Section 3.4. Note that as opposed to the KTNS rule, the KSTNS rule is not necessarily optimal for the tool loading problem. In fact, with tools of different sizes, a "bin packing" issue exists, so it cannot be expected that a simple rule would be optimal. Thus, the KSTNS rule is used here heuristically.

The amount of work required by the KSTNS policy can be computed as follows. In a preprocessing step, tools are sorted at each instant according to the first time at which they will be needed. This preprocessing step takes $O(NM \log M)$ time. The rest of the procedure takes $O(M)$ time, and is executed $O(N)$ times, resulting in an overall complexity of $O(MN)$. Thus, given the preprocessing step, the KSTNS takes $O(MN)$ time. This preprocessing step has to be performed only once for all places where KSTNS is used, and its complexity is not significant relative to the overall complexity of any of the other procedures. That is, given the preprocessing step, whenever the KSTNS is used in any of the heuristics, it would take $O(MN)$ time. If the resulting complexity of the heuristic becomes, say, $O(f(H))$, then its overall complexity (including the preprocessing step) would be $O(NM \log M) + O(f(H)) = O(f(H))$ since $O(f(H))$ always dominates $O(NM \log M)$. Hence, in all future calculations, we refer to the complexity for the KSTNS $O(MN)$.

3.3. *Previous approaches*

As mentioned in Section 3.1, we use the sequencing heuristics MSG, G2OPT and GENIUS. In this section we review these procedures. Since in all of them we use the KSTNS instead of the KTNS, and otherwise follow the original procedures as they appear in the literature, we denote the modified procedures as MSG^m, G2OPT^m and GENIUS^m, respectively.

Crama *et al.* (1994) suggested the MSG greedy heuristic. The jobs to be scheduled are divided into two groups. The first group consists of the non-sequenced candidates, designated as Q. The second group is an ordered set σ, which includes the partial job sequence. At each phase of this constructive strategy, the next job to be sequenced is chosen from Q, according to minimal cost (switches) as computed by the KSTNS (originally the KTNS was used). Namely, for each job j in Q, a cost $c(j)$ is computed by employing the KSTNS for the partial sequence (σ, j). The job with the minimal $c(j)$ is removed from Q, and the sequence σ is updated accordingly. The algorithm runs N times, once for each initial sequence $\sigma = (j)$, $j = 1, 2, \ldots, N$ and retains the best complete sequence found. MSG$^{\text{m}}$ runs in $O(MN^4)$ since each initial sequence requires $O(N^2)$ applications of the KSTNS procedure, and there are N initial sequences.

The procedure G2OPT used by Crama *et al.* (1994), is based on an idea that has been widely used for other combinatorial optimization problems (originally used for the TSP). 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 the pth position and j in the kth position. The procedure continues as long as an improvement from the exchange can be achieved. G2OPT$^{\text{m}}$ runs in $O(MN^3)$ operations, since there are

$$O(N^2)$$

possible pairs of jobs to examine, resulting in $O(N^2)$ times of finding two jobs to exchange, in each of which the KSTNS procedure is called.

GENIUS is a two-phase, constructive and improvement TSP based heuristic, proposed by Gendreau *et al.* (1992) (in order to emphasize this TSP oriented approach, the term "tour" will be used to describe the job sequence). It consists of two parts, a GENeral Insertion procedure called GENI, and an improving procedure called US. When using the GENI procedure with KSTNS instead of KTNS, we refer to it as GENI$^{\text{m}}$. The main feature of GENI is that insertion of job v in the tour does not necessarily take place between two jobs, which are consecutive (adjacent). However, after insertion, these two jobs become adjacent to v. For any job v, define its p-neighborhood $Np(v)$ as the set of the p jobs on the tour closest to v, with respect to some distance measure. Starting from three arbitrary jobs, GENI inserts at each step a job v not yet on the current tour, between two jobs already on the tour. The latter two jobs are among the p closest neighbors of v. Gendreau *et al.* (1992) distinguished between two possible insertion types, for each orientation of the tour. Suppose that we wish to insert job v between two jobs v_i and v_j of the tour. Let v_k be a job on the path from v_i to v_j, and v_l be a job on the path from v_j to v_i. The insertion type is affected by the orientation of the quadruple v_i, v_j, v_k and v_l. In type I insertion, $v_k \neq v_i$ and $v_k \neq v_j$. Inserting v in the tour results in re-sequencing of the jobs, in a manner

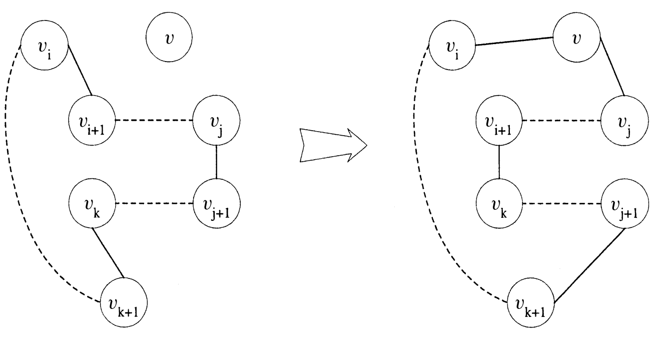

Fig. 1. Type-1 insertion of job v between jobs v_i and v_j in GENI.

that is illustrated in Fig. 1 (job v_l does not participate in type-I insertion).

Dotted arcs between any two jobs v_i and v_j stand for the subset V_{ij} of jobs of the tour, that their subsequence between jobs v_i and v_j is not affected by the insertion. Specifically, type-I insertion is obtained by the following. Given the following sequence (as in Fig. 1):

$$\langle V_{k+1,i}, v_i, v_{i+1}, V_{i+1,j}, v_j, v_{j+1}, V_{j+1,k}, v_k, v_{k+1} \rangle,$$

inserting v in the tour between jobs v_i and v_j results in the following sequence:

$$\langle V_{k+1,i}, v_i, v, v_j, V_{j,i+1}, v_{i+1}, v_k, V_{k,j+1}, v_{j+1}, v_{k+1} \rangle.$$

In type-II insertion, $v_k \neq v_j$ and $v_k \neq v_{j+1}$, $v_l \neq v_i$ and $v_l \neq v_{i+1}$. Inserting v in the tour results in resequencing of the jobs, so that given the following sequence:

$$\langle V_{k,i}, v_i, v_{i+1}, V_{i+1,l-1}, v_{l-1}, v_l, V_{l,j}, v_j, \\ v_{j+1}, V_{j+1,k-1}, v_{k-1}, v_k \rangle,$$

after inserting v in the tour between jobs v_i and v_j, we get the sequence:

$$\langle V_{k,i}, v_i, v, v_j, V_{j,l}, v_l, v_{j+1}, V_{j+1,k-1}, v_{k-1}, v_{l-1}, \\ V_{l-1,i+1}, v_{i+1}, v_k \rangle.$$

Gendreau *et al.* (1992) devised the aforementioned insertion types, corresponding to the Euclidean nature of the orientation among vertices in a plane. No matter how vague the intuitive applicability of such insertion procedures to our problem is, we follow its guidelines.

GENI is more than a standard insertion procedure as each insertion is executed simultaneously with a local optimization of the tour. At each step the GENI procedure selects a job to be inserted in the current tour and its best position in the tour. These are done by computing for each tentative insertion the number of tool switches using the KSTNS (originally the KTNS was used) policy, and performing the insertion yielding the smallest number of tool switches.

The neighborhood of GENI has to be computed according to some distance criterion. Hertz *et al.* (1998) found out that best results were obtained by using $p = 6$ neighbors and the distance $d_{(i,j)} = |T_i \cup T_j| - |T_i \cap T_j|$ where T_i

is the set of tools required by job *i*. This distance measure is natural in the sense that it takes a larger value when jobs *i* and *j* have fewer tools in common.

In the post-optimization phase, US, each job is in turn removed from the tour using the reverse GENI operation, and the job is then reinserted in the tour using GENI. These processes are called *Unstringing* and *Stringing* (US). The procedure ends when removing and reinserting any job can obtain no further improvement. During the *unstringing* phase, similar to the insertion processes of GENI, when a job is removed, two ways of reconnecting the tour are considered. A formal description and complexity analysis of GENIUS^m are provided in Appendix B.

3.4. *The BSP*

Once the tool presence in the magazine at each instant is known, we need to employ a procedure that will take into account the *physical placement* of the tools. This issue has not yet been addressed by the job sequencing phase, nor by the tool loading phase. We suggest here a procedure that we denote as the BSP.

We define a *block*, associated with a given tool, as a rectangle whose height is equal to the tool size, and whose length is equal to the number of consecutive jobs for which the tool occupies the same magazine slot(s). We refer to a *block length* as the length of the rectangle it represents. The basic idea of the BSP is to horizontally justify the spaces that each tool occupies along the job sequence in the magazine, in order to create blocks. Figure 2 illustrates tool blocks in the magazine.

The BSP is suggested both as a method for tool placement, as well as an estimate of the minimum number of switches. We have to alter the definition of tool switches, since now it is applicable to tool movement/replacement inside the magazine only, namely, permuting the tools for each job along the *C* magazine slots, since the tool loading outline is already determined. Thus, given an initial assignment of some blocks in the magazine, if a block (associated with a given tool) cannot be placed in the lowest indexed available slots of the magazine in its full length (for all the jobs in which it is present), a *block breaking* must occur. A block breaking means that during the time in which the tool is present in the magazine, the tool changes its position in the magazine and is therefore represented by two (or more) smaller rectangles, instead of a single large rectangle. As a result, a block breaking is equivalent to a tool switch. Thus, the total number of switches is the sum of the switches as computed by the KSTNS and the number of block breakings that we have to perform in order to physically fit all tool blocks into the magazine.

The fewer breakings a block suffers, the less we have to replace a tool's position along the production process. However, once a block breaking must occur, the length of the resulting block (rectangle) is not important. Namely, it is equivalent to break a block of length eight into two blocks of length four each, or into two blocks of lengths one and seven. Hence, the breaking procedure should not seek a breaking point by considering the resulting blocks' lengths. We use a rule of thumb in breaking blocks (when necessary), which is: break the block at the point where it intersects another block.

Another issue that has to be addressed is prioritizing the blocks, that is, according to what priority rule should we "submerse" the blocks to the bottom of the magazine. We use an intuitive rule, LBF (Longer Blocks First). The LBF

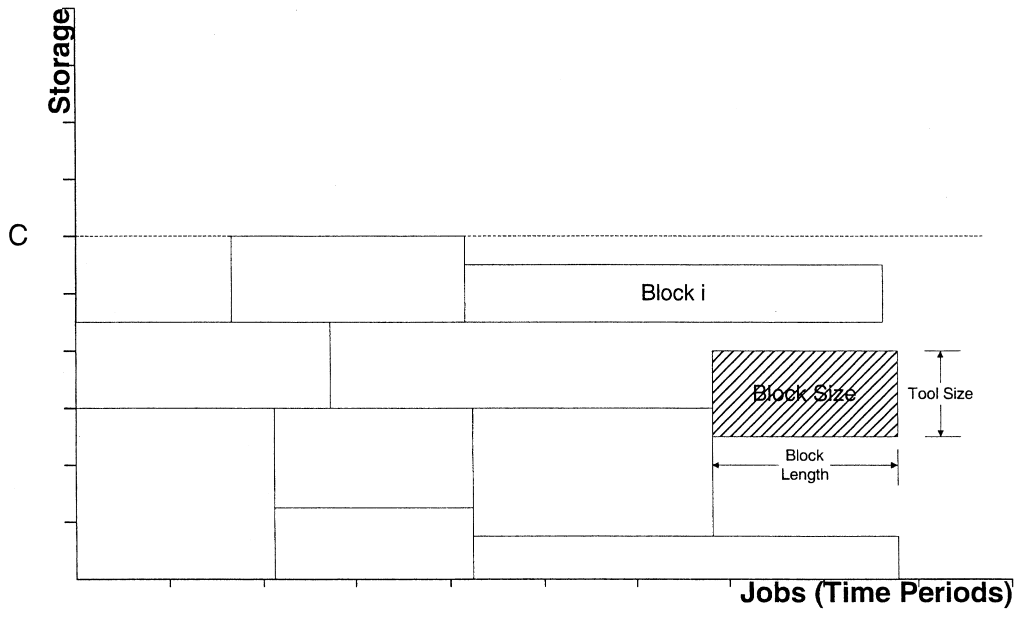

Fig. 2. An illustration of the blocks.

rule is natural to consider, since occupying magazine slots with smaller blocks will reduce the degrees of freedom for the placement of larger blocks (the same logic is employed in bin packing heuristics).

Now, a general outline of the BSP can be drawn. Given the tools present in the magazine at each instant, define the tool blocks. Submerse the blocks according to the LBF priority rule, that is, placing it as deep as possible towards the bottom of the magazine. Intersection with other blocks is of course not allowed. If a block cannot fit into the magazine with a full length, break it at the points of intersection, and keep moving the block parts downward. Stop when all blocks have been placed.

It is easy to see that the BSP always delivers a feasible solution since no free spaces beneath the blocks are created. A precise specification of the BSP and its complexity are provided in Appendix C.

In Appendix A we presented two integer programming formulations for the problem. One corresponds to the straight type magazine and the other to the round type magazine. The description of the BSP depicts the magazine as an array of "vertical columns", each of size C. This description is mostly illustrative. The algorithm refers to each slot individually, hence, it is applicable to both magazine types.

3.5. *Aladdin*

In this section we describe Aladdin, our suggested solution procedure to the tool switching problem. The Aladdin procedure takes precedence over previously proposed schemes, by answering simultaneously the three types of decisions stated in the Introduction, namely, machine loading, tool loading and slot loading. Next, we present informally some of the core notions of the Aladdin procedure.

A most important design guideline is *not to answer the above questions sequentially*. The weakness of previous procedures lies in their sequential nature, namely, first determine the job sequence, then determine the tool loading scheme, and finally determine in which slot each tool should be placed. Another important principle of Aladdin is its *strong tool placement orientation*. As will be demonstrated in Section 4, the number of tool switches significantly increases when dealing with actual tool placements. Thus, a sequence-oriented procedure is myopic in the sense that a relatively good result without placement will turn out to be poor when placement is considered.

Aladdin starts by using the GENIUS algorithm to produce a *tool sequence.* Note that since we produce a tool sequence, the use of KSTNS is not needed, therefore we apply the original GENIUS algorithm, with a distance measure between tools as defined below. The original neighborhood and distance measures when producing a job sequence concern the relation between two jobs. However, in order to maintain the principle of tool placement orientation, we consider the relation between two tools. Similarly to the GENI procedure, for any tool v, we define its p-neighborhood $Np(v)$ as the set of the p tools on the tour closest to v, with respect to the distance measure $d_{(v,j)}$. Thus, we redefine the distance between two tools to be $d_{(i,j)} = N - |J_i \cap J_j|$, where J_i is the set of jobs that require tool i. Note that we not only use J_i instead of T_i, but also we negate the original expression by subtracting it from the total number of jobs. This is because GENI seeks the least cost routing, which will be achieved in our case, if we place in the magazine those tools that share many common jobs. In that case, during a long job sequence they may not have to be switched.

Once we obtain the tool sequence, we start to load and place the tools into the magazine in accordance with that order, as long as we do not exceed the magazine's capacity. We can start each time with another tool in the sequence, thus employing the Multiple Start feature of the MSG algorithm. Furthermore, doing so is useful for another reason. By using the GENIUS algorithm, we create a globally short path. It does not assure us that the distance between any two sequential tools is the minimum. However, when loading tools onto the magazine, the question that we want to answer is: how many jobs can we produce with that set of tools? This is the quality of a *set of tools* and not of a *pair of tools*. Thus, if we scan more starting sets, we may obtain a better final solution. For those tools that are in the magazine, we look for those jobs whose tool requirements are fulfilled by that placement. The sequence for those jobs can be determined arbitrarily. When no more jobs are found for the current tool set, a tool switch must occur.

A tool switch is a combination of insertion and extraction of tools. We mark, among the jobs that have not yet been sequenced, which tools are available. Next, we start to insert tools according to the sequence until a job to be produced is found. At this point, the capacity constraint is violated. Therefore, we remove tools that are no longer needed, followed by tools that are not needed by the job that was found. We break ties by tool size, keeping smaller tools. The tool that has been extracted is moved to the end of the tool sequence. The extraction step is repeated until there is no violation of the capacity constraint. No tool is removed from the magazine, unless the next tool requires its space. The KTNS and KSTNS employ a similar principle of performing just the necessary movements. However, due to the different tool sizes, there is no guarantee that the tools that have been switched can also switch places. Thus, we have to perform an Interlaced Placement Phase (IPP) for the new tool, this will now be described.

The Aladdin procedure shifts tool positions during the phase of tool switching. Thus, two "pathological" phenomena may occur. One, as described in Fig. 3, is "vertical block shifting". When a tool is removed from the bottom of the magazine, the tools above it may be bottom justified, creating many unnecessary tool movements. This phenomenon is especially problematic for the first tools to be removed.

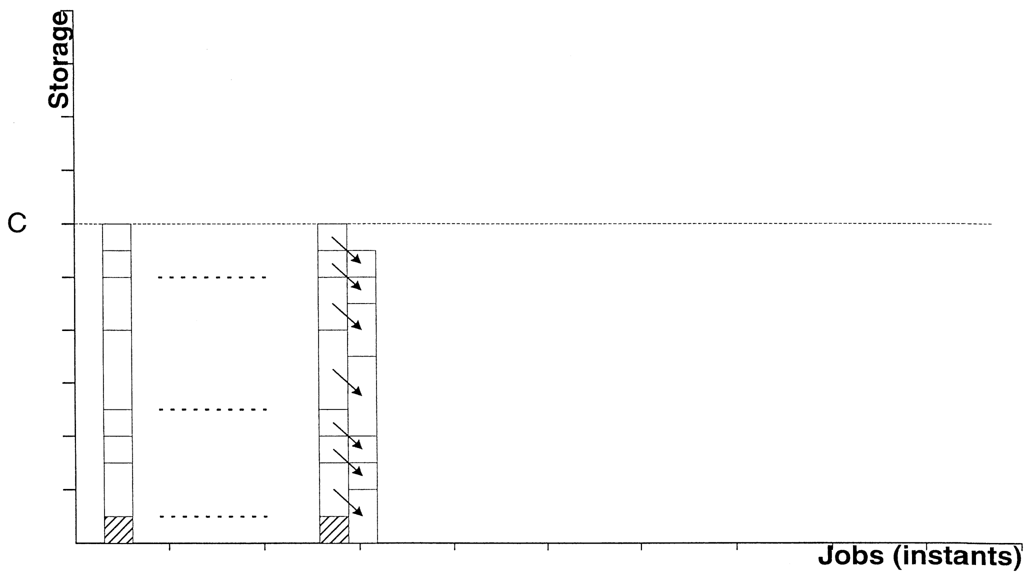

Fig. 3. Vertical block shifting.

Thus, upward shifting of a tool to be removed, and eventually moving it to the top of the magazine, avoids the unnecessary tool movements. This may not be applicable for tools that are removed in later stages of the process. The upward movement may cause an even greater number of switches because of misalignment with respect to previous columns.

The second "pathological" phenomenon, as described in Fig. 4 is "tool zigzagging". This phenomenon applies to the case of a tool, which is in the magazine for a relatively long period. However, it changes its position because of tools that have been moved from beneath it. This can be remedied by the following procedure. If the tool that has been removed is larger than or equal to the one that has been inserted, then it takes its place, bottom justified. If it is not, then more than one tool has to be extracted. The vacant spaces are not necessarily adjacent. We want to create enough space for the new tool by moving the fewest number

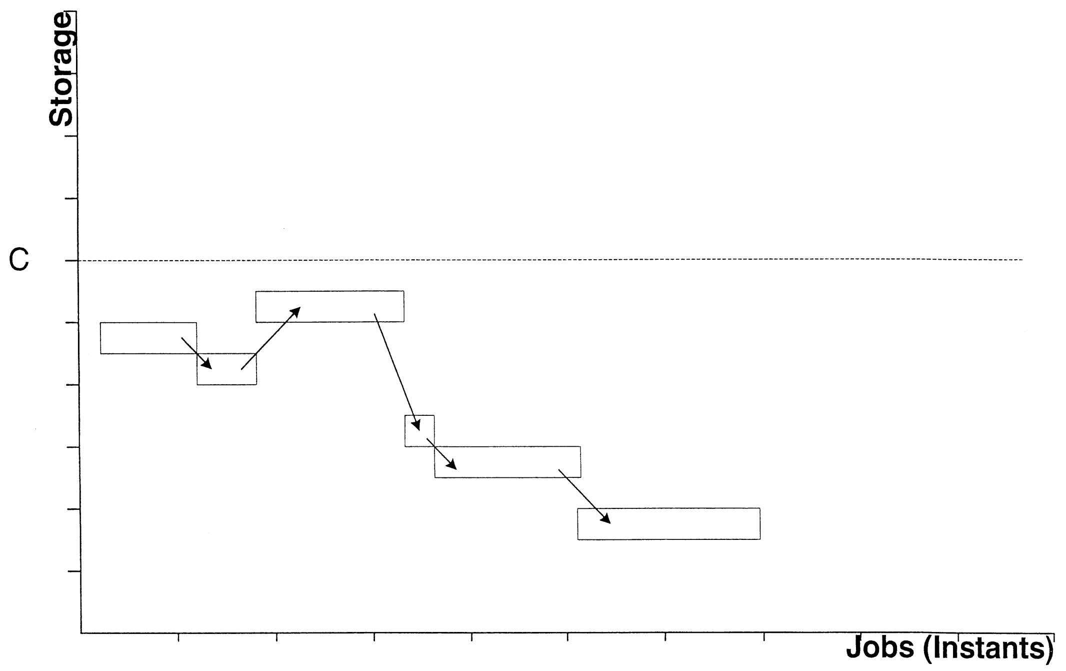

Fig. 4. Tool zigzagging.

of tools as possible. We start to move the tools bottom-wise, starting from the lowest tools. When enough space becomes vacant, we stop.

Once a new tool has been inserted, we look for new jobs to sequence. The process is terminated when there are no jobs to sequence. Finally, the Post-Optimization Phase (POP) moves to the bottom of the stack tools that appear on the magazine for the entire N jobs. A general flowchart of the entire Aladdin algorithm is presented in Fig. 5. A formal description of Aladdin and its complexity analysis are provided in Appendix D.

4. Computational study

In this section we present our computational study, which compares the performance of our suggested Aladdin procedure, with existing procedures from the literature, modified by us to tackle different tool sizes.

4.1. *Data sets*

Our computational study is similar to that of Hertz *et al.* (1998), which followed Crama *et al.* (1994), which in turn

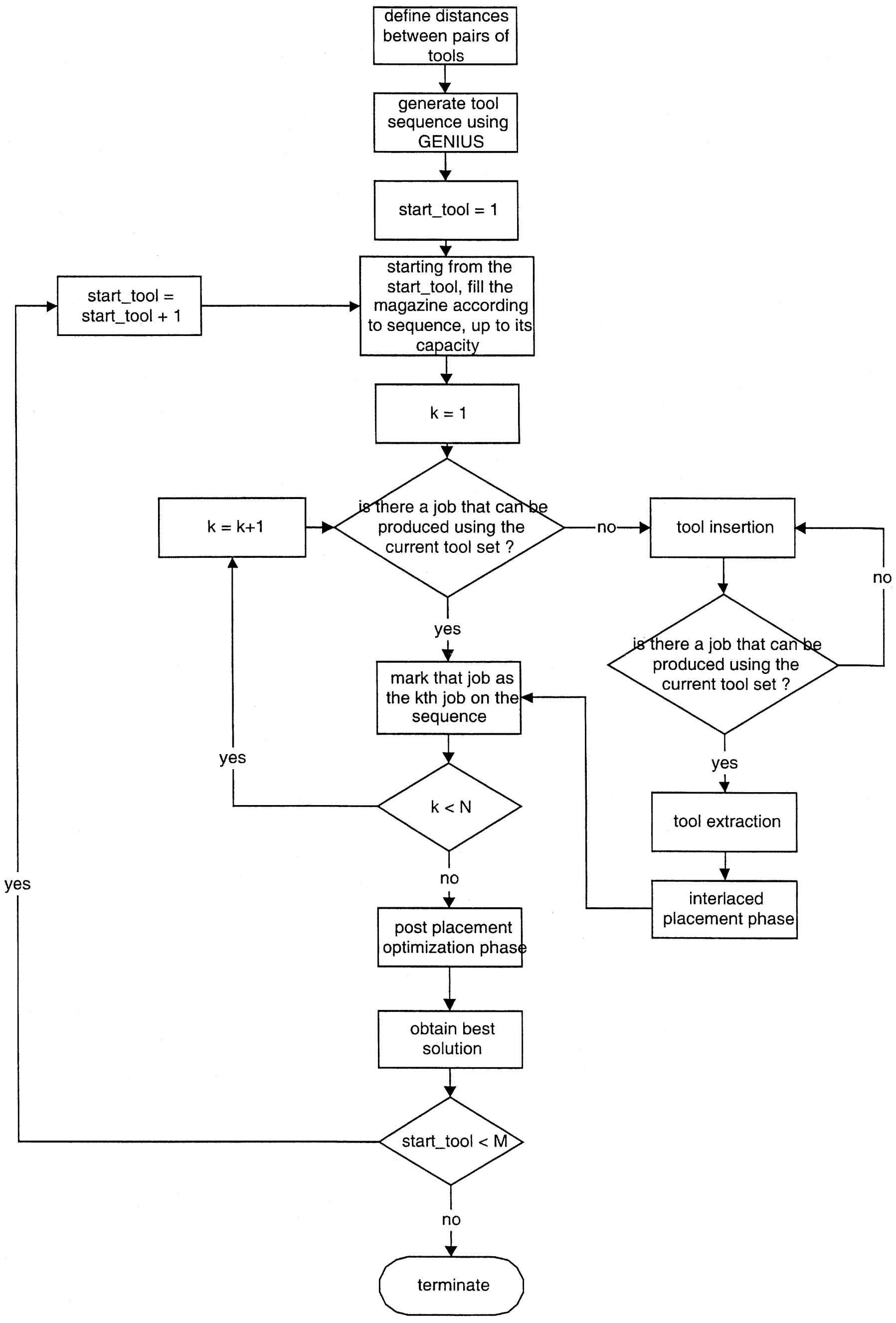
define distances between pairs of tools
generate tool sequence using GENIUS
start_tool = 1
starting from the start_tool, fill the magazine according to sequence, up to its capacity
start_tool = start_tool + 1
k = 1
k = k+1
is there a job that can be produced using the current tool set ?
no
tool insertion
no
yes
is there a job that can be produced using the current tool set ?
mark that job as the kth job on the sequence
yes
yes
k < N
tool extraction
yes
no
interlaced placement phase
post placement optimization phase
obtain best solution
start_tool < M
no
terminate

Fig. 5. Flowchart of the Aladdin procedure.

Table 1. Problem types

N	*M*	*Min*	*Max*	*C*
10	10	2	4	12, 15, 20, 25
15	20	2	6	18, 25, 30, 35
30	40	5	15	45, 50, 55, 60
40	60	7	20	60, 65, 70, 75

followed Tang and Denardo (1988a, 1988b). Hertz *et al.* (1998) refurbished the computational assessment of Crama, *et al.* (1994) producing 10 random repetition instances of each of 16 problem types. We use the same 16 problem types, each type is characterized by a set of parameters $\{N, M, Min, Max$ and $C\}$, where:

N	=	number of jobs;
M	=	number of tools;
Min	=	lower bound on the number of tools required for any job;
Max	=	upper bound on the number of tools required for any job; and
C	=	the capacity of the tool magazine.

Tool sizes vary randomly between one to three magazine slots. We differ from Hertz *et al.* (1998) with respect to generating the magazine capacity. Since unlike previous studies, we allow tool sizes to vary between one to three slots, we have to provide a magazine capacity of at least $3Max$, in order to ensure a feasible tool assignment for each job. Hence, our figures regarding magazine capacity are larger than those of Hertz *et al.* (1998). The various problem types that were generated are described in Table 1. For each type, 10 instances were randomly generated, resulting in a total of 160 instances.

4.2. *Results and analysis*

At phase 1, we implemented our modifications to the existing procedures from the literature. This includes first implementing the sequencing heuristics: G2OPT^{m}, MSG^{m} and GENIUS^{m}, in order to create a job sequence, then using the KSTNS procedure to determine the tool loading scheme. Finally, once the tool presence in the magazine at each instant has been determined, we employed the BSP procedure, which takes into consideration the physical placement of the tools. The total number of switches in the solution is the number of switches as computed by the KSTNS, plus the number of block splits as computed by the BSP. The resulting number of solution runs is:

$$16 \text{ problem types} \times 10 \text{ random instances} \times 3 \text{ sequencing heuristics} = 480 \text{ solution runs.}$$

We obtained results for both with and without tool placement.

At phase 2, we implemented the Aladdin procedure, with the same input data. The results of phase 1 were used as a reference for estimating the quality of Aladdin.

All solution procedures were implemented in C and run on a Pentium II 350 MHz processor PC. For each algorithm/problem type combination, we report two average statistics over the 10 randomly generated instances. The first is the number of tool switches and the second is the computation time in seconds.

In Table 2 we summarize the results of the algorithms that were tested on phase 1, without placement. For problem types 1–8 (small problems), we observe that the solution quality (i.e., number of switches) is almost identical for all algorithms. We believe that for those small-scale problems, the results that were obtained are close to optimum, reflecting the lack of degrees of freedom in the tool placement options. For problem types 9–16 (medium and large problems), the MSG^{m} algorithm performed slightly better than G2OPT^{m} and GENIUS^{m}. However, the running time of the MSG^{m} algorithm was significantly longer than those of the other algorithms: approximately 15 times that of the G2OPT^{m} algorithm, and 144 times that of the GENIUS^{m} algorithm, for the largest problem types (types 13–16).

The computational results of both Aladdin and the algorithms that were tested on phase 1 with placement, are summarized in Table 3. Note first that among the procedures considered in phase 1, when placement is included, G2OPT^{m} has a slight advantage (although not significantly) over the others, while MSG^{m} comes second. When comparing the solution quality of all algorithms, we observe that for small and medium sized problem instances, the various algorithms perform similarly. However, for large-scale instances, Aladdin demonstrates clear dominance. In nine out of 16 types, Aladdin performed the best. Those nine types included the four largest problems, on which Aladdin had on average 26% fewer switches than the second best algorithm (G2OPT^{m}). For three other types, Aladdin performed at least as good as the best among the other sequencing heuristics. For three types the G2OPT^{m} came first, and for one type the MSG^{m} won. With respect to running time, Aladdin was the second fastest algorithm (after GENIUS^{m}), with only a 5.3 seconds running time for the largest problem considered.

In Table 4 we present a comparison of the Aladdin results with and without the post-placement optimization phase, in order to evaluate the importance of this step. The results demonstrate that the saving earned by the POP step is significant, but decreases (in terms of percentage) as the problem size increases. For the largest problems (problem types 13–16), the average saving was about 20%.

Finally, we consider an additional data set in which the various tool sizes vary in their proportion/frequency. Albeit real-life experience gives us no reason to assume differently, one might find it interesting to examine several more tool

Table 2. Summary of computational results: without placement

			G2OPT^m		*MSG^m*		*GENIUS^m*	
M, N, Min, Max	*C*	*Problem type*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*
10, 10, 2, 4	12	1	0.017	3	0.083	3	0.022	3
	15	2	0.022	2	0.082	2	0.022	2
	20	3	0.022	0	0.083	0	0.016	0
	25	4	0.022	0	0.082	0	0.022	0
Average				1.25		1.25		1.25
15, 20, 2, 6	18	5	0.138	10	0.774	10	0.061	9
	25	6	0.137	5	0.775	5	0.06	5
	30	7	0.137	3	0.785	3	0.055	3
	35	8	0.143	1	0.791	1	0.049	1
Average				4.75		4.75		4.5
30, 40, 5, 15	45	9	2.84	48	32.197	44	0.418	45
	50	10	2.817	38	32.165	34	0.511	37
	55	11	2.823	22	32.087	21	0.373	24
	60	12	2.835	14	32.022	12	0.456	14
Average				30.5		27.75		30
40, 60, 7, 20	60	13	11.765	95	173.21	90	1.175	96
	65	14	11.902	82	173.99	77	1.209	82
	70	15	11.716	74	172.76	68	1.214	74
	75	16	11.765	56	173.53	52	1.219	57
Average				76.75		71.75		77.25

size proportions. Hence, we introduced a frequency vector that corresponds to the proportions of a given tool size, and reproduced the results of Table 3. For example, frequency vector (1/3, 1/3, 1/3) stands for assigning equal proportion to tool sizes 1, 2 and 3, as in the previous data set. We have examined three more frequency vectors: (0.2, 0.2, 0.6), (0.2, 0.6, 0.2), and (0.6, 0.2, 0.2). The results are given in Table 5, where the "problem type" heading corresponds to

Table 3. Summary of computational results: with placement

			G2OPT^m		*MSG^m*		*GENIUS^m*		*Aladdin*	
M, N, Min, Max	*C*	*Problem type*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*
10, 10, 2, 4	12	1	0.017	9	0.083	9	0.022	9	0.028	6
	15	2	0.022	4	0.082	5	0.022	5	0.038	4
	20	3	0.022	1	0.083	1	0.016	1	0.033	0
	25	4	0.022	0	0.082	0	0.022	0	0.039	0
Average				3.5		3.75		3.75		2.5
15, 20, 2, 6	18	5	0.138	24	0.774	30	0.061	31	0.11	29
	25	6	0.137	18	0.775	19	0.06	15	0.137	16
	30	7	0.137	8	0.785	10	0.055	9	0.159	9
	35	8	0.143	3	0.791	3	0.049	4	0.192	2
Average				13.25		15.5		14.75		14
30, 40, 5, 15	45	9	2.84	234	32.197	240	0.418	245	1.214	209
	50	10	2.817	197	32.165	221	0.511	216	1.286	183
	55	11	2.823	118	32.087	133	0.373	151	1.45	123
	60	12	2.835	75	32.022	66	0.456	82	1.675	78
Average				156		165		173.5		148.2
40, 60, 7, 20	60	13	11.765	563	173.21	550	1.175	572	4.822	414
	65	14	11.902	517	173.99	526	1.209	544	4.993	382
	70	15	11.716	509	172.76	503	1.214	532	5.174	385
	75	16	11.765	418	173.53	438	1.219	456	5.328	305
Average				501.7		504.2		526		371.5

Table 4. Summary of computational results: POP versus no POP

			Aladdin—No POP		*Aladdin*	
M, N, Min, Max	*C*	*Problem type*	*Time (seconds)*	*Switches*	*Time (seconds)*	*Switches*
10, 10, 2, 4	12	1	0.022	12	0.028	6
	15	2	0.027	8	0.038	4
	20	3	0.022	2	0.033	0
	25	4	0.022	0	0.039	0
Average				5.5		2.5
15, 20, 2, 6	18	5	0.077	39	0.11	29
	25	6	0.077	30	0.137	16
	30	7	0.071	25	0.159	9
	35	8	0.072	12	0.192	2
Average				26.5		14
30, 40, 5, 15	45	9	0.785	271	1.214	209
	50	10	0.736	252	1.286	183
	55	11	0.593	210	1.45	123
	60	12	0.533	168	1.675	78
Average				225.25		148.25
40, 60, 7, 20	60	13	3.422	493	4.822	414
	65	14	3.301	474	4.993	382
	70	15	3.120	472	5.174	385
	75	16	2.730	434	5.328	305
Average				468.25		371.5

Table 5. Computational results for different frequency vectors

	Frequency vectors											
	(0.2, 0.2, 0.6)				*(0.2, 0.6, 0.2)*				*(0.6, 0.2, 0.2)*			
Problem type	*G2OPT^m*	*MSG^m*	*GENIUS^m*	*Aladdin*	*G2OPT^m*	*MSG^m*	*GENIUS^m*	*Aladdin*	*G2OPT^m*	*MSG^m*	*GENIUS^m*	*Aladdin*
1	13.3	16.5	13.7	14.4	9	14.1	10.5	10.8	6.5	8.4	7.2	5.7
2	10.1	12.9	9.6	10.2	5.3	6.2	4.8	4.3	1.6	1.8	2.4	1.5
3	3.6	4.1	4.2	2.8	0.7	0.7	1	0.5	0	0	0.2	0.2
4	1.7	1.7	1.2	0.7	0.5	0.5	0	0.4	0	0	0	0.2
Average	7.2	8.8	7.2	7.0	3.9	5.4	4.1	4.0	2.0	2.6	2.5	1.9
5	36.9	44.5	38.2	47.9	31.7	43.9	35.3	40.3	20.3	30.4	27.3	30.4
6	26.1	37.8	28.2	34.5	16.1	27.3	19.1	26.5	8	8.4	9.7	8.3
7	16.2	21.5	18.7	20.8	8.3	8.8	8.3	9.1	0.8	1.3	2.1	1.3
8	11.5	12.6	11.7	13.4	4.4	5	6	5	1	1.1	1.8	1
Average	22.7	29.1	24.2	29.2	15.1	21.3	17.2	20.2	7.5	10.3	10.2	10.3
9	234.9	294.8	251.8	265.3	201	249.6	211.6	229.9	92	156.7	113.6	136.6
10	250.2	293.7	260.8	260.6	190.7	245.6	199.8	214.8	85.9	153.3	113.7	126.3
11	225.5	268.5	239.9	235.9	136.7	185	164.6	174.3	28.4	60.2	46.2	45.4
12	176.3	261.7	187.4	197.4	85.9	154.4	104.3	124.8	11.2	22.9	18.2	18.1
Average	221.7	279.7	235.0	239.8	153.6	208.7	170.1	186.0	54.4	98.3	72.9	81.6
13	585.5	630.9	576.6	523.8	591.8	643.2	559.8	468.7	405.9	541.5	440.9	366.2
14	598.8	637.7	562.8	507.5	568.3	649.8	584	442.8	350.2	530.3	419.4	336.4
15	576.7	659.7	599.1	458.7	486.2	594	515.8	395.6	224.4	381.8	297.8	261.7
16	576.6	616.5	558.4	450.6	425.7	512.5	478	373.1	139.3	257.8	216.9	203
Average	584.4	636.2	574.2	485.2	518.0	599.9	534.4	420.1	280.0	427.9	343.8	291.8
Average	189.2	217.5	191.0	174.8	154.5	188.2	162.9	143.7	75.8	119.3	94.9	86.1

that in Table 3. The results indicate that overall Aladdin still performs the best, with G2OPT^m in second place. G2OPT^m performs best for problems in which the proportion of tools with size one is largest, only slightly better than Aladdin. This is not surprising, since Aladdin was designed to handle larger tool sizes while G2OPT^m considered tools of size one only. It can also be noted that many large tools cause a reduction in the flexibility of tool placement, thus causing more tool switches, in which case Aladdin performs best.

5. Summary and conclusions

We have presented a new algorithm for the tool switching problem in which each tool can occupy more than one slot of the tool magazine. This problem in its full generality has not been previously addressed in the literature. Our suggested heuristic, Aladdin, addresses all three types of decisions, namely, machine loading, tool loading and slot loading, simultaneously. In our numerical study, our heuristic demonstrated a clear dominance over existing heuristic approaches that were modified by us to handle different tool sizes. The dominance with respect to solution quality (number of switches) increased as the problem size increased and approached problems of realistic size.

Acknowledgement

We are grateful to G. Laporte for sending us the code for the GENIUS algorithm.

References

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

Chen, B., Hassin, R. and Tzur, M. (2002) Allocation of bandwidth and storage. *IIE Transactions*, **34**, 501–507.

Crama, Y. (1997) Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, **99**, 136–153.

Crama, Y., Kolen, A.W.J., Oerlemans, A.G. and Spieksma, F.C.R. (1994) Minimizing the number of tool switches on a flexible machine. *The International Journal of Flexible Manufacturing Systems*, **6**, 33–54.

Crama, Y. and Oerlemans, A.G. (1994) A column generation approach to job grouping for flexible manufacturing systems. *European Journal of Operational Research*, **78**, 58–80.

Crama, Y. and Van de Klundert, J. (1999) Worst-case performance of approximation algorithms for tool management problems. *Naval Research Logistics*, **46**, 445–462.

Gendreau, M., Hertz, A. and Laporte, G. (1992) New insertion and optimization procedures for the traveling salesman problem. *Operations Research*, **40**, 1086–1094.

Goldschmidt, O., Nehme, D. and Yu, G. (1994) On the set-union knapsack problem. *Naval Research Logistics*, **41**, 833–842.

Gronalt, M., Grunow, M., Günther, H.O. and Zeller, R. (1997) A heuristic for component switching on SMT placement machines. *International Journal of Production Economics*, **53**, 181–190.

Günther, H.O., Gronalt, M. and Zeller, R. (1998) Job sequencing and component set-up on a surface mount placement machine. *Production Planning and Control*, **9**(2), 201–211.

Hertz, A., Laporte, G., Mittaz, M. and Stecke, K.E. (1998) Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions*, **30**, 689–694.

Jain, S., Johnson, M.E. and Safai, F. (1996), Implementing setup optimization on the shop floor. *Operations Research*, **43**(6), 843–851.

Matzliach, B. and Tzur, M. (2000) Storage management of items in two levels of availability. *European Journal of Operational Research*, **121**, 363–379.

Privault, C. and Finke, G. (1995) Modeling a tool switching problem on a single NC-machine. *Journal of Intelligent Manufacturing*, **6**, 87–94.

Sethi, A.K. and Sethi, S.P. (1990) Flexibility in manufacturing: a survey. *The International Journal of Flexible Manufacturing Systems*, **2**, 289–328.

Shanker, K. and Tzen, Y.J. (1985) A loading and dispatching problem in a random flexible manufacturing system. *International Journal of Production Research*, **23**(3), 579–595.

Stecke, K.E. (1983) Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, **29**(3), 273–288.

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

Tang, C.S. and Denardo, E.V. (1988b) Models arising from a flexible manufacturing machine, part II: minimization of the number of switching instants. *Operations Research*, **36**(5), 778–784.

Tatikonda, M.V. and Stietz, M.K. (1994) An integrated methodology for operation analysis of computer integrated manufacturing systems: application to an FMS for manufacture of medical instruments. *International Journal of Advanced Manufacturing Technology*, **9**, 245–252.

Appendices

Appendix A: Integer programming formulations

We present in this Appendix integer programming formulations for the problem. First, in program A, we refer to the case of a straight magazine, whereas in program B we refer to the round magazine.

Recall that the definitions of the parameters used in the formulation are given in Section 2.

Decision variables

$$x_{jn} = \begin{cases} 1 & \text{if job } j \text{ is the } n\text{th job in the sequence,} \\ 0 & \text{otherwise.} \end{cases}$$

$$w_{ikn} = \begin{cases} 1 & \text{if tool } i \text{ occupies slot } k \text{ of the machine at instant } n, \\ 0 & \text{otherwise.} \end{cases}$$

$$y_{ikn} = \begin{cases} 1 & \text{if } k \text{ is the first (lowest index) slot that tool } i \text{ occupies at instant } n, \\ 0 & \text{otherwise.} \end{cases}$$

That is, $y_{ikn} = 1 \Leftrightarrow w_{ikn}, \ldots, w_{i,k+bi-1,n} = 1$.

$$p_{ikn} = \begin{cases} 1 & \text{if tool } i \text{ occupies slot } k \text{ as the first slot at the } (n+1)\text{th instant and does not occupy this slot at the } n\text{th instant, i.e., if } y_{ikn+1} \text{ equals one and } y_{ikn} \text{ equals zero,} \\ 0 & \text{otherwise.} \end{cases}$$

Program A

$$\text{Min} \sum_{n=1}^{N-1} \sum_{i=1}^{M} \sum_{k=1}^{C} p_{ikn},$$

subject to

$$\sum_{i=1}^{M} w_{ikn} \leq 1 \quad k = 1, \ldots, C, \quad n = 1, \ldots, N, \quad \text{(A1)}$$

$$a_{ij} x_{jn} b_i \leq \sum_{k=1}^{C} w_{ikn} \quad i = 1, \ldots, M, \quad n = 1, \ldots, N, \quad j = 1, \ldots, N, \quad \text{(A2)}$$

$$\sum_{f=k}^{k+b_i-1} w_{ifn} - y_{ikn} \leq b_i - 1 \quad i = 1, \ldots, M, \quad n = 1, \ldots, N, \quad k = 1, \ldots, C - b_i + 1, \quad \text{(A3)}$$

$$b_i y_{ikn} - \sum_{f=k}^{k+b_i-1} w_{ifn} \leq 0 \quad i = 1, \ldots, M, \quad n = 1, \ldots, N, \quad k = 1, \ldots, C - b_i + 1, \quad \text{(A4)}$$

$$\sum_{k=1}^{C} y_{ikn} \leq 1 \quad i = 1, \ldots, M, \quad n = 1, \ldots, N, \quad \text{(A5)}$$

$$p_{ikn} \geq y_{ik,n+1} - y_{ikn} \quad i = 1, \ldots, M, \quad n = 1, \ldots, N - 1, \quad k = 1, \ldots, C - b_i + 1, \quad \text{(A6)}$$

$$\sum_{j=1}^{N} x_{jn} = 1 \quad n = 1, \ldots, N, \quad \text{(A7)}$$

$$\sum_{n=1}^{N} x_{jn} = 1 \quad j = 1, \ldots, N, \quad \text{(A8)}$$

$$x_{jn}, w_{ikn}, y_{ikn}, p_{ikn} = 0, 1 \quad n = 1, \ldots, N, \quad j = 1, \ldots, N, \quad i = 1, \ldots, M, \quad k = 1, \ldots, C - b_i + 1. \quad \text{(A9)}$$

Constraint (A1) limits each slot to be occupied by one tool at the most. Note that summing Equation (A1) over k is equivalent to making sure that no more than C tools are in the magazine at any instant. Constraint (A2) defines that if job j is the nth job on the sequence, and it requires tool i, then b_i slots have to be "reserved" for it at instant n. Constraints (A3) and (A4) make sure that all the above b_i slots of tool i are adjacent. Constraint (A3) forces y_{ikn} to be one if its b_i corresponding w_{ikn}s, are also all ones. Constraint (A4) forces the opposite direction. Constraint (A5) makes sure that a tool is not loaded into the magazine more than once at any one time. Constraint (A6) defines tool movement. Constraint (A7) makes sure that at each instant exactly one job is processed. Constraint (A8) makes sure that all the jobs are processed exactly once. Constraint (A9) is the integrality constraint.

Program B

The same as program A, except for slight changes concerning the adjacency constraints. In the round magazine, extra slot adjacencies are allowed, for example, for $b_i = 2$ a tool may occupy slots $\{C, 1\}$, and for $b_i = 3$ slots $\{C\text{-}1, C, 1\}$ or $\{C, 1, 2\}$. The modification would be to let k run over the extra indices, writing constraints (A3) to (A6) explicitly for the additional possibilities.

Appendix B: Formal description and complexity of GENIUS^m

GENI^m is described by the following procedure.

Step 1. Create an initial tour by selecting arbitrarily a subset of three jobs. Initialize the p-neighborhood of all jobs.

Step 2. Arbitrarily select a job v that is not yet on the tour. Implement, by using the KSTNS, the least cost insertion of v, considering all choices of v_i, v_j, v_k and v_l and the two possible orientations of the tour and the two possible insertion types. Update the p-neighborhood of all jobs to account for the fact that v is now on the tour.

Step 3. If all jobs are now part of the tour, stop. Otherwise go back to Step 2.

In Step 2, $O(p^4)$ choices of v_i, v_j, v_k and v_l must be considered (a maximum of 360 possible combinations for a full six neighbors situation). Each quadruple is examined four times for two insertion types and two tour orientations, where each examination requires $O(MN)$ time on account of the KSTNS. For the best choice, v is inserted in the tour, and then the p-neighborhood of all jobs is updated, which takes $O(N)$ time. Since Step 2 is executed N-3 times, the overall complexity of GENI^m is $O(MN^2p^4 + N^2) = O(MN^2p^4)$.

The US algorithm is described by the following procedure.

Step 1. Consider an initial tour τ of cost z. $\tau^* \leftarrow \tau$; $z^* \leftarrow z$; $t \leftarrow 1$.

Step 2. Starting from tour τ, apply the unstringing and stringing procedures with job v_t, considering in each case the two possible types of reconnecting the tour, and the two tour orientations. Let τ' be the tour obtained and z' be its cost. $\tau \leftarrow \tau'$; $z \leftarrow z'$.

Step 3. If $z < z^*$, $\tau^* \leftarrow \tau$; $z^* \leftarrow z$; $t \leftarrow 1$; repeat Step 2.
Else ($z \geq z^*$, no improvement was obtained): $t \leftarrow t + 1$; if $t = n + 1$ stop. Record τ^* and z^* as the solution.
Else repeat Step 2.

Note that the cost of two consecutive tours produced by Step 2 may increase. Indeed, a job to be removed from the tour is not necessarily located between two jobs that belong to its p-neighborhood. Thus, reinserting the job in the position it occupied before its removal may not be allowed. However, the best-known tour is always stored. As a result, the complexity of GENIUS and GENIUS^m cannot be bounded as a function of N and M as it can be applied as long as the objective function improves.

Appendix C: Formal description and complexity of the BSP

To formally describe the BSP, we use the following definitions. Let $\boldsymbol{W}$ be an $N \times C$ matrix, which reflects the magazine's slot occupancy for every instant, namely, w_{jk} equals one when slot k at instant j is occupied. Recall that b_i is the number of magazine slots required by tool i. Let J_{in} equal one (or zero) if tool i is (is not) in the magazine at a given instant n. This is given as input for the BSP procedure. Define a *tool block* of $\boldsymbol{W}$ as a maximal subset of horizontal consecutive $J_{in} = 1$ in $\boldsymbol{W}$. Let TB denote the set of tool blocks in $\boldsymbol{W}$. Let h be the number of such tool blocks $TB = \{tb_1, tb_2, \ldots, tb_h\}$. Denote the length of tool block l (the number of periods for which it is on the magazine) as sb_l.

The BSP is described by the following procedure.

Step 1. Set $l = 1$. Splits $= 0$.
Step 2. Sort TB in descending order according to sb_l (according to the LBF priority rule).
Step 3. While not exceeding C and not intersecting other blocks, place sb_l as deep as possible at the bottom of the magazine. Mark magazine slots as occupied. $l = l + 1$. If $l = h + 1$ stop. Return splits.
Step 4. Split tb_l at the points of intersection. Place the block parts as deep as possible at the bottom of the magazine. Mark magazine slots as occupied. $l = l + 1$. Splits = splits + number of intersection points. Go to Step 3.

The amount of work that the BSP takes is computed as follow. Step 2 requires O($h \log h$) time. Step 3 is repeated h times and Step 4 is repeated up to h times. The work of placing blocks in the magazine, and marking magazine slots as occupied takes $O(NC)$ for the entire algorithm. Thus, the total amount of work that the BSP requires is $O(h \log h) + O(NC) = O(h \log h + NC)$.

Appendix D: Formal description and complexity analysis of Aladdin

We present here a formal description of Aladdin. Denote by J and J' the groups of non-sequenced and sequenced jobs respectively ($|J| + |J'| = N$). Similarly, denote by T and T' the group of tools that are unloaded from and loaded into the magazine respectively ($|T| + |T'| = M$). σ_T, $\sigma_{T'}$ and $\sigma_{J'}$ are sequences (permutations) of T, T' and J' respectively. The first tool on σ_T is denoted as tool_in. Denote $\boldsymbol{D}$ as the $M \times M$ triangular distance matrix that corresponds to all M tools. We use the notation "A_p_B" to mark that job A can be produced by the set of tools B. Recall that N, M and C are the number of jobs, tools and magazine capacity, respectively. Aladdin is described by the following procedure.

Step 0. *Initialization*: $J = \{1, \ldots, N\}$. $T = \{1, \ldots, M\}$. Calculate $\mathbf{D}$. Set start_tool $= 0$.
Step 1. Create a tool sequence σ_T using the GENIUS algorithm ($|\sigma_T| = M$).
Step 2. Set start_tool = start_tool + 1. While not exceeding C, start from start_tool and position the tools of σ_T with the lowest slot index of the magazine. Mark those tools as in T'. Mark magazine slots as occupied. Mark the next tool in T as tool_in.
Step 3. If $|J'| = N$ go to Step 6. If there is no job A, such that A_p_T' then Go to Step 4. Else, move job A to J'. $J = J \backslash A$. Place A at the end of $\sigma_{J'}$. Repeat Step 3.
Step 4.
  4.1. Move tool_in from T to T'. $T = T \backslash$tool_in. Mark the next tool in T as tool_in. Repeat this until there is a job A, such that A_p_T'. Denote the newly inserted tools as tools_in.
  4.2. Remove tools that are not used by any other job in J or A from T' to T. Break ties by tool size removing the larger tools first. Place it at the end of σ_T.
  4.3. If the tool to be removed is in the magazine for every job until now, then change its place in the magazine by moving it so as to be the first tool. Repeat this stage as long as exceeding C.
Step 5. Move through the magazine from the bottom. Once you reach a vacant space try to insert a tool from tools_in into the space. If no such tool exists, adjust all tools above the space in a downwards direction towards the bottom. Repeat this step until all new tools_in are placed in the magazine. Go to Step 3.
Step 6. If a tool is in the magazine for N periods move it to the bottom of the stack. Adjust other tools in a downwards direction towards the bottom. Record best solution. If start_tool = M then terminate. Else, reset σ_T. Go to Step 2.

The Aladdin procedure encapsulates the GENIUS procedure at its tool sequencing phase. Hence, due to the US part of GENIUS, its complexity cannot be bounded. Nevertheless, we provide a calculation of the amount of work that the other steps of Aladdin take, except for the tool sequencing phase.

The initialization, Step 0, involves calculating the distance matrix $\boldsymbol{D}$. For each pair of tools, we go over all N jobs. For each job we check if this pair of tools are required, by using the corresponding values in the input matrix $\boldsymbol{A}$ (see Section 2). Therefore, this step consumes $O(NM^2)$.

In Step 1 we create a tool sequence σ_T by using the GENIUS algorithm. The complexity of GENI is $O(NM^2p^4)$ (note that we sequence tools and not jobs), and as specified in Section 3.3, we cannot decisively determine the complexity for the US part of the algorithm.

The main loop of the procedure, involves Steps 2 to 6. Step 2 involves filling the first stack, which takes $O(C)$ time. At Step 3, for each selection of a tool set that was placed in the magazine, we have to find its corresponding jobs. Each job requires at most $O(C)$ tools. The presence of each of those tools has to be searched in an array of at most C components. Since there are at most N non-sequenced jobs, this checking step takes $O(NC)$ time. Step 3 may be repeated $O(N)$ times for each start_tool, therefore the complexity of Step 3 for a given start_tool is $O(N^2C)$. The complexity of Step 4 entails the use of Min(M,C) which will be denoted as C'. Step 4.1 is the tool insertion and checking step which takes $O(NC)$. It can be repeated up to C' times, resulting in $O(NCC')$ time for Step 4.1. Tool extraction, which takes place on Step 4.2, can encounter up to $O(C')$ tools and $O(N)$ jobs resulting in complexity of $O(NC')$ time. Step 4.3 considers at most $O(C')$ tools, to be searched in $O(N)$ jobs backwards, and repositioning $O(C)$ tools, that is $O(NCC')$. Steps 4.1 and 4.2 are executed N times resulting in $O(N^2CC')$ for the entire Step 4. The IPP phase, which take place at Step 5 involves up to $O(C)$ tool-shifting, and is repeated N times thus takes $O(NC)$ time. The POP procedure (beginning of Step 6) involves up to C shifts of tools for each job stack, namely a cost of $O(NC)$. Repeating Steps 2–6 for M possible starting tool sets (Step 6) sums up then to: Step 0: $O(M^2N)$, Step 1: unbounded, Steps 2 to 6: $O(MN^2CC')$.

Biographies

Michal Tzur is a Senior Lecturer in the Department of Industrial Engineering, Tel Aviv University, Israel, and is currently a visiting Associate Professor at the Department of Industrial Engineering and Management Science, Northwestern University. She joined Tel Aviv University in 1994 after spending 3 years at the Department of Operations and Information Management, Wharton School, University of Pennsylvania. She received her B.A. from Tel Aviv University and her M. Phil and Ph.D. from Columbia University. Her research interests are in the areas of supply chain management, multi-echelon inventory management, production planning and operations scheduling.

Avri Altman received his B.Sc. from the Technion and his M.Sc. from Tel Aviv University, both in Industrial Engineering. He has worked as an algorithm and software engineer for several telecom companies.

Contributed by the Applied Optimization Department