European Journal of Operational Research 121 (2000) 363–379

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Theory and Methodology

Storage management of items in two levels of availability

Barouch Matzliach [a], Michal Tzur [b,*]

[a] *Shimoni 17, Ramat Aviv, Tel Aviv, Israel*

[b] *Department of Industrial Engineering, Tel Aviv University, Tel Aviv 69978, Israel*

Received 1 January 1998; accepted 1 November 1998

Abstract

This paper is concerned with the problem of storage management of non-consumable items in two warehouses having different levels of availability. The items are used in a given process, which incurs a cost whenever the required item is not immediately available.

We analyze the storage management problem in which items have non-equal size and prove that the problem is NP-Complete. We present two heuristic algorithms which are inspired by two integer programming formulations. The two heuristics are based on different approaches and provide insight that enhances our understanding of the problem. Our numerical study demonstrates that these heuristics perform very well, and therefore provide satisfactory solutions to the problem. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Flexible manufacturing systems; Production; Complexity; Heuristics

1. Introduction

This paper is concerned with a problem of *storage management* of non-consumable items in two warehouses having different levels of availability. The non-consumable items are needed for an operational process, and lack of their availability is associated with costs to the process. Such items are used in manual production processes (for example, auxiliary tools in assembly operations), as well as in automated production processes (for example, tools in a flexible NC machine). The items may also represent data that is related to the process and which is required in order to make decisions during its execution.

The process uses a nearby "warehouse" for storing the items, from which the items may be brought to the process quickly, without interrupting it. ("warehouse" here is a generic term; in practice it is a specific entity, depending on the exact process concerned.) The problem arises when not all items that are used in the process can be stored in the nearby warehouse, as a result of its capacity limitation. The rest of the items are stored in a farther warehouse which has an unlimited

* Corresponding author. Tel.: +972-3-640-8389; fax: +972-3-640-7669.
E-mail address: tzur@eng.tau.ac.il (M. Tzur).

capacity, but from which they are not accessible to the process. We refer to these warehouses as the *primary* and *secondary* warehouses.

When an item which is in the secondary warehouse is required to the process, it has to be brought first to the primary warehouse. This transfer from the secondary to the primary warehouse is associated with waiting time that is costly to the process, and possibly also with routing costs. Furthermore, if the primary warehouse is full, one or more items that are stored there have to be moved to the secondary warehouse and this is again associated with the above costs.

This paper addresses the problem of which items to store in the primary warehouse at every stage of the process, where the process and the sequence of items required for it are predefined, and where the objective is to minimize the total costs associated with transferring items between the warehouses. Our main contribution is in dealing with items that may be of *different sizes*. This case 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. We prove that the problem is NP-Complete, thus resolving an open question raised by Crama (1997) with respect to the complexity of this problem. We then suggest two heuristic algorithms to solve the problem, inspired somewhat by two integer programming formulations to the problem; each of these heuristics is based on a different approach and provides insight that enhances our understanding of the problem. Our numerical study demonstrates that these heuristics perform very well, and therefore provide satisfactory solutions to the problem. In the rest of the introduction we review related literature.

This problem is discussed in the literature as the *tool switching* problem. There, tools are required during the operation of a flexible Numerically Controlled (NC) machine which has a tool magazine, capable of holding a limited number of tools. The production process is continuous as long as the required tool is in the tool magazine: an automatic arm is moving the required tool from the tool magazine to the tool holder on the machine.

The capacity of the tool magazine is determined by the number of slots it contains, and the common assumption in the literature is that each tool requires one slot. This paper treats the *general* tool switching problem, in which tools may require more than one slot. This is applicable in the case where a big tool, although held in one slot, is covering one or more of its neighbor slots, see Fig. 1, inspired by Stecke (1983). In another case, several tools may always be required together, and are therefore kept as a kit, requiring at least as many slots as the number of tools in the kit.

For example, in an application which we have observed in the metal industry, making a screw thread requires the use of a drill and three types of screw taps. In another case, drilling a hole in a certain size, where high precision is required, involves the use of three types of drills, each is finer than its predecessor, to achieve the highest precision required. In these cases, some or all of the tools that are used for the tasks described, are always kept together as a kit. In the first case, for example, none of the screw taps is ever used in isolation of the other two, and therefore none of them is ever transferred by itself from the tool magazine. The drill may be required for other tasks, and therefore may or may not be part of this kit; if it is, then a duplicate of it is required for other tasks, and this is sometimes done in practice. The result is a requirement for a "tool" whose size is either 3 or 4.

Another application described recently in the literature is that of Printed Circuit Board (PCB)

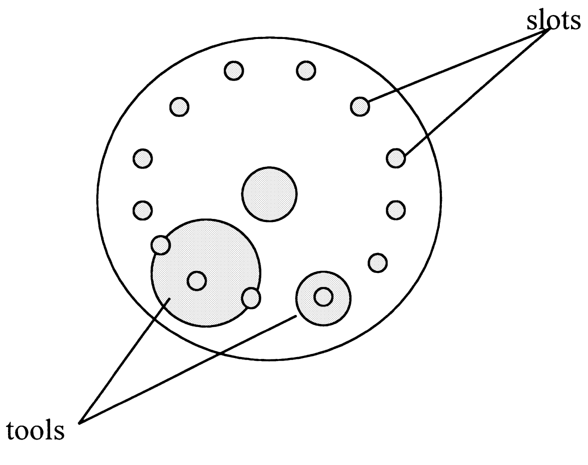

Fig. 1. Tool magazine.

assembly, see Jain et al. (1996). There, components that needed to be placed on a board were fed into the machine through a bank of feeder slots. Components occupied one or two slots and the bank of feeder slots had a limited capacity.

In the tool switching problem, jobs are processed on the machine in a given sequence, and each job may require several tools for its execution. For ease of exposition we assume first that a requirement is always for one tool/item only, and in Section 6 discuss how our formulations, analysis and heuristics are applicable for the more general case, with minor modifications.

Tang and Denardo (1988a) proved that the tool switching problem (with tools of equal size) can be solved in time that is linear in the number of requests and the total number of items by using the Keep Tools Needed Soonest (KTNS) policy. According to this policy, a tool is removed from the tool magazine only when it is full and another tool (not in the tool magazine) is required. Furthermore, the tools that remain in the tool magazine, are those for which the requirement is soonest.

Crama et al. (1994) presented an integer programming formulation of the problem, in which the constraint matrix is an *interval matrix* (a matrix which contains only 0 and 1 elements, and where the 1 elements on the columns are consecutive). An interval matrix is *totally unimodular*, i.e., its linear programming relaxation provides an integer solution, see for example, Nemhauser and Wolsey (1988). Crama et al. (1994) also showed that the integer programming problem can be solved by a *greedy algorithm* (see Hoffman et al., 1985) whose steps are identical to those performed in KTNS, which is another proof to the optimality of the latter. Privault and Finke (1995) reduce the problem where each pair of tools has a different switching cost to the problem of finding a minimum cost flow of maximum value in an acyclic network.

A related problem is that of *sequencing* the jobs to be performed on the machine so as to minimize the number of tool switches associated with the sequence (see Hertz et al., 1998; Crama et al., 1994; Tang and Denardo, 1988a; Bard, 1988). Another related problem is the *job grouping* problem in which all parts that need to be processed are divided into groups such that within groups it is not required to switch tools (see Crama and Oerlemans, 1994; Tang and Denardo, 1988b). The objective is to minimize the number of groups, since switching and waiting occurs when changing tools from one group to the next. This problem arises also in the context of Printed Circuit Board (PCB) production, but in the PCB problem, the switching costs (the waiting times) are related to the number and identity of items in the group (see Maimon and Shtub, 1991; Shtub and Maimon, 1992).

In a different context, the storage management problem is known as the *paging* problem. When managing the memory of a computer, data is organized in fixed-sized blocks called pages. The primary memory is limited to containing up to K pages, while the rest of the data is in the secondary memory. The objective is to minimize the number of switches of pages back and forth between the primary and secondary memory storage, subject to the availability of specific pages whenever they are needed for the running task. However, the paging problem is an *on-line* problem, since requests for specific pages are determined during the operation of the system, and are not known in advance. This calls for solutions of different types, see for example Deitel (1990) for algorithms suggested for the paging problem. The on-line version of the tool switching problem with non-uniform tools' sizes was recently analyzed in Matzliach and Tzur (1998).

2. Notation, formulation and complexity

In this section, we introduce the model terminology and notation which is used throughout the paper. We present a first mathematical programming formulation (in the next section we discuss an alternative formulation) and prove that the problem is NP-Complete.

We use the following notation:

$T =$ the number of item requirements during the process $(t = 1, \ldots, T)$.

Since the sequence of these item requirements is predetermined, one can think of its index as an index of *time periods*, where in each time period there is a requirement for an item. (Some of these requirements may be for the same item.)

$N =$ the number of different items involved in the process ($i = 1, \dots, N$).

$v_i =$ the size of item i.

$c_i =$ the transfer cost of item i.

$K =$ the capacity of the primary warehouse.

$d(t) =$ the tth item required during the process; alternatively, according to the convention of using time periods, $d(t)$ is interpreted as the identity of the item which is required in time period t.

An important special case of our problem is when $c_i = v_i$, that is, when the cost of transferring item i between the warehouses, in either direction, is proportional to its size (and the factor of proportionality may be assumed w.l.o.g. to be 1). We refer to this special case as *the proportional case*. For example, in the tool switching problem, there are T jobs that have to be processed on the machine. The tool which is required for the processing of job t is $d(t)$. Tool i is of size v_i, representing the number of slots it requires on the machine, and its transfer cost is c_i. Transferring a tool kit involves transferring each of the tools in the kit, and each transfer is associated with approximately the same time. When the transfer cost is proportional to the transfer time in which the machine is idle, this results in the proportional case. K is the capacity of the machine, representing the number of slots available in the tool magazine.

Our model assumes that the sizes of items are *additive*. That is, for a set of items, as long as the sum of their sizes does not exceed K, the set is feasible, regardless of the location of the items in the warehouse and/or the identity of the items. In the tool switching problem, this assumption is valid for tool kits, where the size of a tool may in fact equal the number of tools in the kit, and where no adjacency of the slots that occupy the tools of the kit is required. For big tools, adjacency of slots is required, therefore a possible additional rearrangement of the tool magazine may be required. This misrepresentation can be partially overcome by assigning the big tools higher transfer costs, resulting in an approximate transfer cost for those tools.

We use the following auxiliary definition:

$S_t =$ the set of items in the primary warehouse at time t, after the switches that may have been performed at time t, but before the switches that may be performed at time $t+1$. We refer to S_t from now on as *the state of the system*.

With this definition, the problem may be stated as determining the set S_t for *every* $t = 1, \dots, T$, such that the capacity of the primary warehouse is not exceeded. In every time period a possible action is transferring an item between the two warehouses, in either direction. Each transfer is associated with a cost.

Now we are ready to present the first mathematical programming formulation.

For all $t = 1, \dots, T$ and all $i = 1, \dots, N$, define the variable $x_{i,t}$ to be 1 if item i is in the primary warehouse at time t, and 0 otherwise:

$$x_{i,t} = \begin{cases} 1 & \text{if } i \in S_t, \\ 0 & \text{if } i \notin S_t. \end{cases}$$

And the problem is

$$\text{(P1)} \quad \text{Min} \quad \sum_{t=1}^{T}\sum_{i=1}^{N} c_i \cdot |x_{i,t} - x_{i,t-1}| \tag{1}$$

s.t.

$$x_{d(t),t} = 1, \quad t = 1, \dots, T, \tag{2}$$

$$\sum_{i=1}^{N} v_i x_{i,t} \leqslant K, \quad t = 1, \dots, T, \tag{3}$$

$$x_{i,0} = e_i, \quad i = 1, \dots, N, \tag{4}$$

$$x_{i,t} = 0,1, \quad i = 1, \dots, N, \quad t = 1, \dots, T, \tag{5}$$

where $e_i = 0,1$ for all $i = 1, \dots, N$ denote the initial state of the system, i.e., the items that are present at the primary warehouse at time 0.

Defining $y_{i,t} = |x_{i,t} - x_{i,t-1}|$, replacing the absolute value in the objective function by the y-variables and adding 2 constraints for every i and t,

$$y_{i,t} \geqslant x_{i,t} - x_{i,t-1} \quad \text{and} \quad y_{i,t} \geqslant -(x_{i,t} - x_{i,t-1}), \tag{6}$$

we get a 0–1 integer programming formulation. This is an intuitive formulation, but which contains O(NT) decision variables and O(NT) constraints. Solving the problem with this formulation is not efficient for practical problems. Indeed, the problem is hard, as the following theorem proves.

Theorem 1. *Our Storage Management Problem is NP-Complete, even in the proportional case.*

Proof. Our Storage Management Problem is clearly in NP. We now show a reduction from problem Partition, which is known to be NP-Complete (see for example, Garey and Johnson, 1979).

Partition. Given numbers $a_1, a_2, \dots a_n$, $\sum_{i=1}^{n} a_i = 2b$, does there exist a subset I s.t. $\sum_{i \in I} a_i = b$?

Given any Partition instance, define the following Proportional Storage Management (PSM) instance:

$T = n + 1;$

$N = n + 1;$

$c_i = v_i = a_i \quad \text{for } i = 1, \dots, n;$

$c_{n+1} = v_{n+1} = b;$

$K = 2b;$

$d(i) = i \quad \text{for } i = 1, \dots, n;$

$d(n+1) = n + 1.$

Does there exist a feasible solution to the PSM problem defined above with cost no more than $4b$?

We will show that the answer to the Partition problem is YES if and only if the answer to the PSM problem is YES.

(a) Assume first that in the Partition problem there exists a subset I s.t. $\sum_{i \in I} a_i = b$. Then consider the following solution to PSM: in $t = 1$ we insert to the primary warehouse items $1, \dots, n$; in period $n + 1$ we transfer out from the primary warehouse all items that are in I and insert item $n + 1$. Clearly this solution is feasible and its cost is $2b + b + b = 4b$. We conclude that the answer to the PSM problem is also YES.

(b) Assume now that in the Partition problem there does not exist a subset I s.t. $\sum_{i \in I} a_i = b$. In PSM we must insert during $t \leqslant n$ all items $1, \dots, n$ and by time $t = n + 1$ remove from the primary warehouse a subset of items with total size of at least b. Since we cannot find a subset whose total size is exactly b, we choose a subset J which satisfies $\sum_{i \in J} v_i \equiv a \geqslant b + 1$. By time $t = n + 1$ we also have to insert item $n + 1$. Therefore, the cost of any feasible solution is at least $2b + a + b$ which is at least $2b + b + 1 + b = 4b + 1$. We conclude that in this case, the answer to PSM is also NO.

We note that the NP-completeness of the problem stems from the fact that one cannot necessarily provide the exact space for the new item, as is the case with the Partition as well as the Knapsack problems.

3. A heuristic and a related alternative formulation

In this section we present our first heuristic, which is based on a representation of the problem in a matrix form. This representation also leads to an alternative 0–1 integer programming formulation, similar to the formulation presented by Crama et al. (1994). It is less obvious but more efficient than the previous formulation (given in Section 2) and is quite interesting.

3.1. The first heuristic

In the matrix form representation of the problem, we denote the value of the $x_{i,t}$ variables of the mathematical programming formulation (P1) in a matrix that has a row i for each item and a column t for each time period. To describe the heuristic, we first define the feasible solution which serves as the starting point of the heuristic.

The *naive solution* is a feasible solution which assigns to the primary warehouse at time period t *only* the item required at that time, that is, $d(t)$. According to this solution, at time t item $d(t-1)$ is transferred from the primary warehouse to the secondary, and item $d(t)$ is transferred in the opposite direction (unless $d(t-1) = d(t)$ in which case nothing has to be done).

For example, assume that $N = 3$, $T = 10$, $e_i = 0 \ \forall i$, and the required items for $t = 1, \dots, 10$

are: 1, 2, 1, 3, 2, 1, 2, 1, 3, 2, respectively. Then the naive solution in a matrix form is the following:

	i	$t=0$	1	2	3	4	5	6	7	8	9	10
	1	0	1	0	1	0	0	1	0	1	0	0
$X=$	2	0	0	1	0	0	1	0	1	0	0	1
	3	0	0	0	0	1	0	0	0	0	1	0

In this representation, every change from 0 to 1 or from 1 to 0 in row i of the matrix incurs a transfer cost of c_i; for a series of periods that starts and ends with 1 and has 0s in between, the cost is $2c_i$. This cost may be saved if in row i, the block of consecutive 0s is replaced by a block of consecutive 1s. A block of consecutive 1s means that the item remains in the primary warehouse in periods between two consecutive requirements for it. Note that replacing only *some* of the 0s of a given block into 1s, does not result in any savings at all, since the cost of transferring the item back and forth remains, as long as at least one 0 remains. We define an *improvement* as converting a block of consecutive 0s into 1s, which is associated with a saving of twice the transfer cost of the associated item. An improvement is *feasible* only if the capacity constraint in each of the periods whose value was replaced from 0 to 1, is not violated as a result of this change.

Note that every time period t is associated with one improvement, i.e., the block of 0s which follow the 1 in period t for item $d(t)$. Making this improvement means keeping (in the primary warehouse) item $d(t)$ until the next time it is needed. Therefore there are up to T possible improvements (the exact number depends on initial conditions and on whether there is a requirement for the same item in consecutive periods). We have to consider the *feasible improvements* and to choose among them, noting that an improvement may be feasible initially, but not after other improvements have been added to the initial solution. To choose among the possible improvements, we define the following parameter:

Let $q_t =$ the distance (number of time periods) between period t and the next period with a requirement for item $d(t)$ (i.e., the number of consecutive 0s in row $i = d(t)$, starting at time t).

$$\underset{\forall t\neq T}{q_t} = \begin{cases} m & \text{if } d(t+m+1) = d(t) \text{ and} \\ & \quad d(t+\tau) \neq d(t) \quad \forall \tau = 1,\ldots,m, \\ T-t & \text{if } d(t+\tau) \neq d(t) \\ & \quad \forall \tau = 1,\ldots,T-t. \end{cases} \tag{7}$$

The saving from inserting the improvement associated with time t is $2c_{d(t)}$. On the other hand, inserting a feasible improvement associated with period t consumes $v_{d(t)}$ units of the primary warehouse's capacity in all those time periods between the consecutive requirements; this consumption may transform other possible improvements from feasible to infeasible. Therefore, an optimal solution has to consider simultaneously the saving and the capacity consumption of all feasible improvements, leading to the alternative integer programming formulation, see Section 3.2 below.

Here we choose to *heuristically* weight the two factors (saving and consumption) for every feasible improvement, by defining the following improvement criterion:

$$\text{improvement criterion} = \frac{\text{cost saving}}{\text{total capacity consumption}} = \frac{2c_{d(t)}}{q_t v_{d(t)}} \tag{8}$$

and choosing the highest such value among all the feasible improvements. In the proportional case, the above improvement criterion chooses improvements in opposite relation to q_t, the distance between consecutive requirements. In the general case, the opposite relation is also factored by the cost/size ratio. In cases of ties, we choose arbitrarily.

To keep track of the available capacity in every period we define the following quantity:

$B(t) =$ the consumption (of the primary warehouse's capacity) at period t.

This quantity is being updated every time that an improvement is inserted to the solution.

The entire algorithm is described as follows:
Heuristic 1.

```
$S_t := \{d(t)\}, \quad B(t) := v_{d(t)}$
  for all $t = 1, \ldots, T$.
Put $t = 1, \ldots, T$ in a list of nonincreasing $c_{d(t)}/q_i v_{d(t)}$ values, breaking ties arbitrarily.

for $t$ on top of the list:
  if $B(\tau) + v_{d(t)} \leqslant K$    $\forall\, t+1 \leqslant \tau \leqslant t+q_t$
                                          then (improvement $t$ is chosen):
    $S_t := S_t \cup d(t)$              $\forall\, t+1 \leqslant \tau \leqslant t+q_t$
    $B(\tau) := B(\tau) + v_{d(t)}$     $\forall\, t+1 \leqslant \tau \leqslant t+q_t$
  else remove $t$ from the list
endfor
end
```

Most of the work in this algorithm is in executing the loop, in which for every t there may be up to N operations, resulting in a complexity of O(NT) for this part. In addition, sorting the $c_{d(t)}/q_t v_{d(t)}$ values takes $O(T \log_2 T)$ time. Therefore the total complexity of this algorithm is $O(NT + T \log_2 T)$, which is almost linear in NT. We give an illustrative example of the heuristic:

Example 1. The parameters are: $N = 3$, $T = 8$, $K = 7$, $e_i = 0\ \forall i$; the required items for $t = 1, \ldots, 8$ are: 1, 2, 3, 2, 1, 3, 2, 1;

$$v = c_1 = 3,\ v_2 = c_2 = 2\ v_3 = c_3 = 3.$$

Then the naive solution in a matrix form is the following:

	i	$t = 0$	1	2	3	4	5	6	7	8
	1	0	1	0	0	0	1	0	0	1
$X =$	2	0	0	1	0	1	0	0	1	0
	3	0	0	0	1	0	0	1	0	0

The initial consumption in each period is

$$B(1) = 3,\ B(2) = 2,\ B(3) = 3,\ B(4) = 2,$$

$$B(5) = 3, B(6) = 3,\ B(7) = 2,\ B(8) = 3.$$

Calculating the distances until the next requirement for each improvement t results in

$$q_1 = 3,\ q_2 = 1,\ q_3 = 2,\ q_4 = 2,\ q_5 = 2,$$
$$q_6 = 2,\ q_7 = 1;$$

and these values are placed in a non-decreasing list:

$$q_2 = 1,\ q_7 = 1,\ q_3 = 2,\ q_4 = 2,\ q_5 = 2,$$
$$q_6 = 2,\ q_1 = 3.$$

We now consider these improvements according their order in the list:

For $q_2 = 1$: $B(3) + v_{d(2)} = 3 + 2 = 5 < K$
$\Rightarrow$ insert, $B(3) = 5$;

For $q_7 = 1$: $B(8) + v_{d(7)} = 3 + 2 = 5 < K$
$\Rightarrow$ insert, $B(8) = 5$;

For $q_3 = 2$: $B(4) + v_{d(3)} = 2 + 3 = 5$
$< K$, $B(5) + v_{d(3)} = 3 + 3 = 5 < K$
$\Rightarrow$ insert, $B(4) = 5$, $B(5) = 6$;

For $q_4 = 2$: $B(5) + v_{d(4)} = 6 + 2 = 8 > K$
$\Rightarrow$ cannot insert;

For $q_5 = 2$: $B(6) + v_{d(5)} = 3 + 3 = 6 < K$,
$B(7) + v_{d(5)} = 2 + 3 = 5 < K$
$\Rightarrow$ insert, $B(6) = 6$, $B(7) = 5$;

For $q_6 = 2$: $B(7) + v_{d(6)} = 5 + 3 = 8 > K$
$\Rightarrow$ cannot insert;

For $q_1 = 3$: $B(2) + v_{d(1)} = 2 + 3 = 5 < K$,
$B(3) + v_{d(1)} = 5 + 3 = 8 > K$
$\Rightarrow$ cannot insert;

To conclude the algorithm, note that we have inserted the improvements that are associated with periods 2, 7, 3 and 5 with a total saving of $4 + 2 + 6 + 6 = 18$. The initial cost (of the naive solution) was $3 + 5 + 5 + 5 + 5 + 6 + 5 + 5 = 39$, thus the cost of the resulting solution is 21.

The matrix representing the resulting solution is the following:

	i	$t = 0$	1	2	3	4	5	6	7	8
	1	0	1	0	0	0	1	1	1	1
$X =$	2	0	0	1	1	1	0	0	1	1
	3	0	0	0	1	1	1	1	0	0

3.2. An alternative formulation

The matrix form representation of the naive solution suggests that an optimal solution may be achieved by optimally selecting a subset of all blocks of 0s. While in Section 3.1 a subset of blocks was *heuristically* chosen according to the improvement criterion, the next integer programming formulation selects an *optimal* set of improvements. In this formulation, each block of 0s is associated with a binary variable whose value is 1 if the improvement associated with this block is inserted to the solution, and 0 otherwise. There are $T-1$ such variables, since every demand (except the last one) is the beginning of a new block of 0s. (A block of 0s may be empty if two consecutive periods require the same item). The constraints state that in every time period the total capacity consumed by the subset of improvements does not exceed the available capacity at that time period, which is K minus the capacity of the required item; the latter is automatically in the primary warehouse, according to the definition of the naive solution.

We now define two more parameters that are used in the formulation: First, $m_{j,t}$ describes which time periods are contained in a given block of 0s:

$$m_{j,t} = \begin{cases} 1 & \text{if period } t \text{ is contained in block } j, \\ 0 & \text{otherwise.} \end{cases}$$

Second, r_j is a parameter describing the cost saving resulting from inserting improvement block j into the solution:

$$r_j = \begin{cases} 2c_{d(j)} & d(j+t) = d(j) \text{ for some } t \geqslant 2 \\ & \quad \text{(there exists future demand} \\ & \quad \text{for item } d(j)\text{)}, \\ c_{d(j)} & d(j+t) \neq d(j) \ \forall\ t = 1, \ldots, T-j \\ & \quad \text{(there is no additional demand} \\ & \quad \text{for item } d(j)\text{)}, \\ 0 & d(j+1) = d(j) \\ & \quad \text{(the next requirement is} \\ & \quad \text{for item } d(j) \text{ as well).} \end{cases}$$

Finally, the decision variable in the new formulation is defined as follows:

$$z_j = \begin{cases} 1 & \text{if improvement block } j \text{ is inserted} \\ & \text{to the solution,} \\ 0 & \text{otherwise.} \end{cases}$$

The resulting Integer Programming Formulation is given below:

$$\text{(P2)} \quad \text{Max} \quad \sum_{j=1}^{T-1} r_j \cdot z_j$$

s.t.

$$\sum_{j=1}^{T-1} v_{d(j)} \cdot m_{j,t} \cdot z_j \leqslant K - v_{d(t)}$$

$$\forall t = 1, \ldots, T,$$

$$z_j = 0, 1.$$

The new formulation has only $O(T)$ variables and constraints, as opposed to $O(NT)$ in the previous formulation, (P1). In a few examples for $N=25$ and $T=100$ that we solved on a 486-66 MHZ personal computer, the running time using this formulation was about 2 hours, compared to about 24 hours when using (P1). Crama et al. (1994) presented a similar formulation for the tool switching problem with tools of equal size. In their formulation the constraint matrix includes only 0 and 1 elements, and can be shown to be totally unimodular.

4. A second heuristic

In this section we present another heuristic algorithm, which is based on a totally different approach towards obtaining a solution to the problem. We discuss the advantages of each approach in the next section, when we present our numerical study.

The main difference between the two heuristics is in the type of information that is used to make decisions. While Heuristic 1 takes a global look at the system, and chooses among all possible improvements, Heuristic 2 is an iterative procedure, where every time period is an iteration. In every period, decisions are made only with respect to that period, and then the algorithm proceeds to the next period. We emphasize that although the procedure is iterative as described, it is still using the fact that the model is static, and all demands are known in advance. When decisions are made, this information is used, as we show next.

One can prove that there exists an optimal solution which does not change the state of the system as long as the required item is in the primary warehouse (postponing an action that contradicts this property leads to an alternative optimal solution). When the required item is not in the primary warehouse, the decision that needs to be taken is which item/s to transfer, in order to make enough room for the required item. In Heuristic 2, this decision is based on a criterion that generalizes the KTNS principle which removes the tool which is required furthest in the future, as discussed in Section 1. In our problem, we also have to consider the transfer cost of an item. Moreover, we have to consider the capacity that each item occupies, and we may remove a set of items. The idea of Heuristic 2 is to calculate for every set of items (that makes enough room for the required item) a *weighted* distance until the next requirement, and choose the largest.

We start by defining for every item $i \in (S_{t-1} \setminus d(t))$, the following value: $\mathrm{dist}(i,t) =$ the number of periods from time t until the next time this item is required. If there is no additional requirement for item i, this value is set equal to T.

The value of $\mathrm{dist}(i,t)$ is found according to the following formula:

$$\underset{\substack{i \in S_{t-1} \\ i \neq d(t)}}{\mathrm{dist}(i,t)} = \begin{cases} m & d(t+m) = i,\ d(t+\tau) \neq i \\ & \quad \text{for } \tau = 1,\ldots,m-1, \\ T & d(t+\tau) \neq i \quad \forall \tau = 1,\ldots,T-t. \end{cases} \tag{9}$$

Note that here the block of 0s includes the 0 of period t (since $i \neq d(t)$), so that this distance is from a point of view of the end of period $t-1$.

Based on the distance for a single item, we define a *weighted* distance for a *set* of items $J \subseteq (S_{t-1} \setminus d(t))$ as follows:

$$w(J,t) = \frac{\text{average distance (from time } t\text{) of items in the set } J}{\text{sum of the transfer costs of all items in the set } J} = \frac{\sum_{i \in J} \mathrm{dist}(i,t)/|J|}{\sum_{i \in J} c_i}. \tag{10}$$

The reason that for distances we take an average value while for costs we take a sum is that intuitively we would like to get for a set of items, value with a similar meaning as for the single item: with respect to the item: with respect to the item distance – the *average* item distances indicate the attractiveness of the set; with respect to the item transfer cost – we prefer to remove a set of items with a smaller *sum* of transfer cost as long as it is feasible.

We use again the quantity $B(t)$ to denote the consumption at period t, and check the feasibility of every set of items J by checking if the following condition is satisfied:

$$B(t-1) + v_{d(t)} - \sum_{i \in J} v_i \leqslant K. \tag{11}$$

If a set J satisfies the capacity constraint, it is a candidate to be transferred to the secondary warehouse. Finally, to ensure that the heuristic's complexity remains polynomial, we impose a limit, L, on the number of items that can be transferred simultaneously from the primary warehouse. We denote by $Y_L^{S_{t-1}}$ all sets of up to L items which are subsets of S_{t-1}. The choice of L is guided first by the computational complexity required to consider all sets J in $Y_L^{S_{t-1}}$ (see below), and by the expected heuristic's performance as a function of L, see our numerical example below. Another consideration is feasibility, where we need to ensure that in all cases we will be able to find a set of up to L items that free enough capacity. For example, a choice of $L = \max_i(v_i)/\min_i(v_i)$ always satisfies the constraint.

Among all sets that belong to $Y_L^{S_{t-1}}$ and which satisfy the capacity constraint, we choose to transfer from the primary warehouse the set with the *highest value of weighted distance*, $w(J,t)$. Note that according to this criterion, when all sets have equal transfer cost, we choose the set with the largest average distance; in addition, if the items are all of equal size, the set achieving the highest $w(J,t)$ is a set of one item only, whose distance is the largest, thus leading to the KTNS policy which is known to be optimal in this case. When the weighted distance is identical for all sets, we choose the set with the smallest sum of transfer costs. Therefore, the criterion provides the desired measure in these special cases. The entire algorithm is described as follows.

```
Heuristic 2.
$S_0 := \phi, B(0) := 0$;
for $t = 1, \ldots, T$:
    if $d(t) \in S_{t-1}$ then $S_t := S_{t-1}$, $B(t) := B(t-1)$;
    elseif $B(t-1) + v_{d(t)} \leqslant K$ then $S_t := S_{t-1} \cup d(t)$, $B(t) := B(t-1) + v_{d(t)}$;
    else compute $J^* \in Y_L^{S_{t-1}}$ such that:
        $w(J^*, t) = \max\{w(J,t) : J \in Y_L^{S_{t-1}}$ and $B(t-1) + v_{d(t)} - \sum_{i \in J} v_i \leqslant K\}$;
        $S_t := S_{t-1} \setminus J^* \cup d(t)$,    $B(t) := B(t-1) + v_{d(t)} - \sum_{i \in J^*} v_i$;
    endif
endfor
```

Most of the work in Heuristic 2 is associated with computing J^*. The $\mathrm{dist}(i,t)$ values are calculated in $\mathrm{O}(NT)$ time through a recursive procedure (for every t, $\mathrm{dist}(i,t)$ is found by subtracting 1 from $\mathrm{dist}(i,t-1)$, except for the new item in the primary warehouse, $d(t-1)$, which has to be calculated directly from (9)). The number of sets in $Y_L^{S_{t-1}}$ that have to be considered is

$$\binom{|S_{t-1}|}{1} + \binom{|S_{t-1}|}{2} + \cdots + \binom{|S_{t-1}|}{L} = \mathrm{O}((\min\{N,K\})^L)$$

(assuming K is an integer, possibly after appropriate scaling) which is polynomial in N, but exponential in L.

The performance of the heuristic as a function of L is an important issue. As L increases we expect the solution to improve since the algorithm may choose a set to be transferred among larger number of different sets. On the other hand, we expect that for large values of L, the benefit from increasing L will be negligible. Table 1 demonstrates the results of an empirical analysis that we performed to verify this point. We ran the algorithm for the proportional case with $N = 25$, $T = 100$, $K = 100$ and where all items have equal probability to be required in every period. The items' sizes were chosen randomly from a uniform distribution with varying intervals, as specified in Table 1.

Table 1
The impact of L on the heuristic's performance

L	Interval			
	(10, 10)	(7, 13)	(5, 15)	(1, 19)
1	562	604	691	831
2	562	581	602	658
3	562	580	594	606
4	562	580	592	601

We see from Table 1 that the performance of the heuristic indeed improves as we increase L, and for $L = 4$ most or all of the possible cost reduction has been achieved. Moreover, increasing L has a more significant impact as the standard deviation of the items' sizes and transfer costs becomes larger (the range of the interval is larger). This is explained by noting that with a large standard deviation one item may be replaced by a set of items with the appropriate sum of their sizes, that have the largest weighted distance. As the standard deviation increases, there are more such sets to choose from. In the extreme case when the standard deviation is 0 (the interval (10,10)), L has no impact on the performance of the heuristic, since we get the case with equal items' sizes and costs, for which transferring one item for each new item that is required, is optimal. We give an illustrative example of the heuristic.

Example 2. We consider the same data as in Example 1 and set $L = 2$. $S_0 = 0$, $B(0) = 0$.

$t = 1$: $(d(1) = 1) \Rightarrow$ insert item 1, $S_1 = \{1\}$, $B(1) = 3$ (cost = 3).

$t = 2$: $(d(2) = 2) \Rightarrow$ insert item 2, $S_2 = \{1,2\}$, $B(2) = 5$ (cost = 2).

$t = 3$: $(d(3) = 3) B(2) + v_{d(3)} > K$,
$\Rightarrow$ need to remove a set of one or two items;

$$\mathrm{dist}(1,3) = 2, \quad \mathrm{dist}(2,3) = 1,$$
$$w(\{1\},3) = \frac{\mathrm{dist}(1,3)}{v_1} = \frac{2}{3} = 0.67,$$

$$w(\{2\},3)=\frac{\mathrm{dist}(2,3)}{v_2}=\frac{1}{2}=0.5,$$

$$w(\{1,2\},3)=\frac{(\mathrm{dist}(1,3)+\mathrm{dist}(2,3))/2}{v_1+v_2}=\frac{3}{10}=0.3.$$

The largest weighted distance is obtained for $w(\{1\},3)$, therefore: transfer item 1 out, transfer item 3 in; $S_3=\{2,3\}$, $B(3)=5$ (cost = 6).

$t=4$: $(d(4)=2)$ $d(4)\in S_3 \Rightarrow$ do nothing.

$S_4=\{2,3\},\ B(4)=5.$

$t=5$: $(d(5)=1)$ $B(4)+v_{d(5)}>K$,

$$\mathrm{dist}(2,5)=2,\quad \mathrm{dist}(3,5)=1,$$

$$w(\{2\},5)=\frac{\mathrm{dist}(2,5)}{v_2}=\frac{2}{2}=1,$$

$$w(\{3\},5)=\frac{\mathrm{dist}(3,5)}{v_3}=\frac{1}{3}=0.33,$$

$$w(\{2,3\},5)=\frac{(\mathrm{dist}(2,5)+\mathrm{dist}(3,5))/2}{v_2+v_3}=\frac{3}{10}=0.3,$$

therefore: transfer item 2 out, transfer item 1 in; $S_5=\{1,3\}$, $B(5)=6$ (cost = 5).

$t=6$: $(d(6)=3)$ $d(6)\in S_5 \Rightarrow$ do nothing.

$S_6=\{1,3\},\ B(6)=6.$

$t=7$: $(d(7)=2)$ $B(6)+v_{d(7)}>K$,

$$\mathrm{dist}(1,7)=1,\quad \mathrm{dist}(3,7)=8,$$

$$w(\{1\},7)=\frac{\mathrm{dist}(1,7)}{v_1}=\frac{1}{3}=0.33,$$

$$w(\{3\},7)=\frac{\mathrm{dist}(3,7)}{v_3}=\frac{8}{3}=2.67,$$

$$w(\{2,3\},7)=\frac{(\mathrm{dist}(1,7)+\mathrm{dist}(3,7))/2}{v_1+v_3}=\frac{9}{12}=0.75,$$

therefore: transfer item 3 out, transfer item 2 in; $S_7=\{1,2\}$, $B(7)=5$ (cost = 5).

$t=8$: $(d(8)=1)$ $d(8)\in S_7 \Rightarrow$ do nothing.

$S_8=\{1,2\},\ B(8)=5.$

End of problem. Cost $=3+2+6+5+5=21$.

The matrix representing the resulting solution is the following:

	i	$t=0$	1	2	3	4	5	6	7	8
	1	0	1	0	0	0	1	1	1	1
$X=$	2	0	0	1	1	1	0	0	1	1
	3	0	0	0	1	1	1	1	0	0

For this example, both heuristics resulted in the same solution, although it is apparent that the sequence of decisions was different.

5. Numerical study

In this section we evaluate the heuristic methods presented in Sections 3 and 4 by comparing their solution values to the optimal solution value (obtained by solving the exact formulation of the problem) for problems with various characteristics.

In addition, we compare our results to two other simple solutions: the naive solution (defined in Section 3) and the random solution, defined as follows: transferring is performed only when the required item is not in the primary warehouse, and not enough capacity is available for it. In that case, one of the items in the primary warehouse is chosen randomly (with an equal probability for each item), and transferred out. If this does not free enough capacity for the required item, another item is transferred out in the same way, until the required item can be brought to the primary warehouse. The random solution is a reasonable algorithm when no other alternatives are available; we see below that when a larger capacity of the primary warehouse is available, the random solution utilizes it.

On the other hand, the cost of the naive solution is independent of K by definition, since this solution does not take advantage of the available capacity, but rather transfers out every item after its use. The naive solution is not intended as an attractive or a reasonable policy, but rather as a benchmark representing the worst possible case.

In all our test problems we used the proportional case with $N=25$, $T=100$ and $\max_i(v_i)\leqslant K\leqslant 250$. In problem sets 1–3, the items' sizes (and

Table 2
Problem set 1 – low standard deviation

Cost	K					% deviation	
	20	50	100	150	200	Average deviation	Maximum deviation
Naive solution C_N	1890	1890	1890	1890	1890	Not relevant	Not relevant
Random solution C_R	1870	1610	1240	875	454	161%	312%
Optimal solution C_{opt}	1719	1118	594	312	110	–	–
Heuristic 1 C_{H1}	1725	1123	612	320	116	2.8%	5.4%
Heuristic 2 C_{H2}	1748	1121	601	317	112	1.2%	1.8%

Remark: For every K, average of 10 instances.

costs) were chosen from a uniform distribution with expected item size of 10 and varying standard deviation; each item had an equal probability of being required in each time period. In Heuristic 2 we used $L = 3$.

Table 2 presents the cost of the naive, random, optimal, Heuristic 1 and Heuristic 2's solutions for problem set 1 in which the standard deviation of the items' sizes was low (uniform between 7 and 13). The table also summarizes the average and maximum deviations of the random and heuristic solutions from the optimal solution value, for $50 \leqslant K \leqslant 200$, where the maximum deviation is the maximum of the averages computed for each value of K. We note that for both heuristics as well as for the random solution, the maximum deviation is obtained for a larger value of K ($K = 200$); we believe that the reason is that in this case, the total cost is much lower, therefore, every deviation is large in percentages. However, these are not the most interesting or difficult cases.

The cost of the naive, random, optimal and heuristics' solutions are also presented in Fig. 2; the optimal and the heuristics' solutions are on the same line, since their differences are very minor and indistinguishable. We can see from the figure that the value of the random solution decreases almost linearly with K where the heuristics and the optimal solution's values are considerably below that.

Tables 3 and 4 summarize the costs of all solutions and the deviations of the random and heuristic solutions for problem sets 2 and 3 in which the standard deviation of the items' sizes is medium (uniform between 5 and 15) and high (uniform between 1 and 19), respectively.

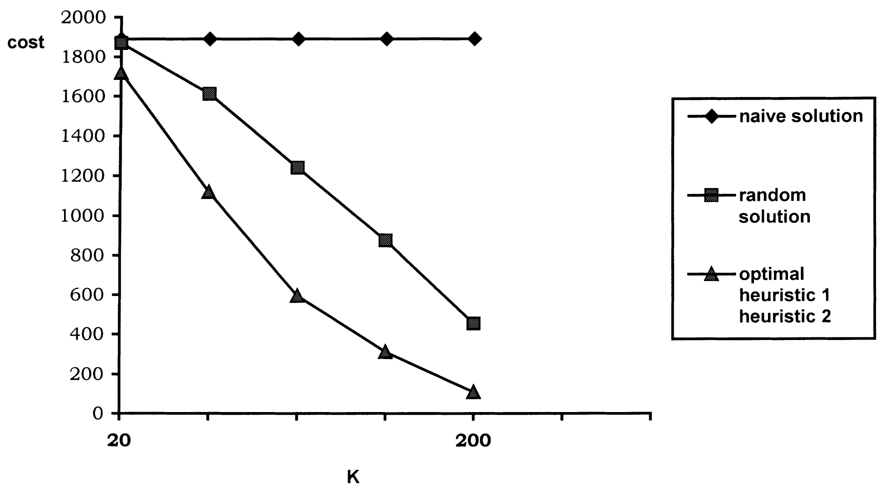

Fig. 2. Behavior of the cost as a function of K.

Table 3
Problem set 2 – medium standard deviation

Cost	K					% deviation	
	20	50	100	150	200	Average deviation	Maximum deviation
Naive solution C_N	1900	1900	1900	1900	1900	Not relevant	Not relevant
Random solution C_R	1869	1669	1217	863	417	160.2%	308%
Optimal solution C_{opt}	1750	1148	584	310	102	–	–
Heuristic 1 C_{H1}	1757	1156	590	315	106	1.8%	3.9%
Heuristic 2 C_{H2}	1764	1158	587	320	102	1.2%	3.2%

Remark: For every K, average of 10 instances.

Table 4
Problem set 3 – high standard deviation

Cost	K					% deviation	
	20	50	100	150	200	Average deviation	Maximum deviation
Naive solution C_N	1928	1928	1928	1928	1928	Not relevant	Not relevant
Random solution C_R	1790	1733	1312	753	506	136%	274%
Optimal solution C_{opt}	1726	1225	658	325	135	–	–
Heuristic 1 C_{H1}	1732	1232	666	329	136	0.9%	1.2%
Heuristic 2 C_{H2}	1747	1257	669	337	136	2.2%	2.6%

Remark: For every K, average of 10 instances.

Tables 2–4 show that both heuristics perform very well, with average deviations of no more than 2.8%, and maximum deviation of no more than 5.4% on an average, for large K. The random solution does not perform well, with an average deviation of about 150% above the optimal solution.

When comparing Heuristics 1 and 2, Heuristic 2 has a slight advantage for problems with low standard deviation of the items' size, as opposed to problems with high standard deviation of items' size for which Heuristic 1 has a slight advantage. We believe that it is the difference in the basic approach (the global look of Heuristic 1 as opposed to the iterative procedure of Heuristic 2) which gives Heuristic 1 the advantage in problems with high standard deviation of items' size.

In other problem sets that we have tested, the item's probability to be required in a given period, was positively (problem set 4) and negatively (problem set 5) correlated with the item's size. Again, both heuristics performed very will with an average deviation of no more than 2% and maximum deviation of no more than 5% on an average; here, Heuristic 2 was slightly better in both cases. Finally, for items of equal size, the average and maximum deviation of Heuristic 1 were 0.4% and 0.7%, respectively, where for Heuristic 2 we obtained the optimal solution.

The running time of both heuristics is very fast (less than 1 minute on a 486-66MHZ computer). Heuristic 1 is a bit faster, since it is a one pass method, while Heuristic 2 has to choose for every t from as many sets as determined by the choice of L and the size of S_{t-1}.

6. Generalization of multi-item requests

In this section we demonstrate how our heuristics can be generalized to the case where several items are requested simultaneously, that is, in the same time period. This case is of great relevance to the tool switching problem, where a given operation may require more than one tool, and all the required tools have to be on the tool magazine when the operation is performed. In Section 6.1 we give a sketch of the required modifications, the details are then relatively straightforward; in Section 6.2 we present the results of a numerical study, designed to evaluate the performance of the modified heuristics.

6.1. Modifications of the heuristics

We modify our notation as follows:
T continues to represent the number of requests and we continue to associate $t = 1, \dots, T$ with time periods, but every request may consist now of *up to p items.*
$d(t)$ is now a *set* of items required at time t; $d(t) = \{d^1(t), d^2(t), \dots, d^{p_t}(t)\}$, where $p_t \leqslant p$ is the *number* of items required at time t.

Modifying Heuristic 1
In the matrix representation of the problem (see Section 3.1), every column t may contain more than one element which equals 1. This implies that there may exist more than T possible improvements (in fact, the number of improvements is $\sum_{t=1}^{T} p_t \leqslant pT$). While before, each time period was associated with an improvement, now every pair of t and $d^i(t) \in d(t)$ (a time period and an item which is required in that time period), are associated with an improvement.

For every period t, an analogous of Eq. (7) is now defined for every $d^i(t) \in d(t)$, and the resulting values are denoted as $q_t^1, q_t^2, \dots, q_t^{p_t}$, where q_t^i is the distance between period t and the next period with a requirement for item $d^i(t) \in d(t)$.

An analogy of Eq. (8) results in an improvement criterion for every pair of time/item which defines an improvement, i.e., for every t and every $d^i(t) \in d(t)$:

$$\text{improvement criterion} = \frac{\text{cost saving}}{\text{total capacity consumption}} = \frac{2c_{d^i(t)}}{q_t^i v_{d^i(t)}}.$$

The heuristic again chooses the improvement which is associated with the highest improvement criterion, and therefore the exact algorithm is described in a similar way.

Modifying Heuristic 2
The only modifications required here are the following:

1. The set of items transferred out in period t is chosen such that the resulting free capacity in the primary warehouse can contain *all* items that belong to $d(t)$.
2. The set of items transferred out in period t contain items that are in the primary warehouse, *and which do not belong to d(t).*

The rest of the heuristic is unchanged. With these modifications, however, the parameter L would typically be large than in the case where only one item is required every period, therefore the actual complexity would increase. We expect, however, that in practical applications, the value of p (the maximum number of items required every period), would be reasonably small, since we do not expect to replace in every period a large part of all items in the primary warehouse. When p is small and bounded, the complexity of the algorithm is still bounded by a polynomial. To reduce the number of sets that are candidates to be transferred out, L may be chosen dynamically in each iteration, according to the amount of space that needs to be freed. For example, if at least b_t, units have to be transferred out of the primary warehouse in period t, we may set in period t: $L_t = \lceil b_t / \min_i(v_i) \rceil$.

An alternative way of modifying Heuristic 2 is by replacing each period t by p_t periods, each of which requires one item. Then, the only modification required is that in none of the p_t periods would item $d^i(t) \in d(t)$ be transferred out of the primary warehouse. This modification does not increase the algorithm's complexity, but has the disadvantage of performing an iteration of period t sequentially rather than simultaneously.

Note that the modifications for the heuristics are necessary only when the items may be transferred from the primary warehouse in isolation from each other; otherwise (when they always have to be kept together), they may be treated as one big item, as is the case with tool kits discussed in the introduction.

6.2. Numerical study for the generalized case

We conducted an additional computational study, designed to evaluate the performance of the modified heuristics for the general case. As in the previous study, we compare the solutions obtained by the heuristics to the value of the optimal solution, obtained by solving the exact formulation of the problem. The latter is obtained by a straightforward generalization of problem (P2) in Section 3; the generalized formulation contains a larger number of decision variables, determined by the total number of items requested. As a result, the general problem is more difficult to solve optimally, and in a few cases it look several days to obtain the optimal solution.

In all our test problems we used the proportional case with $N = 100$, $T = 100$ and $K = 100$ or 150. The value of p, the maximum number of items requested simultaneously, was set to 6 (Table 5) or 10 (Tables 6 and 7), where the number of items requested in every time t, p_t, was uniformly distributed over the interval $[1, p]$; for every t, all items had an equal probability of being requested. Note that when a big tool represents several tools that form a kit, p_t is in fact associated with a larger number of "real" tools.

We applied Heuristic 2 with 3 values of L, see Tables 5–7; the numbers in the tables represent the average and the maximum deviations from

Table 5
% deviation from the optimal solution for $p = 6$ (average, maximum)

Heuristic	K					
	100			150		
Heuristic 1	(0.9, 2.0)			(0.7, 1.8)		
Heuristic 2	$L = 6$	$L = 8$	$L = 10$	$L = 6$	$L = 8$	$L = 10$
	(3.8, 7.0) [a]	(3.6, 7.0)	(3.6, 7.0)	(2.5, 4.4)	(2.5, 4.4)	(2.5, 4.4)
Average CPU seconds	2	9	24	18	177	941

[a] In 10% of the problems the execution of the heuristic was not possible due to the restriction of the L value.

Table 6
% deviation from the optimal solution for $p = 10$ (average, maximum)

Heuristic	K					
	100			150		
Heuristic 1	(1.0, 1.8)			(0.9, 1.8)		
Heuristic 2	$L = 6$	$L = 8$	$L = 10$	$L = 6$	$L = 8$	$L = 10$
	[a]	(5.5, 8.2) [b]	(5.4, 7.9) [c]	(4.7, 6.7)	(3.8, 5.0)	(3.4, 4.9)
Average CPU seconds	1	18	64	27	353	2415

[a] In all 100% of the problems the execution of the heuristic was not possible due to the restriction of the L value.
[b] In 20% of the problems the execution of the heuristic was not possible due to the restriction of the L value.
[c] In 10% of the problems the execution of the heuristic was not possible due to the restriction of the L value.

Table 7
% deviation from the optimal solution for $p = 10$ (average, maximum) with low standard deviation of items' sizes

Heuristic	K					
	100			150		
Heuristic 1	(1.2, 2.0)			(0.9, 1.9)		
Heuristic 2	$L = 6$	$L = 8$	$L = 10$	$L = 6$	$L = 8$	$L = 10$
	[a]	(3.2, 4.6) [b]	(3.3, 4.7)	(2.9, 4.4) [c]	(2.3, 3.4)	(2.4, 3.7)
Average CPU seconds	N/A	5	13	9	178	928

[a] In all 100% of the problems the execution of the heuristic was not possible due to the restriction of the L value.
[b] In 20% of the problems the execution of the heuristic was not possible due to the restriction of the L value.
[c] In 60% of the problems the execution of the heuristic was not possible due to the restriction of the L value.

optimality of 10 instances of the problem characteristics specified. In addition, we specify for Heuristic 2 the average CPU time (in seconds) that was required to execute the algorithm on a Silicon Graphics ORIGIN200 workstation, 180 MHz; for Heuristic 1 all algorithm runs took less than 1 second. In Tables 5 and 6 the items' sizes (and costs) were chosen from a uniform distribution over the interval $[1, 10]$, and in Table 7 from a uniform distribution over the interval $[4, 7]$.

The results demonstrate that in all problem sets examined, both heuristics performed very well, with only a few percentage deviation from optimality. Heuristic 1 performed better than Heuristic 2, with an average deviation of less than 1%, and as mentioned in less than 1 second. The time to execute Heuristic 2 depended both on the problem parameters, and in particular on the value of L.

These results indicate that for the general case, Heuristic 1 performs better than Heuristic 2. However, as can be observed from Table 7, when the standard deviation of the items' sizes decreases, the gap between the two heuristics becomes smaller, a trend which was also observed in the results of the numerical study of the basic problem. Since the problems solved here are relatively large, we expect that Heuristic 2 would still take a reasonable amount of time in most real applications, and recommend to use both heuristics in order to choose the better solution.

7. Summary

In this paper we provided for the first time an extensive analysis of the storage management problem of items with unequal sizes, in two warehouses with different levels of availability. This problem has been studied in the context of the tool switching problem on a flexible NC-machine, mostly for the uniform tool size case. The problem arises similarly in other cases in which tools/items are used in a production or a service operation and where the immediate/primary storage area is not capable of containing all tools/items that may be required for the process.

We proved that the problem is NP-Complete, thus resolving the open question with respect to its complexity. We further developed two heuristic procedures that were shown to perform well in a numerical study. When only one item is required in each request, no one heuristic performed better than the other in all cases considered, and in fact, one should consider using both procedures and choosing the better solution (their running time and coding complexity are very minor). In the generalized case, when several items may be requested simultaneously, one of the heuristics (denoted Heuristic 1 in our paper) performed better than the other in terms of optimality gap as well as running time, but the gaps between the two heuristics are smaller for some problem characteristics.

An interesting problem for future research is determining the *sequence* of tasks to be performed in the process, if possible. The objective of such a sequence is to minimize switching costs that will follow once the process is executed.

References

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

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

Crama, Y., 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., Kolen, A.W.J., Oerlemans, A.G., 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.

Deitel, H.M., 1990. Operating Systems, 2nd ed. Addison-Wesley, Reading, MA.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability. Freeman, New York.

Hertz, A., Laporte, G., Mittaz, M., Stecke, K.E., 1998. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Transactions 30 (8), 689–694.

Hoffman, A.J., Kolen, A.W.J., Sakarovitch, M., 1985. Totally-balanced and greedy matrices. SIAM Journal on Algebraic and Discrete Methods 6 (4), 721–730.

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

Maimon, O., Shtub, A., 1991. Grouping methods for printed circuit board assembly. International Journal of Production Research 29 (7), 1379–1390.

Matzliach, B., Tzur, M., 1998. The online tool switching problem with non-uniform tool size. International Journal of Production Research 36 (12), 3407–3420.

Nemhauser, G.L., Wolsey, L.A., 1988. Integer and combinatorial optimization. Wiley, New York.

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

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

Shtub, A., Maimon, O., 1992. Role of similarity measures in PCB grouping procedures. International Journal of Production Research 30 (5), 973–983.

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., 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., 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.