INT. J. PROD. RES., 1998, VOL. 36, NO. 12, 3407–3420

The online tool switching problem with non-uniform tool size

B. MATZLIACH‡ and M. TZUR†*

The tool switching problem with non-uniform tool sizes is considered in this paper for the first time in a dynamic (online) environment. We develop three heuristics, designed for various assumptions with respect to the randomness of the process; the heuristics are based on the insight and experience that we obtained from the static problem with non- uniform tool sizes. Our computational experiments which examined various test problems with different underlying assumptions indicate that our heuristics are performing well, that is, they are reasonably close to the static solution and perform much better than a random solution.

1. Introduction and literature review

Flexible and automated manufacturing systems are assisted by tool management procedures; tool switching is a basic issue of tool management, addressed at the individual machine level. An automated machine which processes various types of parts requires certain tools which are located on the machine's tool magazine. The total number of tools required to process a set of parts is typically larger than the available magazine storage capacity. Therefore, when a certain tool is required, if it is not loaded on the tool magazine, switching of tools occurs between the tool magazine and a centralized tool storage area. The latter can hold (practically) any number of tools.

A tool switching strategy states which (if any) tools to change on the machine's tool magazine prior to processing each part. Tool switching is a time consuming operation which is costly to the process due to machine down-time; also, there may be costs associated with moving tools to and from the machine. The objective is to minimize the total cost of tool switches.

The tool switching problem has received considerable attention from researchers, see for example Bard (1988), Tang and Denardo (1988), Crama *et al.* (1994) and Privault and Finke (1995). It is assumed in these papers that processing each part may require several tools, but all tools are of uniform size, occupying one slot of the tool magazine. The latter assumption simplifies the problem considerably, and it was shown by Tang and Denardo (1988) that the policy KTNS (Keep Tools Needed Soonest) is optimal for this case. Some of the extensions considered include non-uniform switching times (Crama *et al.* 1994, Privault and Finke, 1995) as well as addressing the related problem of determining the optimal sequence of parts to be processed on the machine.

Matzliach and Tzur (1997) analysed the tool switching problem in which the tool sizes are non-uniform. As explained there, non-uniform tool sizes are of interest in

Revision received June 1998.
† Industrial Engineering Department, Tel Aviv University, Tel Aviv 69978, Israel.
‡ Shimoni 17, Ramat Aviv, Tel Aviv, Israel.
* To whom correspondence should be addressed.

the case where a big tool, although held in one slot, covers one or more of the neighbouring slots, see Stecke (1983), Shanker and Tzen (1985) and Jain *et al.* (1996). 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. Then, the kit may be considered as one tool, whose size is the sum of sizes of all tools that are included in the kit. Matzliach and Tzur (1997) considered the problem in the static environment, where the sequence of parts to be processed on the machine is given and known in advance, in which case the switching policy can also be determined in advance for the entire process. They proved that the problem is NP-complete, and developed two heuristic procedures. All of the previous work on the tool switching problem assumed a static environment as well.

In this paper we analyse, for the first time, the dynamic (online) tool switching problem in which parts that need to be processed on the machine arrive randomly, and tool sizes are non-uniform. In an FMS environment, static planning (at least for the next shift or so, as discussed in the literature), is the ideal situation whenever possible. However, random events such as breakdowns may require planning to become dynamic. For example, when a machine breakdown occurs in a system which consists of several flexible machines, jobs that cannot proceed on that machine as planned, are sent to alternative machines unexpectedly. Alternatively, dynamic planning results from a (first-in first-out) FIFO scheduling rule which is frequently used in a make-to-order environment. FIFO ensures equal service to all customers as well as short lead times in a flexible environment. The resulting identity and order of jobs is again unknown in advance.

The online arrival of parts and therefore tool requests calls for the design of an online algorithm to control the tool switching strategy. Here we suggest such algorithms and examine their performance. Our computational experiments indicate that a significant improvement is achieved by the suggested procedures, compared with reasonable and simple rules. We see the main contribution of this paper as being both in presenting and modelling a situation which was not addressed before, as well as in providing solutions that are shown to perform well.

A special case of our online tool switching problem is the *paging* problem, a fundamental online problem in computer science. It involves a fast memory, which can hold K pages of a standard size, and a slow memory, representing the virtual memory, which can hold a large (practically unlimited) number of pages. A sequence of requests for pages is presented during the execution of a task; if the requested page is in the fast memory, no cost is incurred and the task proceeds with no interruptions. Otherwise, the algorithm has to bring the requested page from the slow to the fast memory, and a unit cost is incurred. The decision then is which of the K pages in the fast memory to move to the slow memory, to make room for the requested page. In the case of paging, the offline problem is very simple. The algorithm MIN which always removes from the fast memory the page whose next request is furthest in the future, is optimal, see Belady (1966).

However, the paging problem is practically important only when no knowledge on future requests for pages is available, which calls for online algorithms for the problem. One commonly used online paging algorithm is called *Least Recently Used* (*LRU*), see for example Coffman and Denning (1973). According to the LRU algorithm, when the requested page is not in the fast memory, the page removed from the fast memory to make room for it is the one whose most recent access was earliest. A

major advantage mentioned with respect to this algorithm is its simplicity, as well as its modest memory requirements.

In Irani and Karlin (1997) it was shown that for the paging problem with a maximum of K pages in the fast memory, any deterministic online algorithm, as well as any memoryless randomized algorithm, is likely to produce a cost which is at least K times the cost of an optimal offline algorithm. McGeoch and Sleator (1991) proved that the expected cost of the randomized online algorithm which they developed, called the partitioning online algorithm, is within a factor of H_K of optimum (where H_K is the Kth harmonic number, which is approximately $\ln(K)$), and that no online algorithm can perform better than this measure. This last negative result is also applicable for our problem since we consider the more general case of non-uniform individual capacities; this indicates that the difference between the online and the offline versions of our problem may be very large.

We investigate the difference between the online and offline versions from the *average* point of view with respect to our test problems, and not from the worst case point of view. Our results indicate that a large gap indeed exists between the two versions, but the order of magnitude is much smaller. The theoretical results mentioned above imply that the gap we observe may be an unavoidable consequence of the online nature of the problem.

The paper is organized as follows: in section 2 we introduce the notation and discuss the problem assumptions; in section 3 we present two heuristic algorithms for stationary demand distributions of tools and test them numerically; in section 4 we develop a heuristic algorithm for non-stationary demand distribution of tools and present its performance; finally, in section 5 we draw our conclusions and discuss future research.

2. Notation

The problem is described by the following notation:

- T number of requests for tools that occur in the process; $(t = 1, \ldots, T)$,
- t index of time periods where each time period represents one request and in each time period, there is a requirement for one tool,
- N number of tools involved; $(i = 1, \ldots, N)$,
- v_i size of tool i (the number of slots it occupies in the tool magazine),
- K capacity of the tool magazine (the number of slots it contains),
- $d(t)$ identity of the tth tool required during the process; alternatively, when using the time periods terminology, $d(t)$ is interpreted as the identity of the tool which is required in time period t, or the demand in period t.

Here, we assume that $d(t)$ becomes known only in period t.

Whenever a tool is removed from or loaded on the tool magazine, a cost is incurred, which is proportional to the tool size, i.e. v_i for tool i. This is motivated by the fact that the cost is proportional to the time of removing/loading the tool which, by itself, is proportional to its size. Such a cost structure is appropriate in particular for tool kits, and serves as an approximation for large tools (see, however, our next assumption). We assume that the location of each tool in the tool magazine is not relevant, that is, every arrangement of a set of tools, which satisfy the capacity constraint, is feasible. This is applicable for the case of tool kits, where for big tools some time-consuming rearrangements may be needed.

The objective is to minimize the cost of removing and loading the tools subject to the capacity constraint of the tool magazine. Since the basic assumption of our model is that future demands are revealed only as they occur, the policy may use historical data only. Therefore, all decisions that are made prior to period t, are made without the knowledge of $d(t)$ or any later demand information. (Possible extensions may consider cases where there exists limited information on the future, that is, when there exists a time window of information, and no information is available beyond that.)

Indeed, the heuristics we employ base their decisions on past demands only. This approach has to be supported by additional assumptions with respect to the distribution of the requirements over time. We will demonstrate that an appropriate reaction to a specific assumption indeed improves the operation of the system, and suggest several possible procedures to control the system effectively.

3. Heuristics for stationary demand distributions

It can be shown that there exists an optimal solution in which at every time period a tool is inserted to the tool magazine only if it is required at that period, and tools are removed only if there is not enough capacity for the required tool. That is, tool switches are made only if it is necessary. This property is optimal for the static problem (when the requirements are known in advance), as well as for the dynamic problem discussed here. Therefore, all heuristics that we present in this paper adopt this rule of *no initiated removals*.

We use the following definitions:

S_t = the set of tools that are present on the tool magazine 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 as *the state of the system*.

$$B(t) = \sum_{i \in S_t} v_i = \text{the capacity usage (number of slots occupied) in period } t.$$

The value $B(t)$ keeps track of the used (and therefore implies the available) capacity in every period. It is updated every time that the heuristic changes the state of the system.

3.1. *The weighted backward distance (WBD) algorithm*

Matzliach and Tzur (1997) presented a heuristic algorithm (denoted there as heuristic 2), based on weighted distances of the future, that was shown to perform very well (producing solutions that are on average about 2% more costly than the optimal solution). We use here a similar idea, modified to incorporate the uncertainty in the demand.

The algorithm is iterative, that is, makes at every iteration (period) t a decision with respect to that iteration only. The algorithm is further based on the expectation that future demands will have a pattern that is similar to past demands; this is a basic assumption in most forecasting systems. Combining this expectation with the good performance of the above-mentioned weighted distances heuristic for the static case, we calculate here weighted backward distances (WBD) as a basis for the decision at each iteration of the heuristic regarding which tools to remove (when necessary).

For every period t, and every tool $i \in S_{t-1}$ s.t. $i \neq d(t)$, we define:

back_dist(i,t) = the distance (number of periods) from the last period when tool i was required until period t. In fact, back_dist(i,t) is the age of tool i in the system at time t.

(We assume that the magazine is initially empty; therefore if $i \in S_{t-1}$, it was undoubtedly required in the past, according to the principle of no initiated switches; if this assumption does not hold, the distance of $i \in S_{t-1}$ which was not required in the past should be set to t.) We refer to this quantity as the *backward distance* of tool i. A notational definition is as follows:

$$\text{back\_dist}(i,t) = m \text{ if } d(t-m) = i \text{ and } d(t-\tau) \neq i \text{ for } \tau = 1,\ldots,m-1$$

We are now ready to define back_w(J,t), the *backward weighted distance* of a *set* of tools J that are in the tool magazine in time $t-1$.

For every $J \subseteq S_{t-1}$ s.t. $d(t) \notin J$:

$$\text{back\_w}(J,t) = \frac{\text{average distance of tools in the set } J}{\text{sum of the sizes of tools in the set } J} = \frac{\sum_{i \in J} \text{back\_dist}(i,t) \Big/ |J|}{\sum_{i \in J} v_i}. \quad (1)$$

Our algorithm removes from the tool magazine the set of tools J that has the maximum backward weighted distance and whose removal frees enough capacity for inserting the required tool. The motivations leading to this criteria are:

(1) the further in the past a tool has been required, the further in the future it is expected to be required again, since the past gives us an indication on its usage rate;
(2) the larger the tool size is, the less attractive is its removal, because of its larger removal cost.

However, if every set $J \subseteq S_{t-1}$ will be examined, the algorithm will not run in polynomial time; therefore we choose a relatively small parameter L which denotes the maximum number of tools to be included in the set J. We denote by $Y_L^{S_{t-1}}$ all sets of up to L tools 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 $Y_L^{S_{t-1}}$ (the complexity increases with L) and by a feasibility constraint that needs to ensure that, in all cases, we will be able to find a set of up to L tools that free enough capacity. For the latter consideration, a choice of

$$L = \max_i (v_i) \Big/ \min_i (v_i)$$

for example will satisfy the constraint; more flexibility will be obtained by choosing larger L, on account of an increased complexity. (A more complicated procedure may choose L as a function of the needed capacity, but in this paper we choose L to be a constant.)

The algorithm proceeds as follows:

Step 1. For every iteration t, if $d(t) \in S_{t-1}$ then proceed to iteration $t+1$. Otherwise:

Step 2. If $B(t-1) + v_{d(t)} \leq K$ then insert tool $d(t)$ into the tool magazine and proceed to iteration $t+1$. Otherwise:

Step 3. For all $i \in S_{t-1}$ evaluate back_dist(i,t) and for every $J \in Y_L^{S_{t-1}}$ which satisfies $B(t-1) + v_{d(t)} - \sum_{i \in J} v_i \leq K$, calculate back_w$(J,t)$ according to (1).

Step 4. Remove the tools in the set J, which achieves the maximum back_w(J,t). (Tie-break arbitrarily.) Insert tool $d(t)$.

In the above description we have omitted the technical and trivial details of initializing the algorithm and updating the state of the system at every iteration.

If we apply our weighted backward distance algorithm to the special case where the tool sizes are uniform, we are back to the LRU policy applied in the paging problem, as discussed in the introduction.

3.2. *The weighted probabilistic (WP) algorithm*

The essential difference between the weighted probabilistic (WP) algorithm of this section and the weighted backward distance (WBD) algorithm presented in the previous section is in the method of using past data. While the WBD algorithm looks back to the last time a tool was required, assuming that the expected distance until the next demand is identical, the WP algorithm accumulates information on *all* the times in the past in which the tool was required. Based on this information, the probability that a given tool will reappear is calculated at every iteration t and the expected distance until its next demand is calculated as the reciprocal of this probability. The latter calculation is appropriate when the demand distribution is geometric, and we find it useful to use when no other information on the distribution type is available.

For every tool i and time t define:

$$\text{prob}(i,t) = \frac{n(i,t)}{t}$$

where $n(i,t)$ is the number of periods (up to period t) in which the requested tool was i. Updating the probabilities from period $t-1$ to period t is performed as follows, for all i:

$$\text{prob}(i,t) = \begin{cases} \dfrac{\text{prob}(i,t-1)\cdot(t-1)+1}{t} & d(t) = i \\ \dfrac{\text{prob}(i,t-1)\cdot(t-1)}{t} & \text{otherwise} \end{cases}$$

Now we estimate in period t the expected distance to the next demand of tool i as:

$$P\_\text{dist}(i,t) = 1/\text{prob}(i,t)$$

based on which the expected *weighted* distance of a set of tools is calculated (denoted by $P\_w(J,t)$), exactly as in the WBD algorithm:

$$P\_w(J,t) = \frac{\sum_{i \in J} P\_\text{dist}(i,t)/|J|}{\sum_{i \in J} v_i}$$

and the tools in the set $J \in Y_L^{S_{t-1}}$ which satisfies $B(t-1) + v_{d(t)} - \sum_{i \in J} v_i \leq K$ and has the maximum value of $P\_w(J,t)$ are removed.

The advantage of the WP algorithm over the WBD algorithm is indeed in utilizing more information, and learning from the past in order to get a better estimate of the expected distance to a future demand. On the other hand, the WP algorithm

needs to keep information about *all* tools, including those that are presently not in the tool magazine. For a typical process this may not be a problem since the total number of tools is larger by only a constant factor than the average number of tools that can simultaneously be present on the tool magazine. However, in other applications of this problem this fact may cause large memory requirements. For example, in the related paging problem mentioned in the introduction, the number of pages involved in a process is much larger than the capacity of the fast memory and this is the reason for not using such an algorithm. Compared to that, in the WBD algorithm, only information with respect to the tools that are in the tool magazine is kept; once a tool leaves the tool magazine the information with respect to it is forgotten.

If we apply the WP algorithm to the special case where the tool sizes are uniform, then we choose to remove in every period t the tool with the smallest probability (estimated thus far) of reappearing.

3.3. *Computational experiments*

We note that the cost of an optimal solution of the static problem (when all requirements are known in advance) is a lower bound for the cost of the dynamic problem discussed in this paper. In general, the cost of an optimal solution of the static problem is not achievable in a dynamic setting.

In this empirical study we compare the two dynamic algorithms to a heuristic algorithm for the static problem, the heuristic denoted as heuristic 2 in Matzliach and Tzur (1997), which is also mentioned in the previous section. The reason that we use a heuristic for the static problem is that obtaining the optimal solution is very time consuming (the problem is NP-Complete and it took us several hours to get an optimal solution to one instance of the problem by solving its integer programming formulation); the heuristic's value comes extremely close ($< 2\%$ average) to the optimal value.

In addition, to evaluate the effectiveness of the heuristics, we compare their values to a simple policy, the *random solution*, obtained as follows: removing a tool is performed only when the required tool is not in the tool magazine, and not enough capacity is available for it. In that case, one of the tools in the tool magazine is chosen randomly (with an equal probability for each tool), and removed. If this does not free enough capacity for the required tool, another tool will be removed in the same way until the required tool can be loaded on the tool magazine. We also specify the value of a *naive solution* whose rule is to load every tool on the tool magazine when it is required, and to remove it immediately thereafter. The naive solution is not intended as an alternative or reasonable policy, but rather as a benchmark representing the worst possible case.

The parameters for the test problems were generated as follows: for each tool, a geometric distribution was assumed, indicating whether the tool is required in each time period. The probability parameter of the distribution was generated from a uniform distribution between 0 and 1 (later normalized so that the sum of the probabilities of all tools is one). The size of each tool was generated from a discrete uniform distribution in the range (5, 15). We generated 10 instances for every problem set where, in every instance, a new realization of the requirements as well as a new realization for the tool sizes is generated. The dynamic heuristics (WBD and WP) were executed with $L = 3$.

In problem set 1 we used $N = 25$ (number of tools), $T = 200$ (number of requirements) and a varying value for the capacity K. The resulting cost values are presented

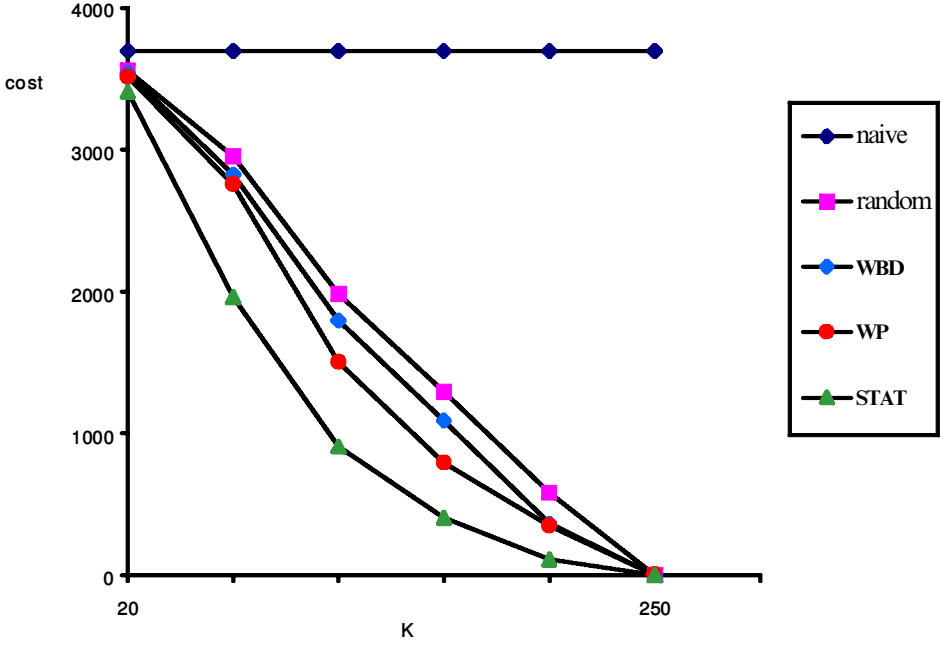

Figure 1. Problem set 1: total cost of the various solutions as a function of K.

K	$K=20$	$K=50$	$K=100$	$K=150$	$K=200$	$K=2500$
Naive solution	3965 (81)	3695 (81)	3695 (81)	3695 (81)	3695 (81)	3695 (81)
Random solution	3555 (163)	2952 (141)	(1980 (134)	1294 (180)	(582 (70)	0
WBD heuristic	3539 (169)	2825 (104)	1795 (140)	1088 (170)	364 (61)	0
WP heuristic	3511 (161)	2752 (129)	1502 (133)	791 (190)	347 (42)	0
Static problem (heuristic)	3407 (161)	1961 (65)	904 (90)	403 (100)	111 (25)	0

Remark: the costs are averages of 10 instances, the standard deviation is in parenthesis.

Table 1. Problem set 1: total cost values of the various solutions.

in table 1 and figure 1, as a function of the parameter K. By definition of the naive solution, its cost is constant as a function of K while the other heuristics' cost values decrease as a function of K. We note that the relative order of all heuristics' cost values remains unchanged as a function of K. For $K=250$, all items could fit together in the tool magazine, therefore no cost was incurred.

The average deviations (over the values $50 \leq K \leq 200$) of the WBD and WP heuristics from the cost of the (close to optimal) static solution were 135% and 103% respectively, while the average deviation of the random solution was 203%.

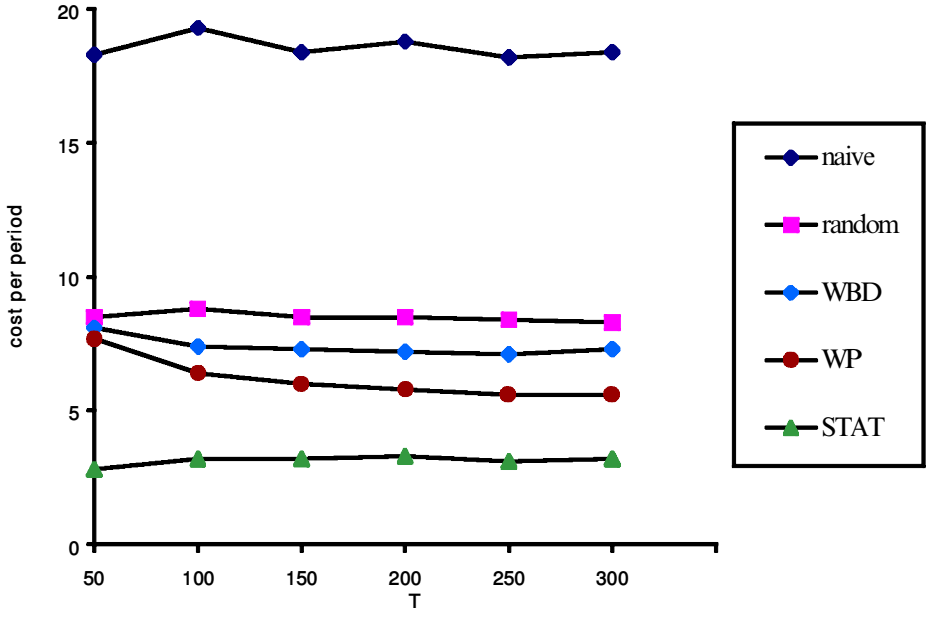

Figure 2. Problem set 2: cost values per period as a function of T.

T	$T = 50$	$T = 100$	$T = 150$	$T = 200$	$T = 250$	$T = 300$
Naive solution	18.3	19.3	18.4	18.8	18.2	18.4
Random solution	8.5	8.8	8.5	8.5	8.4	8.3
WBD heuristic	8.1	7.4	7.3	7.2	7.1	7.3
WP heuristic	7.7	6.4	6	5.8	5.6	5.6
Static problem (heuristic)	2.8	3.2	3.2	3.3	3.1	3.2

Table 2. Problem set 2: cost values per period (average of 10 instances).

The maximum deviations were 227%, 212% and 424%, respectively, for the three solutions. The best heuristic appears to be WP, which is not surprising given that it uses more information about past demands than the other heuristics. Both the WP and WBD heuristics demonstrate a big improvement over the random solution. However, note that the deviation from the static solution is still significant, around 100% on average for WP.

In problem set 2 we focus on the behaviour of the heuristics as a function of T, the number of periods/requirements. We used $N = 25$ and $K = 125$ throughout, and T varied from 50 to 300 in intervals of 50. The resulting cost values per time period are presented in table 2 and figure 2, as a function of T. In this case, while the value of the naive solution varies according to the randomness of the tool sizes, the other heuristic values are less variable as a function of T. Among those, we observe that the value of the WP heuristic is decreasing with T, and this result is due to the learning that occurs over time with respect to the probabilities of the tools being required. The relative order of all heuristic values remains unchanged as a function of T as well.

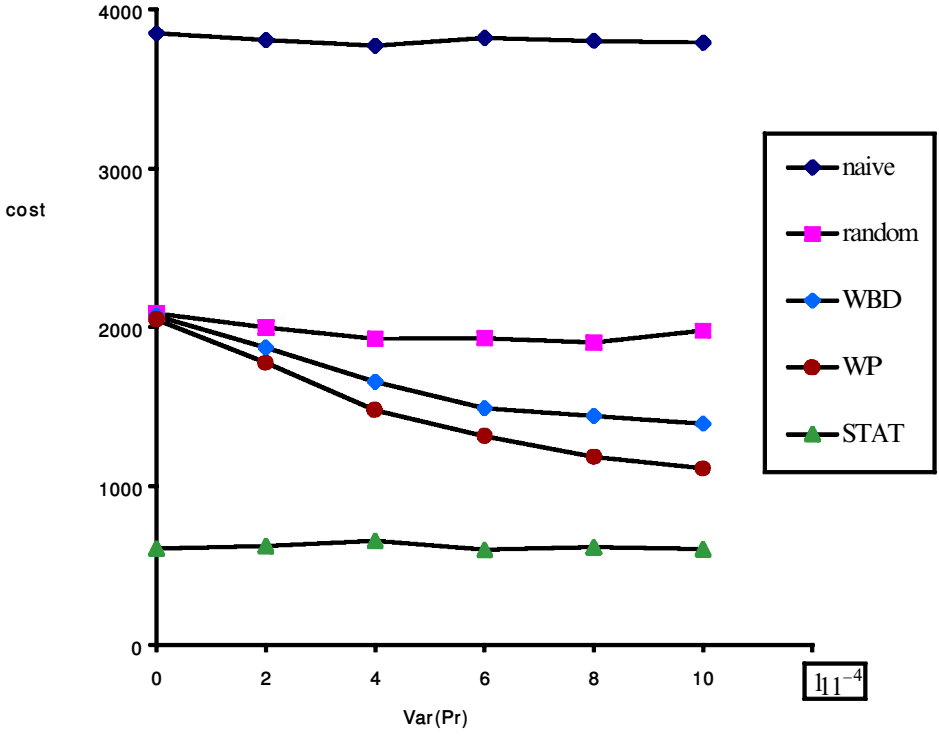

Figure 3. Problem set 3: total cost as a function of the demand variability between items.

Finally, in problem set 3 we analyse the cost of the heuristics as a function of the demand's variability among the tools. We denote by Var (Pr) the variance of the probability distribution according to which the requirements are generated, and run all algorithms for variance values of 0, 2×10^{-4}, 4×10^{-4}, 6×10^{-4}, 8×10^{-4} and 10×10^{-4}. The other parameters are: $N = 25$, $K = 125$ and $T = 200$. The results (depicted in figure 3) are quite interesting. As the variance increases, algorithms WBD and WP are improving and getting closer to the value of the static problem, while the other algorithms are relatively constant over the variance range. We explain it as follows: with a large variance among the tools, some tools are much more likely to be required than others. These two algorithms apply past experience, recognize the 'popular' tools and try to keep them in the tool magazine. The WP algorithm accumulates more information about the past than the WBD algorithm and therefore its advantage is once again very clear. The random algorithm does not make use of past data, and therefore is not affected much by variance changes. In the extreme case where Var (Pr) = 0, the probability distribution of the demands for the tools is uniform, in which case the WBD and WP algorithms have no advantage over the random algorithm.

4. Heuristic for non-stationary demand distributions

The underlying assumption in the design of the heuristics of the previous section was that the demand distribution of the requested tools does not change over time. In particular, this is the reason why, in the WP heuristic, information was accumulated over the entire history. If the demand distribution changes over time, the entire

history is a misleading estimator, and the performance of the WP heuristic would deteriorate.

For those processes that are characterized by (possible) changing activities and therefore by changing demand probabilities, we suggest in this section a heuristic which gives a large weight to recent information, and a smaller weight to older information. We do this by using an exponential smoothing approach with respect to the demand probabilities, combined with the WP heuristic. More specifically, we update each tool's demand probability by averaging a new estimate for it with a factor of α $(0 \leq \alpha \leq 1)$ and its old estimate with a factor of $1 - \alpha$.

Our new estimate is based on a duration of several periods, denoted by *length*. At the completion of each *length* we update the demand probabilities. We use the following definitions:

τ = the running index for *length*
Prob (i, t, τ) = the probability of demand for tool i in period t of *length* τ, calculated at the end of *length* $\tau - 1$ for $\tau > 1$, and at the end of period $t - 1$ for $\tau = 1$.
num (i, τ) = the number of demands for tool i in *length* τ.

According to the definition above, we have the following formula:

$$\text{Prob}(i,t,\tau) = \begin{cases} \dfrac{n(i,t-1)}{t-1} & \tau = 1 \\ (1-\alpha)\,\text{Prob}(i, \textit{length}, \tau - 1) + \alpha \cdot \dfrac{\text{num}(i,\tau-1)}{\textit{length}} & \text{otherwise} \end{cases}$$

In the first *length*, the probability is updated every single period, since no prior information is available; afterwards, it is updated at the end of every *length*. The rest of the algorithm is identical to that of WP and we denote it as the *exponential algorithm* (EXP). Finally, the parameters *length* and α have to be determined. The value of *length* has to be long enough (multiples of N) so that enough information can be gathered, but not too long, so that the system is updated in real time. The value of α needs to be determined as in every exponential smoothing application: if we believe that there is a rapid change in the demand probabilities then a large α needs to be chosen so that more weight is given to the new information. On the other hand if we think that the system is relatively stable then we better choose a relatively small α so that our estimate will not get biased by the last *length* realization.

In the next test problem set, denoted as set 4, we study the performance of the EXP algorithm compared with the previous algorithms, as well as the effects of the parameters *length* and α on it. The overall duration of this problem was 800 periods, but every 200 periods, the probability distributions of the 5 tools were changed, according to the vectors shown below. These probabilities were used to generate the requirements, but obviously were not known during the execution of the EXP algorithm. For simplicity we used here equal tool sizes of 1, $K = 3$ and $N = 5$. The value of *length* was chosen to be $2N$, that is: 10.

$$\text{Pr}(i,t) = \begin{cases} [0.5, 0.2, 0.1, 0.1, 0.1] & 0 < t \leq 200 \\ [0.1, 0.1, 0.1, 0.2, 0.5] & 200 < t \leq 400 \\ [0.1, 0.5, 0.2, 0.1, 0.1] & 400 < t \leq 600 \\ [0.2, 0.1, 0.1, 0.5, 0.1] & 600 < t \leq 800 \end{cases}$$

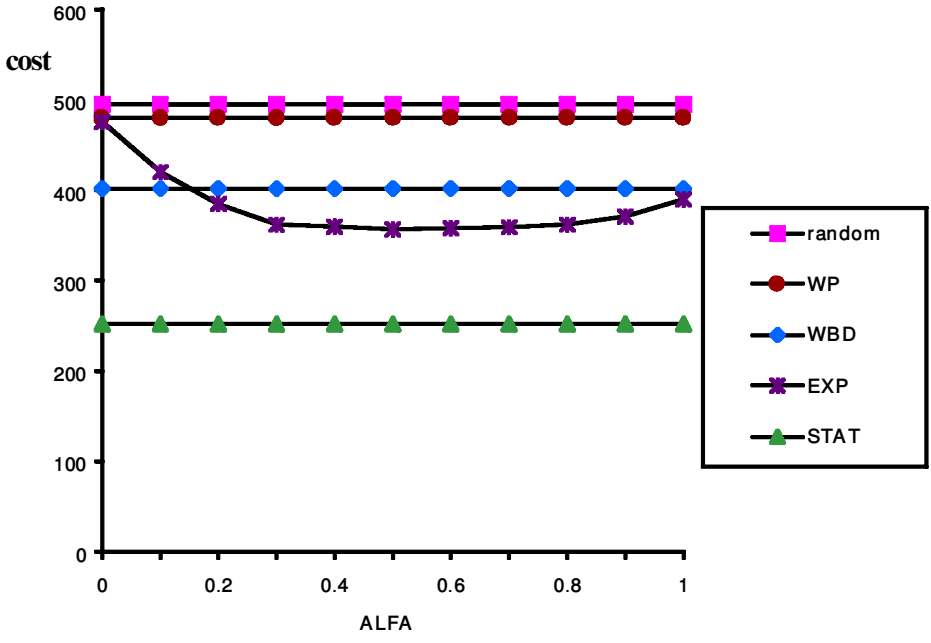

Figure 4. Problem set 4: cost values as a function of α.

Algorithm	Total cost
Static problem (heuristic)	252
Random solution	495
PBD heuristic	402
WP heuristic	480

Table 3. Problem set 4: total costs with changing demand distributions.

α = 0	α = 0.1	α = 0.2	α = 0.3	α = 0.4	α = 0.5	α = 0.6	α = 0.7	α = 0.8	α = 0.9	α = 1
476	420	385	362	360	357	358	359	362	371	390

Table 4. Problem set 4: cost of the EXP heuristic as a function of α.

N	2N	3N	4N	5N	6N	7N	8N	9N	10N
370	357	358	360	369	389	402	425	450	470

Table 5. Problem set 4: cost of the EXP heuristic as a value of *length*.

The resulting costs of all algorithms are shown in tables 3 and 4 where, in table 4, the result of the EXP algorithm is shown for various values of α. These results are depicted graphically in figure 4. Finally, in table 5 and figure 5 the dependency of the cost of algorithm EXP on the parameter *length* is presented; there, a value of α = 0.5 was chosen.

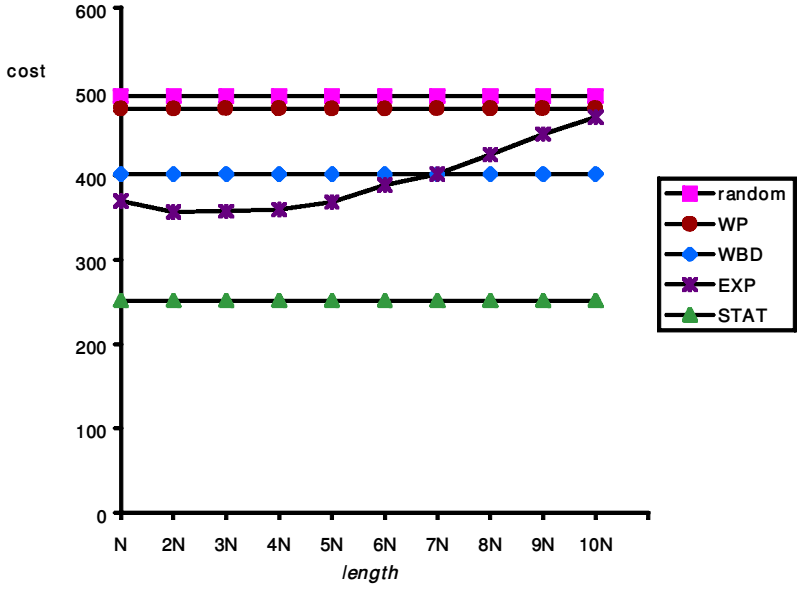

Figure 5. Problem set 4: cost values as a function of *length*.

We first note from these results that when the demand distribution is non-stationary, algorithm WBD performs better than algorithm WP, as opposed to the case where the demand distribution is stationary. This results from the fact that the WP algorithm remembers all past information with equal weights, therefore not estimating correctly the most recent (and relevant) distribution. On the other hand, the WBD algorithm looks back only until the most recent request which (except for the transition times) is drawn from the recent and relevant distribution.

Better than these two algorithms is the EXP algorithm, for almost all values of α and *length*. For large values of *length* ($> 7N$) the EXP algorithm becomes inferior to WBD, since its updating procedure becomes too slow. Overall, the enhancement of the EXP algorithm is rewarding.

5. Conclusions and future research

We presented in this paper the first analysis of the dynamic tool switching problem with non-uniform tool sizes. We developed several heuristic rules to control the switching strategy, differing from each other in their underlying assumptions with respect to the process randomness. Having no prior rules to compare our heuristics to, we compared them to a random policy and demonstrated their superiority.

Several related issues remain unanswered and may be part of future research. One such issue is developing an efficient strategy for an environment where a forecast window exists for a limited number of periods in the future in which demand is known, and beyond that demand is unknown. Another interesting related question is the determination of the best storage size, that is, the capacity of the tool magazine. There is a clear trade-off between a large tool magazine which allows for lower

operating costs (tool switches), and a smaller tool magazine that is less costly to purchase but more expensive to operate.

References

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

Belady, L. A., 1966, A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, **5**, 78–101.

Coffman, E. G. and Denning, J. P., 1973, *Operating Systems Theory* (Englewood Cliffs, NJ: Prentice Hall).

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.

Irani, S. and Karlin, A. R., 1997, On-line computation. In *Approximation Algorithms for NP-hard Problems*, edited by D. S. Hochbaum (PWS Publishing Company).

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

Matzliach, B. and Tzur, M., 1997, Storage management of items in two levels of availability. Submitted for publication to *European Journal of Operational Research*.

McGeoch, L. A. and Sleator, D. D., 1991, A strongly competitive randomized paging algorithm. *Algorithmica*, **6**, 816–825.

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

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

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

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