Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 157 (2004) 784–790
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Short Communication

Scheduling with tool changes to minimize total completion time: Basic results and SPT performance

M. Selim Akturk [a,*], Jay B. Ghosh [b], Evrim D. Gunes [c]

[a] Department of Industrial Engineering, Bilkent University, Ankara 06800, Turkey
[b] Department of Information and Operations Management, University of Southern California, Los Angeles, CA, USA
[c] Production and Operations Management, INSEAD, Fontainebleau, France

Received 18 April 2002; accepted 12 March 2003
Available online 1 August 2003

Abstract

We consider a single machine sequencing problem subject to tool wear, where the objective is to minimize the total completion time. We briefly describe the problem and discuss its properties, complexity and solution. Mainly, however, we focus on the performance of the SPT list-scheduling heuristic. We provide theoretical worst-case bounds on SPT performance and also demonstrate its empirical behavior.

Keywords: Scheduling; Tool change; SPT list-scheduling; Performance analysis

1. Introduction

The traditional literature on machine scheduling generally assumes that a machine is able to process jobs continuously at all times. In practice, however, that is rarely the case; machine operation is often disrupted by random breakdown, preventive maintenance or tool change necessitated by job mix or tool wear. Only recently has this issue begun to get the attention that it deserves. For example, see Adiri et al. [1] and Albers and Schmidt [5] for research on scheduling subject to random machine breakdown, Lee [12] for scheduling research in presence of preventive maintenance, and Schmidt [15] for a review on machine availability. Scheduling subject to tool change, while it is more common and frequent, has not been addressed thus far.

There is a growing body of literature on tool management that considers tool change explicitly; Crama [7] provides an overview. This is perhaps due to the late recognition that lack of tooling considerations has led to the considerable under performance of automated manufacturing systems; see Gray et al. [9]. However, its origins being in the work on flexible machines, this literature has focused mainly on tool change induced by job mix rather than that due to tool wear; Akturk and Avci [2] is a notable exception. This is in sharp contrast to the finding in Gray et al. [9] that in real life tool change due to tool wear is approximately 10 times

* Corresponding author. Tel.: +90-312-290-1360; fax: +90-312-266-4054.
E-mail address: akturk@bilkent.edu.tr (M.S. Akturk).

0377-2217/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0377-2217(03)00232-7

more frequent than that induced by job mix. Furthermore, tool management research does not address the scheduling-related performance measures such as total completion time; the emphasis typically is on minimizing the number of tool changes. The models are mostly motivated by past industrial experience that the time needed for changing tools overwhelmingly dominates the job processing times; see Tang and Denardo [16].

In this note, we attempt to redress the above situation by considering, as a first step, a prototypical single-machine scheduling problem that accounts for tool change due to tool wear. The chosen objective is to assign a given set of jobs successively to identical tools from an available pool and sequence them on the tools such that the total job completion time is minimized. Extending the standard scheduling notation, the problem at hand can be called $1|\text{tool-change}|\sum_j C_j$. We admit that our adoption of a single tool type makes the problem a bit restrictive; it nonetheless retains a practical basis (as we will discuss later).

We wish to point out at this stage that our work is similar in spirit to that on preventive maintenance [12,13]. The most significant differences with the early work are that we allow multiple tool changes (machine unavailability periods) as opposed to just a single one and that we do not allow partial processing of a job (resume policy) or any machine idle time other than that forced by tool change (restart policy). Recently, Qi et al. [14] has addressed a maintenance problem that is in fact mathematically equivalent to our tool change problem and presented results that are similar to what we have obtained independently [3]. We should point out that, in their context the maintenance interval is variable and thus that their model may not apply to scheduled maintenance (which is performed by specialist crews at fixed intervals); it may be more appropriate for routine maintenance performed by the machine operator (such as adjustment, lubrication and cleaning) within a specified period. More recently, Graves and Lee [8] has addressed a generalization of the Qi et al. model and, in that sense, has a bearing on our work as well.

In the sequel, we first introduce the problem and its two versions. We then briefly state several structural properties for an optimal schedule, precisely establish the complexity of the problem and propose a dynamic program for its exact solution. As our main contribution, we study the performance of the shortest processing time first (SPT) list-scheduling heuristic next. We show that SPT is optimal if the tool-change time is negligible or if the number of tools it needs is two or less. We further show that SPT has a worst-case performance bound of 1.5 (which is tight) if it uses three tools and 2.0 if it uses more. Finally, we report a limited computational study that shows that SPT performs quite well in practice.

2. Problem definition and solution properties

We are given a single machine that will remain continuously operational from time zero except when there is a tool change in progress. There are also n independent jobs that are ready for processing at that time. The job processing time (p_j for job j, $j = 1, \ldots, n$) is known and constant. Also, m identical units of a single tool type with known, constant life (T_L, $T_L \geqslant p_j$ for $j = 1, \ldots, n$) are available. When an active tool is due to wear out, no new job is assigned to it and it is replaced with a new tool; the time needed for this tool change (T_C) is also known and constant. The processing of a job is never interrupted because of tool change or otherwise. Finally, all numbers are assumed to be integers.

A few comments are in order here. First, the use of a single tool type is not entirely uncommon. Computer numerically controlled (CNC) drilling machines can use identical drill bits to process a number of jobs in succession. Second, non-preemption of jobs is also reasonable in many machining situations because preemption affects surface finish and results in non-machining activities such as job removal, placement and orientation. Last, tools used by CNC machines are quite expensive; see Kouvelis [10] and Tomek [17]. It is not uncommon at all to see that tool cribs maintain only a small number of such tools on hand. The challenge often is thus to determine how to schedule a given set of jobs with only a limited number of tools (a situation that we will encounter shortly).

Going back to the problem, we note that we wish to find a feasible schedule s^* that will minimize the total completion time of the jobs. It is easily seen that there is no advantage to have any machine idle time other than what is forced by a tool change. Nor is there any advantage to change a tool when it can process the job that is next in sequence. We assume that these policies are being enforced. A job sequence σ then translates uniquely to a job-to-tool assignment using $m(\sigma)$ tools. If $m \geqslant m(\sigma)$, σ is feasible and yields the associated schedule s; the sequence is infeasible otherwise. In the remainder of the paper, we use the terms schedule and sequence interchangeably.

We now introduce two versions of $1|\text{tool-change}|\sum_j C_j$. Ignore m and assume that σ^* is an optimal sequence that uses the least number of tools $m(\sigma^*)$ among all optimal sequences. If $m \geqslant m(\sigma^*)$, tool availability does not limit the scheduling process from realizing the minimum total completion time. So, whenever we have an assuredly large m, we say that we have the unlimited tools version of our problem; the objective here is to find σ^* regardless of m. Of course, we do not know a priori if a problem instance is such, short of finding σ^*, unless m is very large (as when $m \geqslant n$). Anyhow, it is worthwhile to note that there is no need to specify m when addressing the unlimited tools version, that σ^* always exists in this case and that for computational purposes an upper bound on $m(\sigma^*)$ can be used in lieu of m. One may also note that it is this very version that coincides with the work of Qi et al. [14].

If m is not assuredly large, tool availability may impact the scheduling process and we say that we have the *limited tools* version of our problem. We note that the specification of m is integral to this version and further that a feasible sequence may not exist in this case. Disregarding feasibility for the moment, let BIN be a sequence that uses the least number of tools among all sequences; we can generate BIN from the exact solution of a bin-packing problem. If $m < m(\text{BIN})$, no feasible sequence can be found. If $m(\text{BIN}) \leqslant m < m(\sigma^*)$, only a constrained optimal sequence $\sigma^{\wedge}$ can be found such that $Z(\sigma^{\wedge}) > Z(\sigma^*)$. Again, we do not know a priori what kind of a situation we are dealing with, short of solving the bin-packing problem, unless m is very small (as when $m < \sum_j p_j/T_L$).

The upshot is that we will have one of two versions of $1|\text{tool-change}|\sum_j C_j$ to deal with: the *unlimited tools* version where m is not explicitly considered, and the *limited tools* version where m is considered as such. We proceed to describe certain developments that apply to both versions of the problem.

For a given sequence σ using $m(\sigma)$ tools, let C_j be the completion time of job j, $p_{[k]}$ be the processing time of the job in position k of the sequence, t_i be the total processing time of all jobs assigned to tool i, η_i be the number of jobs assigned to tool i, Z (equal to $\sum_{1 \leqslant j \leqslant n} C_j$) be the total completion time. It can then be shown that $Z = Z_S + Z_T$, where $Z_S = \sum_{1 \leqslant k \leqslant n} (n-k+1)p_{[k]}$ and $Z_T = [\sum_{1 \leqslant i \leqslant m(\sigma)} (i-1)\eta_i]T_C$. We recognize that Z_S is the total completion time of the jobs in σ when there is no idle time due to tool change (that is when $T_C = 0$). Similarly, Z_T is the adverse effect of a non-zero T_C. Clearly, as $T_C \to 0$ and $T_C \to \infty$, Z_S and Z_T, respectively, dominate in the minimization of Z. Beyond these insights, the above characterization is also helpful in analyzing the problem and its solution (as we will see later).

3. Properties, complexity and solution

We start by stating a number of structural properties that hold for an optimal sequence; they have been arrived at independently by Qi et al. [14] and us [3]. From this point on, we will assume (without loss of generality) that the jobs are indexed in the SPT order and the tools are numbered in order of their use.

Property 1 (SPT within tool). *The jobs assigned to the same tool are sequenced in the SPT order.*

Property 2 (Tool utilization). $T_L - t_i < p_j$, *for any tool* i *and any job* j *assigned to tools* $i+1, \ldots, m(\sigma)$.

Property 3 (Average job time). $(t_{i_1} + T_C)/\eta_{i_1} \leqslant (t_{i_2} + T_C)/\eta_{i_2}$ *for any tools* i_1 *and* i_2 *such that* $i_1 < i_2$.

Property 4 (Job loading). *$\eta_{i_1} \geqslant \eta_{i_2}$ for any tools i_1 and i_2 such that $i_1 < i_2$.*

We now establish the complexities for the *limited tools* and the *unlimited tools* versions of $1|\text{tool-change}|\sum_j C_j$ and give a dynamic program for their solution. Note that Result 1 is straightforward and that Result 2 has been obtained independently by Qi et al. [14] and us [3].

Result 1 (Limited tools). *The limited tools version of $1|\text{tool-change}|\sum_j C_j$ is NP-hard in the ordinary sense if m is fixed and in the strong sense if m is arbitrary, even when $T_C = 0$.*

Result 2 (Unlimited tools). *The unlimited tools version of $1|\text{tool-change}|\sum_j C_j$ is NP-hard in the strong sense.*

Lastly, we describe a Lawler–Moore [11] type dynamic program (DP) for the $1|\text{tool-change}|\sum_j C_j$ problem. It helps us establish the precise complexity of the *limited tools* version when m is fixed; it is also practically viable as long as the problem parameters remain agreeably small. The DP algorithm becomes impractical at $n = 16$ and $m = 4$ because of its huge storage requirement; at this size, it also takes more than an hour of CPU time.

For the *limited tools* version, we use m as it is. For the *unlimited tools* version, we use an upper bound on $m(\sigma^*)$; we may alternatively use $m(\text{SPT})$ as a practical surrogate (in view of the high tool costs). We assume that the jobs are scheduled one at a time starting with job 1; at any stage r, job r is thus scheduled on a tool with an index between 1 and m (if it is possible to do so). Let t_i be the total processing time and η_i be the number of jobs assigned to tool i, and $f_r(t_1,\ldots,t_m;\eta_1,\ldots,\eta_m)$ be the minimum total completion time realizable at stage r for a given state $(t_1,\ldots,t_m;\eta_1,\ldots,\eta_m)$. Clearly, $0 \leqslant t_i \leqslant T_L$, $\sum_{1\leqslant i\leqslant m} t_i = \sum_{1\leqslant j\leqslant r} p_j$, $0 \leqslant \eta_i \leqslant r$ and $\sum_{1\leqslant i\leqslant m} \eta_i = r$. The dynamic programming recursion is given by

$$f_0(t_1,\ldots,t_m;\eta_1,\ldots,\eta_m) = 0 \quad \text{for all } t_i \text{ and } \eta_i = 0;$$
$$= \infty \quad \text{otherwise.}$$

$$f_r(t_1,\ldots,t_m;\eta_1,\ldots,\eta_m)$$
$$= \min_{1\leqslant i\leqslant m}\left[f_{r-1}(t_1,\ldots,t_i-p_r,\ldots,t_m;\eta_1,\ldots,\eta_i-1,\ldots,\eta_m) + \sum_{1\leqslant q\leqslant i} t_q + (i-1)T_C + \sum_{i+1\leqslant q\leqslant m} \eta_q p_r \right]$$
$$\text{for all feasible states;}$$
$$= \infty \text{ otherwise.}$$

The optimal solution value is given by the minimum of $f_n(t_1,\ldots,t_m;\eta_1,\ldots,\eta_m)$ over all feasible states at stage n (with a value of ∞ indicating that there is no feasible sequence) and an optimal sequence (if it exists) is constructed through backtracking. The overall complexity of DP is $O(mn^{m+1}T_L^m)$, which is pseudo-polynomial for a fixed m.

DP enumerates over a minimal representative set of all non-dominated partial schedules that upon completion will potentially lead to an optimal sequence; it is thus correct. Its state space at any stage is bounded by $[\Pi_{1\leqslant i\leqslant m}(ti)(\eta_i)] \leqslant T_L^m n^m$, and m computations occur at each state. Over n stages, this translates to time and space requirements loosely bounded by $mn^{m+1}T_L^m$.

4. SPT performance

We now look at the SPT list-scheduling heuristic as an approximate solution for $1|\text{tool-change}|\sum_j C_j$. Certainly, this exercise is meaningful only when we are considering the *unlimited tools* version of the problem or when $m \geqslant m(\text{SPT})$. We start with the following result.

Result 3 (SPT optimality). *The SPT sequence is optimal if $T_C = 0$ or $m(\text{SPT}) \leqslant 2$.*

The first part is obvious from past observations. The second part is also easy to see as SPT minimizes both Z_S and Z_T.

If $m(\text{SPT}) > 2$, the SPT sequence provides only a heuristic solution to our problem. Let OPT represent the corresponding optimal sequence. Now define the performance ratio ρ of the SPT sequence as follows:

$$\rho = \min_I \{Z(\mathrm{SPT}(I))/Z(\mathrm{OPT}(I))\},$$

where I represents a problem instance. (Henceforth, we drop I from the description, whenever there is no scope for confusion.) Since $Z_S(\mathrm{SPT}) \leqslant Z_S(\mathrm{OPT})$ and $Z_T(\mathrm{SPT}) \geqslant Z_T(\mathrm{OPT})$, it follows that $\rho \leqslant Z_T(\mathrm{SPT})/Z_T(\mathrm{OPT})$. This in turn implies that

$$\rho \leqslant \left[\sum_{1\leqslant i\leqslant m(\mathrm{SPT})} (i-1)\eta_i(\mathrm{SPT})\right] \Big/ \left[\sum_{1\leqslant i\leqslant m(\mathrm{OPT})} (i-1)\eta_i \times(\mathrm{OPT})\right].$$

The following result provides an upper bound on ρ when $m(\mathrm{SPT}) = 3$.

Result 4 (SPT performance ratio—special)**.** *If* $m(\mathrm{SPT}) = 3$, *then* $\rho \leqslant 1.5$; *and this bound becomes tight as* $T_C \to \infty$.

Clearly, $m(\mathrm{OPT}) \geqslant m(\mathrm{BIN}) \geqslant 2$. We get an upper bound on $[\sum_{1\leqslant i\leqslant m(\mathrm{SPT})} (i-1)\eta_i(\mathrm{SPT})]$ if we use $\eta_i(\mathrm{SPT}) = [n - \eta_1(\mathrm{SPT})]/2$ for $i = 2, 3$, and a lower bound on $[\sum_{1\leqslant i\leqslant m(\mathrm{OPT})} (i-1)\eta_i(\mathrm{OPT})]$ if we use $\eta_1(\mathrm{OPT}) = \eta_1(\mathrm{SPT})$, $\eta_2(\mathrm{OPT}) = n - \eta_1(\mathrm{SPT})$ and $\eta_i(\mathrm{OPT}) = 0$ for all other i. Substituting these in the right hand side of the inequality on ρ immediately yields the first part of the result.

To see that the bound on ρ becomes tight as $T_C \to \infty$, consider the following instance: $n = 5$, $m = 3$, $T_L = 6$, $\{p_j : j = 1, \ldots, 5\} = \{1, 2, 2, 3, 4\}$. The SPT sequence uses all three tools and has jobs 1–3 on tool 1, job 4 on tool 2 and job 5 on tool 3. The OPT sequence uses two tools and has jobs 1, 2, 4 on tool 1 and jobs 3, 5 on tool 2. On evaluation, it is seen that

$$Z(\mathrm{SPT})/Z(\mathrm{OPT}) = (29 + 3T_C)/(30 + 2T_C).$$

As $T_C \to \infty$, $Z(\mathrm{SPT})/Z(\mathrm{OPT}) \to 1.5$.

We now state a more general result on ρ. This is also the main new result of this note.

Result 5 (SPT performance ratio—general). *If* $m(\mathrm{SPT}) \geqslant 3$, *then* $\rho \leqslant 2$.

The proof is by induction. In what follows, SPT_j, OPT_j and SEQ_j represent, respectively, the SPT sequence, an optimal sequence and any arbitrary sequence of jobs 1 through j. Now, let q be the maximum number of jobs that a SPT sequence can assign on two tools. For jobs 1 through j, $1 < j \leqslant q$, the optimal sequence is known to be SPT_j (Result 3). Let r be the maximum number of jobs that the SPT sequence can assign on three tools. For jobs 1 through j, $q < j \leqslant r$, SPT_j may not be optimal. However, it follows from the proof of Result 4 that $Z_T(\mathrm{SPT}_j)/Z_T(\mathrm{OPT}_j) \leqslant 1.5 < 2$.

We now hypothesize that $Z_T(\mathrm{SPT}_j)/Z_T(\mathrm{OPT}_j) \leqslant 2$ for some j, $j > q$. It is clear that we are dealing with $m(\mathrm{SPT}_j) \geqslant 3$ and that any sequence will need two or more tools for jobs 1 through j. We will prove that $Z_T(\mathrm{SPT}_{j+1})/Z_T(\mathrm{OPT}_{j+1}) \leqslant 2$.

If $m(\mathrm{BIN}_{j+1})$ is the number of tools used for jobs 1 through j in an optimal bin-packing solution, it is known from Anily et al. [6] that $m(\mathrm{SPT})/m(\mathrm{BIN}) \leqslant 1.75$. Since $m(\mathrm{OPT}_{j+1}) \geqslant m(\mathrm{BIN}_{j+1})$ and $m(\mathrm{SPT}_{j+1})$ is integral, $m(\mathrm{SPT}_{j+1}) \leqslant \lfloor 1.75m(\mathrm{OPT}_{j+1})\rfloor$. It is easy to see

$$Z_T(\mathrm{SPT}_{j+1}) = Z_T(\mathrm{SPT}_j) + [m(\mathrm{SPT}_{j+1}) - 1]T_C$$

(since job $j+1$ is the longest and assigned on the last tool as the last job)

$$\leqslant 2Z_T(\mathrm{OPT}_j) + [\lfloor 1.75m(\mathrm{OPT}_{j+1})\rfloor - 1]T_C$$

(as hypothesized and as shown above)

$$\leqslant 2Z_T(\mathrm{OPT}_j) + 2[m(\mathrm{OPT}_{j+1}) - 1]T_C$$

(for $m(\mathrm{SPT}_{j+1}) \geqslant 3$ or $m(\mathrm{OPT}_{j+1}) \geqslant 2$, the ratio $[\lfloor 1.75m(\mathrm{OPT}_{j+1})\rfloor - 1]/[m(\mathrm{OPT}_{j+1}) - 1]$ can be shown to be bounded above by 2)

$$\leqslant 2[Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C].$$

We will now show that $Z_T(\mathrm{OPT}_{j+1}) \geqslant Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C$. There are two cases to consider. First, suppose that, in OPT_{j+1}, the longest job, job $j+1$, is assigned to the last position on the last tool, $m(\mathrm{OPT}_{j+1})$ and that removing job $j+1$ yields the j-job sequence SEQ_j. We have

$$\begin{aligned} Z_T(\mathrm{OPT}_{j+1}) &= Z_T(\mathrm{SEQ}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C \\ &\geqslant Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C. \end{aligned}$$

Next, suppose that job $j+1$ is not assigned as above in OPT_{j+1}, but to the last position on a preceding tool. Remove job $j+1$ and replace it with the job that is in the last position on the last tool. Say, this yields the j-job sequence SEQ'_j. Clearly,

Table 1
Computational results on SPT performance ratio (ρ)

n	m	T_L	T_C/T_L								
			0.1			1.0			10.0		
			Best	Average	Worst	Best	Average	Worst	Best	Average	Worst
20	4	31	1.00	1.00	1.01	1.00	1.02	1.06	1.00	1.06	1.15
	5	24	1.00	1.00	1.01	1.00	1.03	1.08	1.00	1.07	1.18
25	4	39	1.00	1.00	1.01	1.00	1.02	1.06	1.00	1.04	1.13
	5	30	1.00	1.00	1.00	1.00	1.01	1.04	1.00	1.03	1.09
30	4	47	1.00	1.00	1.00	1.00	1.01	1.04	1.00	1.03	1.09
	5	36	1.00	1.00	1.01	1.00	1.02	1.04	1.00	1.05	1.10

$$Z_T(\mathrm{OPT}_{j+1}) \geqslant Z_T(\mathrm{SEQ}'_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C \geqslant Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C.$$

We have now effectively shown that $Z_T(\mathrm{OPT}_{j+1}) \geqslant Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C$.

Combining the two inequalities on $Z_T(\mathrm{SPT}_{j+1})$ and $Z_T(\mathrm{OPT}_{j+1})$ from above, we get

$$Z_T(\mathrm{SPT}_{j+1})/Z_T(\mathrm{OPT}_{j+1}) \leqslant 2[Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C]/[Z_T(\mathrm{OPT}_j) + [m(\mathrm{OPT}_{j+1}) - 1]T_C] \leqslant 2.$$

This completes the proof. We have shown that SPT has $\rho \leqslant 2$. We are, however, not able to show at this time that this bound is tight.

Finally, to test the performance of the SPT heuristic empirically, we solve an mixed integer linear programming (MILP) formulation [3] using CPLEX. The SPT heuristic is coded in the C language and compiled with the Gnu C compiler. All runs are made on a SPARCstation 10 machine operating under SunOS 5.4. We try three levels of n (20, 25 and 30), two levels of m (4 and 5) and three levels of T_C/T_L (0.1, 1.0 and 10.0); for each combination of n and m, an appropriate value of T_L (as shown in Table 1) is used and the integer T_C value for each such combination is derived based on the T_C/T_L ratio and through rounding. There is thus a total of 18 combinations of n, m and T_C/T_L; for each, 10 problem instances are randomly generated by drawing the p_j's from a discrete uniform distribution over [1,10].

While we do not collect information on CPU times, it appears that the SPT heuristic executes within a few milliseconds and that the MILP algorithm takes significantly longer (occasionally more than an hour of CPU time beyond $n = 30$ and $m = 5$). Table 1 shows the best, average and worst values of the SPT performance ratio. In no case does this ratio exceed 1.18; for small T_C/T_L values (which are likely to be encountered in practice), the average actually stays at or below 1.03. Our study is too limited in its scope for us to be able to make any sweeping claims about the performance of SPT in general. However, we can at least say that it is quite effective. This is further borne out by comparisons carried out against more sophisticated heuristics [4].

References

[1] I. Adiri, J. Bruno, E. Frostig, A.H.G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, Acta Informatica 26 (1989) 679–696.
[2] M.S. Akturk, S. Avci, Tool allocation and machining conditions optimization for CNC machines, European Journal of Operational Research 94 (1996) 335–348.
[3] M.S. Akturk, J.B. Ghosh, E.D. Gunes, Scheduling with tool changes to minimize total completion time, Technical Report, Dept. of Industrial Engr., Bilkent University, Ankara, Turkey, 1999.
[4] M.S. Akturk, J.B. Ghosh, E.D. Gunes, Scheduling with tool changes to minimize total completion time: A study of heuristics and their performance, Naval Research Logistics 50 (2003) 15–30.
[5] S. Albers, G. Schmidt, Scheduling with unexpected machine breakdowns, Discrete Applied Mathematics 110 (1999) 269–280.
[6] S. Anily, J. Bramel, D. Simchi-Levi, Worst-case analysis of heuristics for the bin-packing problem with general cost structures, Operations Research 42 (1994) 287–298.

[7] Y. Crama, Combinatorial optimization models for production scheduling in automated manufacturing systems, European Journal of Operational Research 99 (1997) 136–153.
[8] G.H. Graves, C.Y. Lee, Scheduling maintenance and semiresumable jobs on a single machine, Naval Research Logistics 46 (1999) 845–863.
[9] E. Gray, A. Seidmann, K.E. Stecke, A synthesis of decision models for tool management in automated manufacturing, Management Science 39 (1993) 549–567.
[10] P. Kouvelis, An optimal tool selection procedure for the initial design phase of a flexible manufacturing system, European Journal of Operational Research 55 (1991) 201–210.
[11] E.L. Lawler, J.M. Moore, A functional equation and its application to resource allocation and sequencing problems, Management Science 16 (1969) 77–84.
[12] C.Y. Lee, Machine scheduling with an availability constraint, Journal of Global Optimization 9 (1996) 395–416.
[13] C.Y. Lee, S.D. Liman, Single machine flow-time scheduling with scheduled maintenance, Acta Informatica 29 (1992) 375–382.
[14] X. Qi, T. Chen, F. Tu, Scheduling the maintenance on a single machine, Journal of the Operational Research Society 50 (1999) 1071–1078.
[15] G. Schmidt, Scheduling with limited machine availability, European Journal of Operational Research 121 (2000) 1–15.
[16] C.S. Tang, E.V. Denardo, Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches, Operations Research 36 (1988) 767–777.
[17] P. Tomek, Tooling strategies related to FMS management, The FMS Magazine 5 (1986) 102–107.