CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Greenwich Academic Literature Archive

Greenwich Academic Literature Archive (GALA)
– the University of Greenwich open access repository
http://gala.gre.ac.uk

Citation for published version:

Rustogi, Kabir and Strusevich, Vitaly A. (2012) Single machine scheduling with general positional deterioration and rate-modifying maintenance. Omega (International Journal of Management Science), 40 (6). 791–-804. ISSN 0305-0483

Publisher's version available at:
http://dx.doi.org/10.1016/j.omega.2011.12.007

Please note that where the full text version provided on GALA is not the final published version, the version made available will be the most up-to-date full-text (post-print) version as provided by the author(s). Where possible, or if citing, it is recommended that the publisher's (definitive) version be consulted to ensure any subsequent changes to the text are noted.

Citation for this version held on GALA:

Rustogi, Kabir and Strusevich, Vitaly A. (2012) Single machine scheduling with general positional deterioration and rate-modifying maintenance. London: Greenwich Academic Literature Archive. Available at: http://gala.gre.ac.uk/7594/

Contact: gala@gre.ac.uk

Omega 40 (2012) 791–804

Contents lists available at SciVerse ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

Single machine scheduling with general positional deterioration and rate-modifying maintenance

Kabir Rustogi, Vitaly A. Strusevich *

School of Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, UK

ARTICLE INFO

Article history:
Received 23 May 2011
Accepted 14 December 2011
This manuscript was processed by Associate Editor Dolgui.
Available online 11 January 2012

Keywords:
Scheduling
Maintenance
Sequencing
Deterioration
Single machine
Assignment problem

ABSTRACT

We present polynomial-time algorithms for single machine problems with generalized positional deterioration effects and machine maintenance. The decisions should be taken regarding possible sequences of jobs and on the number of maintenance activities to be included into a schedule in order to minimize the overall makespan. We deal with general non-decreasing functions to represent deterioration rates of job processing times. Another novel extension of existing models is our assumption that a maintenance activity does not necessarily fully restore the machine to its original perfect state. In the resulting schedules, the jobs are split into groups, a particular group to be sequenced after a particular maintenance period, and the actual processing time of a job is affected by the group that job is placed into and its position within the group.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

One of the current trends in scheduling research is to increase the practical relevance of deterministic machine scheduling models, which is achieved by the introduction and study of enhanced models that combine scheduling and logistic decision-making. These include models that address the issues of machine non-availability, transportation and delivery, resource management, controlling the processing characteristics of jobs and machines, and many others.

Planning machine maintenance is one of these features, and its importance for production enterprises and service organizations is widely recognized by both practitioners and management scientists; see for example, popular books on maintenance [1,2], and various Internet emporiums such as www.plant-maintenance.com, www.maintenanceworld.com, www.maintenancesources.com. The following quotation from the influential paper by Gopalakrishnan et al. [3] is especially close to the spirit of this paper:

> "Industrial systems used in the production of goods are subject to deterioration and wear with usage and age. System deterioration results in increased breakdowns leading to higher production costs and lower product quality. A well-implemented, planned preventive maintenance (PM) program can reduce costly breakdowns…. Deciding what PM tasks to do, and when, constitutes a critical resource allocation and scheduling problem."

As seen from the quotation above, in the planning of machine maintenance the decision-maker is faced with a trade-off between two processes: (i) *deterioration* of the processing conditions, and (ii) *allocation of a maintenance period* (MP) in order to restore or improve these conditions. However, until very recently the processes (i) and (ii) have not been fully integrated in the models studied in the scheduling literature. There is a long list of papers on process (i) alone, in which the sequencing problems with job deterioration have been analyzed. On the other hand, in various models of deterministic scheduling with maintenance, the focus has mainly been on placing an MP to satisfy certain requirements (e.g., the number of MPs to be scheduled, or the deadline for an MP to start or to finish, or periodicity of MP), but not on the effect that a scheduled maintenance might have on the processing conditions.

This paper considers single machine scheduling models in which the processing times of the jobs increase with their position in a schedule due to the worsening of processing conditions and running machine maintenance improves these conditions, making the processing times shorter. The decision-maker determines how many MPs to allocate and when to start each of them in order to minimize the *makespan*, i.e., the completion time of the last job to be processed.

* Corresponding author. Tel.: +44 20 8331 8662; fax: +44 20 8331 8665.
E-mail addresses: K.Rustogi@greenwich.ac.uk (K. Rustogi), V.Strusevich@gre.ac.uk, V.Strusevich@greenwich.ac.uk (V.A. Strusevich).

0305-0483/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2011.12.007

Below we present a brief literature overview, mainly concentrating on scheduling models with job deterioration or/and maintenance considerations to minimize the makespan. In all models that we consider n jobs should be processed on a single machine without preemption.

Informally, in scheduling with *deterioration* it is assumed that the later a job starts, the longer it takes to process. Less relevant to this paper is the opposite effect, known as *learning*, under which the actual processing time of a job gets shorter, provided that the job is scheduled later. Scheduling problems with these two types of non-constant processing times have received considerable attention; we refer to the surveys [4–6] for recent state-of-the-art reviews in this area, as well as for references to practical applications of these models. Among most common rationales for deterioration, the authors often mention the loss of the processing quality of machinery over time and/or the decrease in the productivity of a human operator who gets tired. In a typical scheduling model with a deterioration effect, each job j is given the value of its 'standard' or 'normal' processing time p_j. For every deterioration model, there is a particular rule that explains how exactly the value of p_j grows, depending either on the start time of a job (*time deterioration*) or on its position in the processing sequence (*positional deterioration*). Scheduling models of the former type are well covered in a recent book by Gawiejnowicz [7]. Below we focus on the positional deterioration effects. For a single machine problem to minimize the makespan, the models with a positional deterioration effect that have been studied so far include those in which the actual duration of a job j scheduled in the r-th position is equal to one of the following: $p_j r^a$, $p_j r^{a_j}$, $p_j \gamma^{r-1}$ or $p_j + b_j r$. The corresponding solution algorithms are presented in [6,8–11]. As a rule the solution is based on reducing the original problem to a linear assignment problem, either in a general form or to a special case. We review these results in more detail in Section 2.

There are several common drawbacks shared by most of the publications on scheduling with deterioration/learning. First, in many papers similar algorithmic ideas are applied to problems with minor differences in formulation, while the common features of these problems are overlooked. For instance, as we demonstrate in Section 2, for the single machine problem to minimize the makespan most known results remain true for a more general form of positional deterioration, e.g., $p_j g(r)$ and $p_j g_j(r)$, where g and g_j are general non-decreasing functions. Second, the practical impact of research on scheduling models with a deterioration effect alone is somewhat questionable: unless the processing conditions are improved by maintenance, the processing times will grow to unacceptable values, especially for a large number of jobs and high deterioration rates.

Another direction of research related to this study deals with scheduling problems with machine maintenance. There are numerous papers on scheduling with fixed machine non-availability intervals, and one possible interpretation of such an interval is to understand it as an allocated time slot to perform machine maintenance. We refer to the two most recent surveys in this area, see [12,13]. Problems in which non-availability (maintenance) intervals of a fixed duration have to be scheduled periodically, with a given bound on how long a machine can work between two MPs are studied in [14–18]. In these models, the jobs are to be packed into the gaps created between the MPs; however, the processing times of the jobs remain unaffected by maintenance. Other models of periodic maintenance focus on minimizing functions that include operational and maintenance costs; see [19,20].

Under another maintenance scenario initiated by Kubzin and Strusevich [21,22], the machines are subject to a compulsory maintenance, and its duration depends on its start time. Still, in these papers, the processing times of the jobs do not depend on their place in a schedule with respect to an MP.

A common drawback of the papers on machine maintenance reviewed above is that they fail to address the issue of machine deterioration and improving its processing conditions, so that the introduced periods are maintenance periods only by name, not by nature. This calls for a study of enhanced models which involve both general deterioration effects and rate-modifying maintenance activities, and have motivated us to write this paper.

One of the first papers that study an effect of maintenance on processing conditions is that by Lee and Leon [23], in which an MP is treated as a *rate-modifying* activity. They look at the problem of scheduling a single MP and assume that the processing time of a job j that is sequenced before the MP is p_j, while if it is sequenced after the MP the processing time becomes $\lambda_j p_j$, where $0 < \lambda_j < 1$. Polynomial-time algorithms for various scheduling problems, including due date assignment and due window assignment, with a single rate-modifying activity are presented in [24–26]. A generalized model in which the duration of a rate-modifying activity depends on its start time is studied in [27]. What makes the impact of these results limited, is the fact that typically only one MP is introduced and the issue of machine deterioration is not fully incorporated.

Studies that consider integrated deterministic machine scheduling models with deterioration effects and machine maintenance have started to appear only very recently. Given their high practical relevance, one may expect that research on these models may become one of the major directions in deterministic machine scheduling in the near future. At the moment we only mention the papers by Kuo and Yang [28], Zhao and Tang [29] and Yang and Yang [30], who give polynomial-time algorithms for single machine problems to minimize the makespan with specific positional deterioration effects and machine maintenance that fully restores the processing conditions. We review these papers in more detail in the subsequent sections. Lodree and Geiger [31] combine a single rate-modifying activity with time-dependent deterioration.

Our paper delivers polynomial-time algorithms for single machine problems with generalized positional deterioration effects and machine maintenance. The MPs improve the processing conditions and the number of MPs to be included into a schedule is decided by the decision-maker. Unlike many of our predecessors, we do not look at different deterioration scenarios (polynomial, exponential, etc.) but deal with general non-decreasing functions to represent deterioration rates. We also make a more realistic assumption regarding machine maintenance. Our model allows a novel assumption that an MP does not necessarily fully restore the machine to its original state. Thus, even after an MP, there might remain some wear and tear in the machine's conditions and therefore a possibility of a worse initial condition after each MP. In the resulting schedules, the MPs divide the jobs into several groups in the processing sequence, and the actual processing time of a job is affected by the group that job is placed into and its position within the group. We are not aware of any previously studied models that benefit from such a general set of assumptions, in particular, from a three-way dependency (job-group-position) of actual processing times.

What we deliver is a collection of polynomial-time algorithms to handle these generalized models. And again, our research compares favorably with what has been done earlier. Typically, previously used methods, applicable to simpler models, would include a straightforward use of the assignment problem or scheduling by priority rules justified by pairwise-interchange argument. We, on the other hand, offer a variety of more efficient methods, such as, the use of a reduced search procedure based on establishing a form of convexity of the objective function

(see Section 3), and reduction to a sequence of dynamically generated rectangular assignment problems (see Section 5), among others.

2. Preliminaries

In our main model, the jobs of set $N=\{1,2,\ldots,n\}$ have to be processed on a single machine. The jobs are available for processing at time zero and are independent, i.e., there are no precedence constraints and any processing sequence is feasible. At time zero, the machine is assumed to be in perfect processing state, and its processing conditions deteriorate as the number of jobs processed increases. Running machine maintenance may restore the state of the machine, either completely or partially. Notice that during each MP no job processing takes place.

Each job $j\in N$ is associated with its normal processing time p_j; essentially p_j is the duration of the processing of job j, provided that job j is the first to be processed on the machine under perfect conditions. For a particular schedule, assume that the jobs are partitioned into k *groups*, $1\le k\le n$, one to be scheduled before the first MP and one after each of the $k-1$ MPs. The actual processing time of job j may depend on

(i) the group it has been placed into,
(ii) the position of the job in the processing sequence in that group.

In the most general model under consideration the actual processing time of job j that is sequenced in the position $r\ge 1$ in group x is given by

$$p_j^{[x]}(r)=p_j g_j^{[x]}(r),\quad j\in N,\quad 1\le r\le n,\ 1\le x\le k\le n. \tag{1}$$

We will call the values $g_j^{[x]}(r)$ *deterioration factors*. We will distinguish between several special cases of this general model:

Job-independent, group-independent deterioration: In this case, it is assumed that

$$g_j^{[x]}(r)=g(r),\quad 1\le x\le n,$$

for all jobs $j\in N$. The function g is given in the form of an ordered array of numbers such that

$$1=g(1)\le g(2)\le\cdots\le g(n). \tag{2}$$

Such a model represents a case where the machine conditions are perfectly restored to the "as good as new" state after each MP. Since the machine is brought back to the same state each time, all groups are indistinguishable, thereby making deterioration factors group-independent.

Job-independent, group-dependent deterioration: In this case, it is assumed that

$$g_j^{[x]}(r)=g^{[x]}(r),\ 1\le x\le n,$$

for all jobs $j\in N$. The function g is given in the form of a collection of ordered arrays of numbers such that for a particular group x we have

$$1\le g^{[x]}(1)\le g^{[x]}(2)\le\cdots\le g^{[x]}(n),\quad 1\le x\le n, \tag{3}$$

where it is known that $g^{[1]}(1)=1$. Notice that in this model it is assumed that after an MP the machine is brought to a condition that is not necessarily perfect, and is no better than its condition after the previous MP. In such a case every position (including the first position) will have a worse deterioration factor than its counterpart in an earlier group, such that

$$g^{[1]}(r)\le g^{[2]}(r)\le\cdots\le g^{[n]}(r),\quad 1\le r\le n. \tag{4}$$

It should be noted that in a schedule with $k, 1\le k\le n$, groups, a particular group can have a maximum of $n-k+1$ jobs. This reflects the fact that each of the other $k-1$ groups contains at least one job.

Job-dependent, group-dependent deterioration: This is the most general case that is governed by (1). Such a model represents a scenario where each job wears out the machine in a different way, hence each job $j\in N$ is associated with a unique set of deterioration factors. For each job $j\in N$, the deterioration factors are given in the form of a collection of ordered array of numbers, similar to (3) and (4):

$$1\le g_j^{[x]}(1)\le g_j^{[x]}(2)\le\cdots\le g_j^{[x]}(n),\quad 1\le x\le n, \tag{5}$$

$$g_j^{[1]}(r)\le g_j^{[2]}(r)\le\cdots\le g_j^{[n]}(r),\quad 1\le r\le n. \tag{6}$$

Informally, these conditions imply that for each job j (i) the deterioration factors do not decrease with each group as the position advances, and (ii) the deterioration factors for a fixed position do not decrease as the number of MPs grows. Again, it should be noted that in a schedule with $k, 1\le k\le n$ groups, a particular group can have a maximum of $n-k+1$ jobs.

Given a schedule S, let $C_j(S)$ denote the completion time of job $j\in N$. In this paper, we focus on the makespan objective function, i.e., we are looking for a schedule that minimizes the maximum completion time $C_{\max}(S)=\max\{C_j(S)|j\in N\}$. If it is clear which schedule is being discussed, we might drop the reference to S and simply write C_j and $C_{\max}$.

To refer to the scheduling problems under consideration, we extend the standard three-field classification scheme employed for scheduling problems. For our general model, we will write $1|p_j g_j^{[x]}(r),MP|C_{\max}$. Here, as usual, the first field implies that we deal with a single-machine environment, and the third field points out that our objective function is the makespan. The first item in the middle field indicates the type of job deterioration; we will use appropriate simplified notation for various special cases of the model. Further in the middle field, we write "*MP*" to stress that the machine is subject to maintenance, but the number of MPs is not known and should be determined by the decision-maker together with an optimal schedule. We number the MPs by the integers $1,2,\ldots,x,\ldots$, and the duration of the x-th MP is assumed equal to a given number $t^{[x]}$. Thus, the problem essentially captures a trade-off between the fast processing of the jobs on a well-maintained machine and the time that is required to guarantee that the machine is in an acceptable condition.

Often, we also consider a version of the problem in which the number of MPs is known in advance and is equal to $k-1$. In this case, we will write $1|p_j g_j^{[x]}(r),MP[k-1]|C_{\max}$ with the first item in the middle field appropriately adjusted for various cases. Besides, for the problems with no maintenance, we will drop the second item in the middle field and write either $1|p_j g_j(r)|C_{\max}$ (for job-dependent deterioration) or $1|p_j g(r)|C_{\max}$ (for job-independent deterioration). In the remainder of this paper, for problem $1|p_j g_j^{[x]}(r),MP[k-1]|C_{\max}$ with k groups, let $S(k)$ denote the best schedule and $C_{\max}(S(k))$ denote the optimal value of the makespan.

In addition to the models with fixed durations of the MPs, we also study those in which the duration of an MP depends on its start time.

We conclude this section with a brief review on the relevant scheduling problems with deterioration and no maintenance. Most of these results reduce the corresponding scheduling problem to a version of the linear assignment problem. Recall that in an $n\times n$ assignment problem, given a matrix of values c_{ji} it is required to select n elements, exactly one from each row (associated with jobs) and exactly one from each column (associated with a position), so that their sum is minimized. See the recent

monograph [32], for an excellent review of the results on this problem.

Several papers study problem $1|p_j g(r)|C_{\max}$ for specific functions $g(r)$ and show that an optimal permutation can be found by the well-known LPT *(Largest Processing Time)* priority rule. Recall that the LPT rule renumbers the jobs in non-increasing order of their normal processing times, i.e.,

$$p_1 \geq p_2 \geq \cdots \geq p_n. \tag{7}$$

For example, Mosheiov [10] proves the optimality of the LPT rule for the *polynomial deterioration* that is defined by the function

$$g(r) = r^a, \quad a > 0, \tag{8}$$

while Gordon et al. [6] do that for the *exponential deterioration* that is defined by the function

$$g(r) = \gamma^{r-1}, \quad \gamma > 0.$$

It is surprising that prior to [33] no attempts have been made to look at the problem with general positional deterioration, defined by a non-increasing array of the values $g(r)$, $r = 1, 2, \ldots, n$, as in (2).

Below we establish an easy link between problem $1|p_j g(r)|C_{\max}$ and a special case of the assignment problem. Given a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, let $C_{\max}(\pi)$ denote the makespan of the schedule in which the jobs are processed in accordance with that permutation. Then the actual processing time of a job in the r-th position in that sequence is $p_{\pi(r)} g(r)$, so that

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} g(r).$$

Notice that this function can be understood as the objective in an assignment problem with $c_{ji} = p_j g(i)$, $1 \leq i, j \leq n$. Recall that the values $g(r)$, $r = 1, 2, \ldots, n$ are non-decreasing. Then, the classical result by Hardy et al. [34] (see also Proposition 5.8 in [32]) states that for the jobs renumbered in accordance with (7), for any permutation π we have that

$$\sum_{r=1}^{n} p_r g(r) \leq \sum_{r=1}^{n} p_{\pi(r)} g(r) \leq \sum_{r=1}^{n} p_{n-r+1} g(r), \tag{9}$$

which implies that the following statement holds.

Proposition 1. *For problem $1|p_j g(r)|C_{\max}$ with general job-independent positional deterioration, a permutation that defines an optimal solution can be obtained in $O(n \log n)$ time by sorting the jobs by the LPT rule.*

Notice that if a function g is given as a collection of n non-increasing values (this is known as a general positional learning effect) the optimal permutation can be found by the SPT rule that renumbers the jobs in non-decreasing order of their normal processing times. For polynomial learning and exponential learning the optimality of the SPT rule is proved in [8,6], respectively.

For job-dependent deterioration rates, problem $1|p_j g_j(r)|C_{\max}$ directly reduces to an $n \times n$ assignment problem with $c_{ji} = p_j g_j(i)$, $1 \leq i, j \leq n$. Here a decision variable x_{ji} is equal to 1 if a job $j \in N$ is assigned to a position i in the processing sequence; otherwise it is equal to zero. See [8,35] for the description of this approach; notice that in both quoted papers the authors address the problem with a learning effect, rather than a model with job deterioration. The assignment problem can be solved by the famous Hungarian algorithm; see [32] for more details. Thus, the following statement holds.

Proposition 2. *For problem $1|p_j g_j(r)|C_{\max}$ with general job-dependent positional deterioration, a permutation that defines an optimal solution can be obtained in $O(n^3)$ time by solving the corresponding assignment problem with $c_{ji} = p_j g_j(i)$, $1 \leq i, j \leq n$.*

Notice that some authors study scheduling problems with a special form of position-dependent processing times. For example, Bachman and Janiak [11] consider a single machine problem to minimize the makespan in which the processing time of job j scheduled in position r is given by $p_j + b_j r$, where p_j is the normal processing time and b_j is a job-dependent rate (positive in the case of deterioration and negative in the case of learning). However, even for a more general situation, e.g., when the actual processing time of job j scheduled in position r is defined by $p_j(a_j + b_j g(r))$, $b_j > 0$, we have that for an arbitrary permutation of jobs:

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} a_{\pi(r)} + \sum_{r=1}^{n} p_{\pi(r)} b_{\pi(r)} g(r),$$

where the first term is a constant, and the second term can be seen as the makespan in problem $1|(p_j b_j) g(r)|C_{\max}$, i.e., for the problem with the normal processing times defined by $p_j b_j$, $j \in N$. Thus, an optimal permutation can be found by Proposition 1, i.e., by applying the LPT rule to the values $p_j b_j$. This implies that job-dependent positional deterioration of this type does not deserve a separate treatment.

In the subsequent sections, we consider various versions of the single machine scheduling problems to minimize the makespan, provided that the jobs are subject to positional deterioration and the machine is subject to maintenance.

3. Job-independent group-independent deterioration

The main problem addressed in this section is $1|p_j g(r), MP|C_{\max}$, in which the jobs are subject to position-dependent deterioration, with the deterioration factors g given as a sorted array (2).

The fact that the deterioration is group-independent implies that each MP restores the machine to its initial "as-good-as-new" state. The objective is to find the optimal number of maintenance activities such that the overall makespan of the schedule is minimized. In a related problem $1|p_j g(r), MP[k-1]|C_{\max}$ the number of MPs is known to be $k-1 \geq 0$, i.e., the jobs are split into k non-empty groups.

A special case of problem $1|p_j g(r), MP|C_{\max}$ with a polynomial deterioration function given by (8) and equal maintenance times $t^{[x]} = t$ is studied by Kuo and Yang [28]. Actually, they also look at the problem $1|p_j + b_j r, MP|C_{\max}$, but as discussed in the end of Section 2, this model does not require a separate consideration. Kuo and Yang [28] turn to problem $1|p_j r^a, MP[k-1]|C_{\max}$ with a fixed number of maintenance periods and prove what they call the *group balance principle*. According to this principle, for problem $1|p_j r^a, MP[k-1]|C_{\max}$ there exists an optimal schedule with k groups such that the difference between the number of jobs in any two groups is at most one. Proposition 1 implies that in each group the jobs are sequenced in the LPT order. As a result, an algorithm for solving problem $1|p_j r^a, MP[k-1]|C_{\max}$ scans the jobs in the LPT order and assigns them one by one to the smallest available position across all k groups. Such an algorithm requires $O(n)$ time to output an optimal schedule $S(k)$ and the optimal makespan $C_{\max}(S(k))$, provided that the LPT sequence of jobs is found. Trying all possible values of k from 1 to n, they obtain a solution to the original problem $1|p_j r^a, MP|C_{\max}$ as the best of all found schedules $S(k)$. It should be noted that Kuo and Yang [28] make a mistake when they claim that their algorithm requires $O(n \log n)$ time: in fact they need $O(n^2)$ time since they do not take into account the linear time that is needed to compute the function $C_{\max}(S(k))$ for each schedule $S(k)$.

It is easy to transfer the results given in [28] to an arbitrary deterioration function given by (2) and arbitrary maintenance times. One can verify that the group balance principle holds in this general case as well, and the running times of the algorithms

for solving problems $1|p_j g(r), MP[k-1]|C_{\max}$ and $1|p_j g(r), MP|C_{\max}$ remain $O(n \log n)$ (or $O(n)$ if the LPT sequence of the jobs is known) and $O(n^2)$, respectively.

The value $C_{\max}(S(k))$ can be seen as $P(S(k))+T(k)$, where $P(S(k))$ denotes the sum of actual durations of the jobs in an optimal schedule with k groups, and $T(k)$ is the total duration of all $k-1$ MPs. Kuo and Yang [28] have made a conjecture that in the case of a polynomial deterioration function the sequence of values $C_{\max}(S(k))$, $1 \le k \le n$, might be V-shaped with respect to k. Recall that a sequence $A(k)$ is called *V-shaped* if there exists a k_0, $1 \le k_0 \le n$, such that

$$A(1) \ge \cdots \ge A(k_0-1) \ge A(k_0) \le A(k_0+1) \le \cdots \le A(n).$$

If this were true for $C_{\max}(S(k))$, $1 \le k \le n$, then at most $\lceil \log_2 n \rceil$ values of k should be tried to solve the original problem $1|p_j r^a, MP|C_{\max}$.

Below we show that under some reasonable conditions on the maintenance times the sequence $C_{\max}(S(k))$, $1 \le k \le n$, is in fact V-shaped, even for a general job-independent deterioration effect.

Lemma 1. *For problem $1|p_j g(r), MP[k-1]|C_{\max}$, let the jobs be numbered in the LPT order (7). There exists an optimal schedule in which a job j, $1 \le j \le n$, is assigned to position $\lceil j/k \rceil$ in some group.*

Proof. Proposition 1 implies that the jobs in each group are sequenced in the LPT order of their normal times. To minimize the value $P(S(k))$, we need to assign the jobs one by one to the smallest available position. This can be done by distributing the first k jobs to the first positions in each of the k groups, then the next k jobs going to the second positions in each of the k groups, and so on, until all jobs have been sequenced.

If $j=ak$ then the predecessors of j are placed into the first a positions of groups $1,2,\ldots,k-1$ and take $a-1$ positions of group k, so that job j gets position $a=\lceil j/k \rceil$. If $j=ak+b$ for $1 \le b \le k-1$, then the predecessors of j will take the first a positions in each group and additionally the (a+1)-th position in each of the groups $1,2,\ldots,b-1$, so that job j gets position $a+1=\lceil j/k \rceil$ in group b. □

It should be noticed that Lemma 1 essentially implies that the group balance principle holds for problem $1|p_j g(r), MP[k-1]|C_{\max}$, even in the case of arbitrary deterioration factors.

Hence, the actual processing time of a job $j \in N$ in a schedule that is optimal for problem $1|p_j g(r), MP[k-1]|C_{\max}$ is equal to $p_j g(\lceil j/k \rceil)$, so that the makespan for that problem becomes

$$C_{\max}(S(k)) = P(S(k))+T(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right) + \sum_{x=1}^{k-1} t^{[x]}. \tag{10}$$

For problem $1|p_j g(r), MP|C_{\max}$, we need to determine the optimal number of groups k to be opened such that the function $C_{\max}(S(k))$, $1 \le k \le n$, is minimized. As k increases, $P(S(k))$ becomes smaller since new groups are added and a greater number of smaller positions become available. At the same time, $T(k)$ becomes larger, with more maintenance activities being performed. The sequence $C_{\max}(S(k))$ captures the trade-off between its two components, $P(S(k))$ and $T(k)$.

An obvious way to find an optimal number of groups k^* and the corresponding optimal makespan for problem $1|p_j g(r), MP|C_{\max}$ is formally described in the following algorithm.

Algorithm 1

Step 1. For k from 1 to n, compute $C_{\max}(S(k))$ by formula (10).
Step 2. Output k^* such that

$$C_{\max}(S(k^*)) = \min\{C_{\max}(S(k)) \mid 1 \le k \le n\}$$

and stop.

This method essentially coincides with the one employed in [28] and requires $O(n^2)$ time, since for each value of k computing $C_{\max}(S(k))$ takes $O(n)$ time.

Now we demonstrate that the sequence $C_{\max}(S(k))$, $1 \le k \le n$, is in fact V-shaped, provided that each sequence $P(S(k))$, $1 \le k \le n$, and $T(k)$, $1 \le k \le n$, is convex. Recall that a sequence $A(k)$, $1 \le k \le n$, is called *convex* if

$$A(k) \le \tfrac{1}{2}(A(k-1)+A(k+1)), \quad 2 \le k \le n-1. \tag{11}$$

Our reasoning is based on the following property proved in our previous paper [36].

Lemma 2. *The sequence*

$$B(k) = \sum_{j=1}^{q} g\left(\left\lceil \frac{j}{k} \right\rceil\right), \quad 1 \le k \le n, \tag{12}$$

is convex for each q, $1 \le q \le n$.

We start with the following preliminary statement.

Lemma 3. *Given a k, $1 \le k \le n$, let $S(k)$ be a schedule in which the jobs are numbered in the LPT order and job $j \in N$ is assigned to the position $\lceil j/k \rceil$ of some group, so that $P(S(k)) = \sum_{j=1}^{n} p_j g(\lceil j/k \rceil)$. Then the sequence of values $P(S(k))$, $1 \le k \le n$, is convex.*

Proof. Due to (11), we need to prove that

$$P(S(k)) \le \tfrac{1}{2}(P(S(k-1))+P(S(k+1)), \quad 2 \le k \le n-1$$

or, equivalently,

$$\sum_{j=1}^{n} p_j \left[2g\left(\left\lceil \frac{j}{k} \right\rceil\right) - g\left(\left\lceil \frac{j}{k+1} \right\rceil\right) - g\left(\left\lceil \frac{j}{k-1} \right\rceil\right)\right] \le 0,$$
$$2 \le k \le n-1.$$

For a given $j \in \{1,2,\ldots,n\}$, define

$$A_j(k) = \left[2g\left(\left\lceil \frac{j}{k} \right\rceil\right) - g\left(\left\lceil \frac{j}{k+1} \right\rceil\right) - g\left(\left\lceil \frac{j}{k-1} \right\rceil\right)\right],$$
$$2 \le k \le n-1.$$

By Lemma 2, due to the convexity of the sequence $B(k)$, $1 \le k \le n$, we deduce that

$$\sum_{i=1}^{q} A_i(k) \le 0 \tag{13}$$

for each k, $2 \le k \le n-1$, and all q, $1 \le q \le n$.

In order to prove the lemma, we need to demonstrate that the inequality:

$$\sum_{j=1}^{n} p_j A_j(k) \le 0 \tag{14}$$

holds for each k, $2 \le k \le n-1$.

Fix a k, $2 \le k \le n-1$, and transform

$$\begin{aligned}\sum_{j=1}^{n} p_j A_j(k) &= p_n\left(\sum_{i=1}^{n} A_i(k) - \sum_{i=1}^{n-1} A_i(k)\right) \\ &\quad + p_{n-1}\left(\sum_{i=1}^{n-1} A_i(k) - \sum_{i=1}^{n-2} A_i(k)\right) + \cdots + p_1 A_1(k) \\ &= p_n \sum_{i=1}^{n} A_i(k) + (p_{n-1}-p_n)\sum_{i=1}^{n-1} A_i(k) \\ &\quad + (p_{n-2}-p_{n-1})\sum_{i=1}^{n-2} A_i(k) + \cdots + p_1 A_1(k)\end{aligned}$$

$$= \sum_{j=2}^{n} \left[(p_{j-1} - p_j) \sum_{i=1}^{j-1} A_i(k) \right] + p_n \sum_{i=1}^{n} A_i(k).$$

The last right-hand expression is non-positive due to (7) and (13), so that the desired inequality (14) holds and the sequence $P(S(k))$, $1 \le k \le n$, is convex. □

It is easy to verify that the sum of two convex sequences is convex and any convex sequence is *V*-shaped; the analogies of these statements for convex functions are well-known. Thus, Lemma 3 immediately implies

Theorem 1. *For problem* $1|p_j g(r), MP|C_{\max}$, *the sequence* $C_{\max}(S(k))$, $1 \le k \le n$, *is V-shaped, provided the sequence* $T(k)$, $1 \le k \le n$, *is convex.*

The assumption on convexity of the sequence $T(k)$, $1 \le k \le n$, is in fact quite natural; e.g., it will hold if the durations of the MPs, $t^{[x]}, 1 \le x \le k-1$, either form a non-decreasing sequence (an MP performed later in the schedule takes longer to restore the initial condition of machinery) or are equal (as in [28]).

Theorem 1 allows us to find an optimal schedule with an optimal number of groups k^* by performing binary search with respect to k. As a result at most $\lceil \log_2 n \rceil$ values of k need to be explored, so that the running time of this method becomes $O(n \log n)$. If, however, the sequence $T(k)$, $1 \le k \le n$, is not convex then we cannot guarantee that the sequence $C_{\max}(S(k))$, $1 \le k \le n$, is *V*-shaped, and problem $1|p_j g(r), MP|C_{\max}$ remains solvable in $O(n^2)$ time by Algorithm 1.

4. Job-independent group-dependent deterioration

As discussed in Section 2, a maintenance activity may not be able to fully restore the machine to its perfect conditions. Thus, in this section we deal with models in which the deterioration factors depend on the group, along with the position in that group. The initial conditions after each MP keep becoming worse, so that the deterioration factor for a particular position might not be the same across all groups in a schedule. Recall that within a group, the deterioration factors are non-decreasing and are governed by (3). Besides, for a fixed position, the sequence of the deterioration factors does not decrease as the number of MPs grows; see (4).

We denote the problem of minimizing the makespan in these settings by $1|p_j g^{[x]}(r), MP|C_{\max}$, and its counterpart in which the number of MPs is known and is equal to $k-1$ by $1|p_j g^{[x]}(r), MP[k-1]|C_{\max}$. For problem $1|p_j g^{[x]}(r), MP[k-1]|C_{\max}$, the actual processing time $p_j^{[x]}(r)$ of a job $j \in N$ in a position r of group $x, 1 \le x \le k$, is given by

$$p_j^{[x]}(r) = p_j g^{[x]}(r), \quad 1 \le r \le n-k+1, \ 1 \le x \le k,$$

where the deterioration factors $g^{[x]}(r)$ obey the conditions (3) and (4). We are not aware of any prior research on this model.

It can be checked that for this extended model the group balancing principle does not apply. Intuitively, this is because if a group deterioration is high enough, it would be more profitable to have more jobs in earlier groups. Indeed, take an instance of problem $1|p_j g^{[x]}(r), MP|C_{\max}$ with four jobs, such that

$$p_1 = p_2 = p_3 = p_4 = 1;$$

$$g^{[1]}(1) = 1, \quad g^{[1]}(2) = 1, \quad g^{[1]}(3) = 2, \quad g^{[1]}(4) = 4;$$

$$g^{[2]}(1) = 2.5, \quad g^{[2]}(2) = 3, \quad g^{[2]}(3) = 4;$$

$$g^{[3]}(1) = 5, \quad g^{[3]}(2) = 5;$$

$$g^{[4]}(1) = 5$$

and each maintenance time is equal to 0.5. As above, let $S(k)$ denote the best schedule with k groups. Then $C_{\max}(S(1)) = 1+1+2+4 = 8$, while $C_{\max}(S(3)) = 1+1+0.5+2.5+0.5+5 = 10.5$ and $C_{\max}(S(4)) = 1+0.5+2.5+0.5+5+0.5+5 = 15$. In schedule $S(2)$ group 1 will contain three jobs and group 2 one job, so that $C_{\max}(S(2)) = 1+1+2+0.5+2.5 = 7$, and this schedule is optimal. On the other hand, applying the group balancing principle to a schedule with two groups, we will obtain a schedule $S'(2)$ with two jobs in each group, for which $C_{\max}(S'(2)) = 1+1+0.5+2.5+3 = 8$.

The counterexample above implies that for finding an optimal schedule for problem $1|p_j g^{[x]}(r), MP|C_{\max}$ we need a technique other than that based on the group balancing principle.

The approach discussed below is motivated by Proposition 1 and inequalities (9), according to which to obtain the best schedule $S(k)$ with k groups, the jobs should be scanned in the LPT order and a job p_j should be matched to the j-th smallest available deterioration factor $g^{[x]}(r), 1 \le x \le k, 1 \le r \le n-k+1$.

Let $G(k)$ denote a list of n smallest deterioration factors that are available across all positions from each of the k groups. The list is sorted in non-decreasing order. Let $\gamma_i(k)$ denote the i-th element in the list $G(k)$, so that $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$. This implies that

$$C_{\max}(S(k)) = P(S(k)) + T(k) = \sum_{j=1}^{n} p_j \gamma_j(k) + \sum_{x=1}^{k-1} t^{[x]}. \tag{15}$$

Suppose for a particular j, $1 \le j \le n$ we have $\gamma_j(k) = g^{[x]}(r)$, where $1 \le x \le k$ and $1 \le r \le n-k+1$, then schedule $S(k)$ is obtained by assigning job j to position r of group x. This can be implemented by running the following rather straightforward algorithm that solves problem $1|p_j g^{[x]}(r), MP|C_{\max}$ in $O(n^3)$ time.

Algorithm 2

Step 1. For $k=1$, define $\gamma_r(1) = g^{[1]}(r)$, $1 \le r \le n$. Compute $C_{\max}(S(1))$ by formula (15). Define $k' := n$.

Step 2. For k from 2 to n do

(a) Take the factors $g^{[x]}(r)$, $1 \le x \le k, 1 \le r \le n-k+1$. Find Γ, the n-th smallest of these factors and create the list $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$ of n factors that do not exceed Γ. If necessary, sort the list $G(k)$ in non-decreasing order of its values.

(b) Compute $C_{\max}(S(k))$ by formula (15). If $P(S(k)) \ge P(S(k-1))$ then define $k' := k-1$ and break the loop by moving to Step 3; otherwise, continue the loop with the next value of k.

Step 3. Find the value k^*, $1 \le k^* \le k'$, such that

$$C_{\max}(S(k^*)) = \min\{C_{\max}(S(k)) \mid 1 \le k \le k'\}.$$

If the condition $P(S(k)) \ge P(S(k-1))$ in Step 2b holds, this implies that the addition of the k-th group does not provide positions with deterioration factors smaller than those in the list $G(k-1)$. If this happens for the k-th group, all groups that will be opened after this will provide even worse deterioration factors because of (4), and hence the makespan cannot be reduced by running more MPs after the k'-th group is opened. Thus, no further values of k should be examined and the best schedule should be found from the set $\{S(k) \mid 1 \le k \le k'\}$. A similar loop-breaking rule is also included in several subsequent algorithms.

Theorem 2. *Algorithm 2 solves problem* $1|p_j g^{[x]}(r), MP|C_{\max}$ *in* $O(n^3)$ *time.*

To prove the theorem, we only need to estimate the running time of Algorithm 2. In each iteration for k, we use the median finding procedure for determining Γ, so that it takes $O(k(n-k+1)) = O(nk)$ time to create a list of n smallest factors. To obtain a sorted list $G(k)$, we need additionally $O(n \log n)$ time, and then $O(n)$ time to compute $C_{\max}(S(k))$. Thus, the overall running time of Algorithm 2 is at most $\sum_{k=1}^{n} O(nk+n \log n) = O(n^3)$. The above algorithm is more or less a brute force method for finding an optimal schedule, since nearly all possible options are evaluated at each iteration.

However, to solve the given problem, the same idea can be implemented faster, as described below. The new algorithm manipulates with the list $G(k)$ described above and another list, which we denote by $H(k)$ and which contains all factors coming from the k-th group, $1 \le k \le n$.

Algorithm 3

Step 1. For $k=1$, define a sorted list

$$H(1) := (g^{[1]}(1), g^{[1]}(2), \ldots, g^{[1]}(n))$$

and $G(1) := H(1)$, so that $\gamma_j(r) = g^{[1]}(r)$, $1 \le r \le n$. Compute $C_{\max}(S(1))$ by formula (15). Define $k' := n$.

Step 2. For k from 2 to n, determine the sorted list

$$H(k) := (g^{[k]}(1), g^{[k]}(2), \ldots, g^{[k]}(n-k+1))$$

and create the list $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$ that contains n smallest elements in the merger of the lists $G(k-1)$ and $H(k)$. Compute $C_{\max}(S(k))$ by formula (15). If $P(S(k)) = P(S(k-1))$ then define $k' := k-1$ and break the loop by moving to Step 3; otherwise, continue the loop with the next value of k.

Step 3. Find the value k^*, $1 \le k^* \le k'$, such that

$$C_{\max}(S(k^*)) = \min\{C_{\max}(S(k)) \mid 1 \le k \le k'\}.$$

This algorithm, unlike Algorithm 2, generates the list of n smallest deterioration factors by merging the list of previously known factors and the list of the factors provided by adding a new group. Unlike in Algorithm 2, the inequality $P(S(k)) > P(S(k-1))$ is impossible, and we use the condition $P(S(k)) = P(S(k-1))$ for breaking the loop. The latter condition implies that the addition of the k-th group does not provide positions with deterioration factors smaller than those in the list $G(k-1)$. Thus, the lists $G(k)$ and $G(k-1)$ are identical, and no further values of k should be examined.

Each of the lists $H(k)$, $1 \le k \le n$, is available from the input data in the sorted form. In each iteration of the loop in Step 2, the list $G(k)$ is obtained by merging two sorted lists, each containing at most n elements. Besides, the value of $C_{\max}(S(k))$ can be computed in $O(n)$ time. Thus, the overall running time of the new algorithm is $O(n^2)$.

Theorem 3. *Algorithm 3 solves problem* $1|p_j g^{[x]}(r), MP|C_{\max}$ *in* $O(n^2)$ *time.*

Notice that Algorithm 3 can be applied to solving the problem $1|p_j g(r), MP|C_{\max}$ considered in Section 3, in which case for each position r the equalities $g^{[x]}(r) = g(r)$ hold for all x, $1 \le x \le n$. Under these assumptions, Algorithm 3 essentially behaves as Algorithm 1 and still requires $O(n^2)$ time.

Algorithm 3 clearly overperforms Algorithm 2 because it captures the idea of reducing the search for potential deterioration factors for a schedule with k groups to examining the factors used in an optimal schedule with $k-1$ groups and the new factors contained in the k-th group. Still, we have included Algorithm 2 here, since the speed-up achieved in Algorithm 3 cannot be used for various extensions of the problem under consideration, in which deterioration factors are either less structured as the ones in problem $1|p_j g^{[x]}(r), MP|C_{\max}$ or are dynamically changing in every iteration; see Section 6. On the other hand, we take the productive idea behind Algorithm 3 further, to the models with job-dependent deterioration effects; see Section 5.

5. Job dependent, group dependent deterioration

Now we turn to the most general situation, in which positional deterioration factors of the jobs depend on the group of the jobs and on the job itself. We reduce the resulting problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ to a sequence of problems $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ with exactly k groups, i.e., we assume that exactly $k-1$ maintenance periods are included. For problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$, the actual processing time $p_j^{[x]}(r)$ of a job $j \in N$ in a position r of group x, $1 \le x \le k$, is given by

$$p_j^{[x]}(r) = p_j g_j^{[x]}(r), \quad 1 \le r \le n-k+1, \ 1 \le x \le k,$$

where the deterioration factors $g_j^{[x]}(r)$ obey the conditions (5) and (6).

The problem closest to that introduced in this section is $1|p_j r^{a_j}, MP|C_{\max}$ which was studied by Zhao and Tang [29]. In their problem, the deterioration factors are polynomial job-dependent and group-independent, and an $O(n^4)$-time algorithm based on solving a series of assignment problems with a square cost matrix is given. In this section, we derive an algorithm for the most general pattern of positional deterioration by solving a sequence of rectangular assignment problems with dynamically generated columns. The resulting running time of our algorithm is $O(n^4)$.

Problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ cannot be solved using any of the previously discussed algorithms in this paper, since now each job is associated with unique deterioration factors. Below we describe an algorithm that solves the original problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ by reducing each problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ to a rectangular assignment problem with n rows, each corresponding to a job $j \in N$, and $m = (n-k+1)k$ columns. For our purposes, it is convenient to number the columns by a string of the form (x,r), where x refers to a group, $1 \le x \le k$, and r indicates a position within the group. Thus, the first $n-k+1$ columns $(1,1),(1,2),\ldots,(1,n-k+1)$ of the matrix are associated with the positions in group 1, the next $n-k+1$ columns $(2,1),(2,2),\ldots,(2,n-k+1)$ are associated with the positions in group 2, etc. Create an $n \times m$ cost matrix $C = (c_{j,(x,r)})$ containing all possible values of $p_j^{[x]}(r)$. More precisely, the value of element $c_{j,(x,r)}$ at the intersection of the j-th row and v-th the column of matrix C for v, $1 \le v \le m$, such that $v = (n-k+1)(x-1)+r$, where $1 \le x \le k$ and $1 \le r \le n-k+1$, is defined by

$$c_{j,(x,r)} = p_j g_j^{[x]}(r). \tag{16}$$

For a group x, $1 \le x \le k$, let $l^{[x]}$ denote the number of potential positions of that group that can be used in a schedule for problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$. Then problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ reduces to a rectangular assignment problem written out below:

$$\begin{aligned}
\text{Min} \quad & \sum_{j \in N} \sum_{x=1}^{k} \sum_{r=1}^{l^{[x]}} c_{j,(x,r)} z_{j,(x,r)} \\
\text{s.t.} \quad & \sum_{j \in N} z_{j,(x,r)} \le 1, \ 1 \le x \le k, 1 \le r \le l^{[x]} \\
& \sum_{x=1}^{k} \sum_{r=1}^{l^{[x]}} z_{j,(x,r)} = 1, \quad j \in N \\
& z_{j,(x,r)} \in \{0,1\}, \quad j \in N, \ 1 \le x \le k, \quad 1 \le r \le l^{[x]},
\end{aligned} \tag{17}$$

where in the case under consideration $l^{[x]} = n-k+1$ for $1 \le x \le k$.

The algorithm to solve such a rectangular assignment problem has been outlined in [37]. The running time of this algorithm is $O(n^2m)$, $m \geq n$. See also [32] for a discussion of the rectangular assignment problem and other modifications of the classical assignment problem.

Later in this section we will establish several properties of the algorithm that solves the rectangular assignment problem (17), this is why we reproduce its main steps below, as described in [37]. Let either a row or a column of the matrix C be called a *line*. The algorithm manipulates with the cost matrix, reduces the original (positive) elements on a line-by-line basis, so that some of them become zeros. Two zeros that do not belong to the same line are called *independent*. There are two types of labels applied to a zero: it can be *starred* to become 0* or *primed* to become 0′. During the run of the algorithm, some lines are said to be *covered*. In all iterations of the algorithm, the starred zeros are independent, and their number is equal to the number of the covered lines, with each covered line containing exactly one 0*. The algorithm stops having found n starred zeros in the current matrix. The primed zeros in a current partial solution are seen as potential candidates to become starred zeros. In our case, the matrix C due to (5) and (6) has a special structure that is characterized by

- non-decreasing order of the elements of the same row that are placed in the columns associated with positions of the same group and
- non-decreasing order of the elements placed in the same row and in those columns associated with a given position $r, 1 \leq r \leq n-k+1$, of each group, from group 1 to group k.

In the description below we make appropriate alterations to the algorithm given in [37] to reflect that special structure. These alterations affect neither the optimality nor the running time of the algorithm.

Algorithm 4 (see Bourgeois and Lassale [37])

Step 0. Consider a row of the matrix C, subtract the smallest element from each element in the row. Do the same for all other rows.

Step 1. Considering the rows in an arbitrary order, search for a zero, Z, in the current row that is located in the left-most column with no starred zeros. If Z is found, star Z. Repeat for each row of the matrix. Go to Step 2.

Step 2. Cover every column containing a 0*. If n columns are covered, the starred zeros form the desired independent set. Otherwise, go to Step 3.

Step 3. Choose a non-covered zero and prime it; in the case of several available zeros prime the one in the left-most column. Consider the row containing the primed zero. If there is no starred zero in this row, go to Step 4. If there is a starred zero Z in this row, cover this row and uncover the column of Z. Repeat until all zeros are covered. Go to Step 5.

Step 4. There is a sequence of alternating starred and primed zeros constructed as follows: let Z_0 denote the uncovered 0′. Let Z_1 denote the 0* in Z_0's column (if any). Let Z_2 denote the 0′ in Z_1's row. Continue in a similar way until the sequence stops at a 0′, Z_{2a}, which has no 0* in its column. Unstar each starred zero of the sequence, and star each primed zero of the sequence. Erase all primes and uncover every line. Return to Step 2.

Step 5. Let h denote the smallest non-covered element of the current matrix. Add h to each covered row, then subtract h from each uncovered column. Return to Step 3 without altering any asterisks, primes, or covered lines.

An iteration of Algorithm 4 is considered complete when all zeros are covered by the end of Step 3. After this, a transition is made to Step 5, where we search for the minimal elements in the uncovered part of the matrix and convert them to zero. At the end of an iteration, one of the two outcomes is possible: either new 0*'s are added to the matrix, or not. If the total number of 0*'s in the matrix is less than n, the existing 0*'s represent a partial solution to the assignment problem. If the total number of 0*'s in the matrix is equal to n, then the solution is considered complete and the optimal assignment is given by the positions occupied by the 0*'s. Below we analyze the outcome of an iteration of this algorithm.

Lemma 4. *Suppose that after some iteration of Algorithm 4, for each x, $1 \leq x \leq k$, the column $(x,l^{[x]})$ is such that it contains a 0*, while none of the columns (x,r) for $r > l^{[x]}$ contain a 0*. If no column (x,r) for $1 \leq r \leq n-k+1$ contains a 0*, then define $l^{[x]} = 0$. Then, for each x, $1 \leq x \leq k$, such that $l^{[x]} \geq 1$, it follows that for each r, $1 \leq r \leq l^{[x]}$, column (x,r) contains a 0*.*

Proof. See Appendix A.

We shall now outline an algorithm that solves the problem $1|p_jg_j^{[x]}(r),MP|C_{\max}$ by solving n rectangular assignment problems of the form (17), each corresponding to problem $1|p_jg_j^{[x]}(r), MP[k-1]|C_{\max}$. This is a brute-force algorithm, since we allow all positions in each group to be potentially used in schedule $S(k)$.

Algorithm 5

Step 1. Find an optimal schedule $S(1)$ with no maintenance periods, in which all jobs are placed in group 1. This is done by solving the $n \times n$ assignment problem with

$$c_{j,(1,r)} = p_jg_j^{[1]}(r), j \in N, 1 \leq r \leq n.$$

Compute $P(S(1))$ as the optimal value of the objective function in this assignment problem. Determine schedule $S(1)$ in which job j is processed in the r-th position of group 1 if and only if $z_{j,(1,r)} = 1$. Define $C_{\max}(S(1)) = P(S(1))$. Define $T(1) := 0$, $k := 1$ and $k' := n$.

Step 2. With the current value of k do

(a) Update $T(k+1) = T(k) + t^{[k]}$. Compute all elements of the matrix C by (16). Run Algorithm 4 to solve the resulting $n \times (n-k)(k+1)$ rectangular assignment problem of the form (17) with $l^{[x]} = n-k$ for each x, $1 \leq x \leq k+1$.

(b) Compute $P(S(k+1))$, the optimal value of the objective function in that assignment problem and $C_{\max}(S(k+1)) = P(S(k+1)) + T(k+1)$. If $P(S(k+1)) \geq P(S(k))$ then define $k' := k$ and break the loop by moving to Step 3; otherwise, proceed to Step 2(c).

(c) Update $k := k+1$. If $k \leq n-1$, repeat Step 2; otherwise go to Step 3.

Step 3. Find the value k^*, $1 \leq k^* \leq k'$, such that

$$C_{\max}(S(k^*)) = \min\{C_{\max}(S(k)) | 1 \leq k \leq k'\}.$$

Notice that the loop in Step 2 of Algorithm 5 is broken at $k = k'$ if $P(S(k'+1)) \geq P(S(k'))$. This corresponds to the existence of empty groups in schedule $S(k'+1)$ and in all schedules $S(k)$ that we might find for $k > k'+1$. In this case, the optimum value k^* in Step 2 should be sought for among the values of k between 1 and k'.

Theorem 4. *Algorithm 5 solves problem $1|p_jg_j^{[x]}(r),MP|C_{\max}$ in $O(n^5)$ time.*

Proof. Suppose that for some k the solution of the assignment problem (17) related to problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ is found. Then $z_{j,(x,r)}=1$ implies that job j is assigned to the r-th position of group x. The conditions of (17) mean that each job will be assigned to a position and no position will be used more than once. Lemma 4 guarantees that the found assignment admits a meaningful scheduling interpretation, because for each of the k groups either several consecutive positions starting from the first are filled or the group is not used at all. Also notice that an empty group cannot be followed by a group with at least one filled position, since it is always better to move the job from the first position of a group to the empty first position of an earlier group.

A solution to problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ is obtained by solving an assignment problem in $O(n^2(n-k+1)k)$ time. Thus, solution to problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ can be found in $O(n^5)$ time. □

The problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ can be solved faster, if we prove the fact that for creating an optimal schedule $S(k+1)$ for $k+1$ groups, it suffices to consider the positions that become available with the introduction of the $(k+1)$-th group, plus the positions that were used in the optimal schedule $S(k)$. Notice that this is the same philosophy that was used earlier to speed up Algorithm 2, however its validity for the job-dependent case is not as obvious.

Lemma 5. *Under the conditions of Lemma 4, while processing all uncovered zeros, the values $l^{[x]}$ for each x, $1\le x\le k$, either remain the same or exactly one of them increases by 1 for every zero considered.*

Proof. See Appendix B. □

Without loss of generality, assume that each of the k groups in schedule $S(k)$ that is optimal for an instance of the problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ with a set of jobs N is not empty. Let $l^{[x]}$ denote the number of positions used in a group x, $1\le x\le k$, so that $\sum_{x=1}^{x=k} l^{[x]}=n$.

Now, find a schedule $\tilde{S}(k)$ that is optimal for an instance of problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ with a set of jobs $\tilde{N}\subset N$. Let in schedule $\tilde{S}(k)$ the number of filled positions in a group x, $1\le x\le k$, be denoted by $\tilde{l}^{[x]}$. Lemmas 4 and 5 immediately imply that $\tilde{l}^{[x]}\le l^{[x]}$ for each x, $1\le x\le k$.

For a set of jobs N, consider problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$ with k groups and problem $1|p_j g_j^{[x]}(r), MP[k]|C_{\max}$ with $k+1$ groups. Let $S(k)$ and $S(k+1)$ be the corresponding optimal schedules. Suppose that $\tilde{N}\subset N$ is the subset of jobs that are assigned to the first k groups in schedule $S(k+1)$. Then, as observed above, none of these groups uses more positions in $S(k+1)$ than it does in $S(k)$. Given the fact, $\sum_{x=1}^{x=k} l^{[x]}=n$, this implies the following statement.

Theorem 5. *Let $S(k)$ be an optimal schedule for problem $1|p_j g_j^{[x]}(r), MP[k-1]|C_{\max}$. In order to find schedule $S(k+1)$ that is optimal for problem $1|p_j g_j^{[x]}(r), MP[k]|C_{\max}$, it is sufficient to use the n positions used in schedule $S(k)$ together with $n-(k+1)+1$ new positions in group $k+1$.*

The algorithm below starts with finding the best schedule with one group, and having found the best schedule with k groups finds the best schedule with $k+1$ groups by solving an assignment problem with $O(n)$ columns and n rows. The columns to be used while solving the problem with $k+1$ groups are the n columns for which an assignment was found in the previous iteration and $n-(k+1)+1=n-k$ new columns corresponding to the new group $k+1$.

Algorithm 6

Step 1. Find an optimal schedule $S(1)$ with no maintenance periods, in which all jobs are placed in group 1. This is done by solving the assignment problem with

$$c_{jr}=p_j g_j^{[1]}(r), j\in N, 1\le r\le n.$$

Compute $P(S(1))$ as the optimal value of the objective function in this assignment problem. Determine schedule $S(1)$ in which job j is processed in the r-th position of group 1 if and only if $z_{j,(1,r)}=1$. Define $C_{\max}(S(1))=P(S(1))$. Define $T(1):=0, l^{[1]}=:n$, $k:=1$ and $k':=n$.

Step 2. With the current value of k do

(a) Update $T(k+1)=T(k)+t^{[k]}$. Define $l^{[k+1]}:=n-k$. Compute all values of the matrix C by (16) for columns $(1,1),\dots(1,l^{[1]}),\dots,(k,1),\dots(k,l^{[k]})$ and $(k+1,1),\dots,(k+1,l^{[k+1]})$. Run Algorithm 4 to solve the resulting $n\times(2n-k)$ rectangular assignment problem of the form (17) with the current values of $l^{[x]}$, $1\le x\le k+1$.

(b) Compute $P(S(k+1))$ as the optimal value of the objective function in that assignment problem and $C_{\max}(S(k+1))=P(S(k+1))+T(k+1)$. If $P(S(k+1))=P(S(k))$ then define $k':=k$ and break the loop by moving to Step 3; otherwise, determine schedule $S(k+1)$ in which job j is processed in the r-th position of group x, $1\le x\le k+1$, if and only if $z_{j,(x,r)}=1$. For each group x, $1\le x\le k+1$, determine the last filled position $l^{[x]}$.

(c) Update $k:=k+1$. If $k\le n-1$, repeat Step 2; otherwise go to Step 3.

Step 3. Find the value k^*, $1\le k^*\le k'$, such that

$$C_{\max}(S(k^*))=\min\{C_{\max}(S(k))\,|\,1\le k\le k'\}.$$

Similar to Algorithm 3, the loop-breaking condition in Algorithm 6 is $P(S(k+1))=P(S(k))$, since the inequality $P(S(k+1))>P(S(k))$ is impossible due to the selection of positions used in schedules $S(k+1)$ and $S(k)$.

Theorem 6. *Algorithm 6 solves problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ in $O(n^4)$ time.*

Proof. The correctness of Algorithm 6 is justified by Theorem 5. To estimate the running time, notice that in Step 1 an $n\times n$ assignment problem is solved in $O(n^3)$ time. For each value of k in Step 2, we solve a rectangular assignment problem which has n rows and $2n-k$ columns, of which n columns, namely $(1,1),\dots,(1,l^{[1]}),\dots,(k,1),\dots,(k,l^{[k]})$ are brought forward from the previous iteration and the remaining $n-k$ columns correspond to the new group $k+1$. Algorithm 4 will require $O(n^2(2n-k))$ time for each $k, 1\le k\le n-1$, so that the overall running time of Algorithm 6 is $O(n^4)$. □

Notice that Algorithm 6 can be applied to solving the problem $1|p_j g_j(r), MP|C_{\max}$, in which case for each position r the equalities $g_j^{[x]}(r)=g_j(r)$ hold for all x, $1\le x\le n$. Under these assumptions, Algorithm 6 essentially behaves as the algorithm provided by Zhao and Tang [29], who use the group balancing principle to solve the special case $1|p_j r^{a_j}, MP|C_{\max}$.

Example 1. Consider problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ with five jobs. A schedule can have up to four MPs, with maintenance durations given as

$$t^{[1]}=3,\quad t^{[2]}=3,\ t^{[3]}=3.5,\ t^{[4]}=4.$$

The jobs can be split in up to five groups. Table 1 represents the actual processing times of jobs from the set $N=\{1,2,3,4,5\}$, with

Table 1
Actual processing times for Example 1.

Job j	Group 1					Group 2			
	$p_j^{[1]}(1)$	$p_j^{[1]}(2)$	$p_j^{[1]}(3)$	$p_j^{[1]}(4)$	$p_j^{[1]}(5)$	$p_j^{[2]}(1)$	$p_j^{[2]}(2)$	$p_j^{[2]}(3)$	$p_j^{[2]}(4)$
1	5	6	7	8	9	6	6	8	9
2	10	12	13	15	16	11	12	14	15
3	1	1	2	2	3	1	2	3	4
4	3	5	7	8	9	4	5	7	9
5	7	7	7	8	8	7	7	8	8

Job j	Group 3			Group 4		Group 5
	$p_j^{[3]}(1)$	$p_j^{[3]}(2)$	$p_j^{[3]}(3)$	$p_j^{[4]}(1)$	$p_j^{[4]}(2)$	$p_j^{[5]}(1)$
1	6	7	8	7	8	7
2	12	13	14	12	14	13
3	2	2	3	3	4	4
4	4	6	8	5	7	5
5	7	7	8	8	8	8

Table 2
Run of Algorithm 6 for Example 1.

$k=1$	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
1	5	6	[7]	8	9
2	[10]	12	13	15	16
3	1	1	2	[2]	3
4	3	[5]	7	8	9
5	7	7	7	8	[8]

$P(S(1)) = 7+10+2+5+8 = 32$
$C_{\max}(S(1)) = P(S(1)) = 32$

$k=2$	(1,1)	(1,2)	(1,3)	~~(1,4)~~	~~(1,5)~~	(2,1)	(2,2)	~~(2,3)~~	~~(2,4)~~
1	5	6	7	8	9	6	[6]	8	9
2	[10]	12	13	15	16	11	12	14	15
3	1	[1]	2	2	3	1	2	3	4
4	3	5	7	8	9	[4]	5	7	9
5	7	7	[7]	8	8	7	7	8	8

$P(S(2)) = 6+10+1+4+7 = 28$
$C_{\max}(S(2)) = P(S(2)) + t^{[1]} = 28+3 = 31$

$k=3$	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(3,1)	(3,2)	(3,3)
1	5	6	7	6	[6]	6	7	8
2	[10]	12	13	11	12	12	13	14
3	1	[1]	2	1	2	2	2	3
4	3	5	7	[4]	5	4	6	8
5	7	7	[7]	7	7	7	7	8

$P(S(3)) = 6+10+1+4+7 = 28$
$C_{\max}(S(3)) = P(S(3)) + t^{[1]} + t^{[2]} = 28+3+3 = 34$

each row containing the values $p_j^{[x]}(r) = p_j g_j^{[x]}(r)$ for a job j when placed in position r of group x. Notice that for each job the values of $p_j^{[x]}(r)$ do not decrease within each group, and do not decrease for each position r as the number x of a group grows. This is consistent with (5) and (6). Besides, a group $x, 1 \le x \le n$, has only $n-x+1$ positions associated with it.

Table 2 shows the details of the run of Algorithm 6 for the above instance.

The run starts with $k=1$, i.e., we solve the 5×5 assignment problem. The optimal solution determines a schedule $S(1)$ with no MPs, in which the jobs are assigned to the positions marked by the boxes. Since all positions of Group 1 are used in this schedule, the list of possible positions in the next iteration $k=2$ includes all five positions of Group 1 and all four positions of Group 2. We solve the corresponding rectangular assignment problem, and the numbers marked with boxes determine a schedule $S(2)$ in which the jobs 2, 3 and 5 occupy the first three positions of Group 1, respectively, while the jobs 4 and 1 are respectively assigned to the first two positions of Group 2, after the MP of duration 3. The positions that are not used are crossed out; they will never be used in subsequent iterations. In the next iteration $k=3$ we use the positions associated with schedule $S(2)$ and three positions of the new Group 3. The method stops here with $k\square = 2$, since none of the positions of Group 3 is filled, i.e., $P(S(3)) = P(S(2))$ as in the loop-breaking rule. Schedules $S(1)$ and $S(2)$ are the two candidates for a global optimal solution, and we choose $S(2)$ with the smaller makespan.

6. Start time dependent maintenance

In this section, we consider a model in which the duration of a maintenance activity depends on the time that elapsed since the previous MP. Thus, the later a machine is sent for maintenance, the longer it takes to restore the machine to an acceptable condition. Such a model has been introduced by Kubzin and Strusevich [21,22] for shop scheduling models; see [27,30] for further developments.

In this section, we assume the duration $D^{[x]}(\tau)$ of the x-th maintenance period to be a linear function of its start time τ that is either measured from zero (if $x=1$) or from the completion of the $(x-1)$-th MP (for $2\le x\le n-1$). Thus,

$$D^{[x]}(\tau)=\alpha^{[x]}\tau+\beta^{[x]},\quad 1\le x\le n-1, \tag{18}$$

where $\alpha^{[x]}$ and $\beta^{[x]}$ are given positive constants such that

$$\alpha^{[1]}\le\alpha^{[2]}\le\cdots\le\alpha^{[n-1]}.$$

As stated by Kubzin and Strusevich [22], the rationale for such a representation is that a maintenance period includes a series of standard tests that require a constant time $\beta^{[x]}$ while $\alpha^{[x]}\tau$ corresponds to the duration of maintenance activities that depend on the state of the equipment.

For a schedule with $k-1$ maintenance activities and k groups, $1\le k\le n$, let $F^{[x]}$ denote the completion time of group x, and $P^{[x]}$ denote the total processing time of the jobs assigned to group x, $1\le x\le k$. We deduce

$$F^{[1]}=P^{[1]};$$

$$F^{[2]}=F^{[1]}+(\alpha^{[1]}P^{[1]}+\beta^{[1]})+P^{[2]}=(1+\alpha^{[1]})P^{[1]}+\beta^{[1]}+P^{[2]};$$

$$F^{[3]}=F^{[2]}+(\alpha^{[2]}P^{[2]}+\beta^{[2]})+P^{[3]}=(1+\alpha^{[1]})P^{[1]}+(1+\alpha^{[2]})P^{[2]}+(\beta^{[1]}+\beta^{[2]})+P^{[3]};$$

$$\cdots$$

$$F^{[k]}=\sum_{x=1}^{k-1}(1+\alpha^{[x]})P^{[x]}+P^{[k]}+\sum_{x=1}^{k-1}\beta^{[x]}.$$

Notice that $F^{[k]}$ is the makespan of a schedule with k groups.

First, assume that the deterioration factors are both group-dependent and position-dependent, but not job-dependent, as in Section 4. We denote the corresponding problem by $1|p_jg^{[x]}(r),MP(\tau)|C_{\max}$, where we write "$MP(\tau)$" to represent that the duration of each MP is start time dependent in the sense of (18).

The best schedule $S(k)$ with k groups can be found by solving the problem $1|p_jW^{[x]}(r),MP[k-1]|C_{\max}$ from Section 4, provided that the number of groups k is fixed and the modified deterioration rates, or more appropriately, the positional weights $W^{[x]}(r)$ are given by

$$W^{[x]}(r)=(1+\alpha^{[x]})g^{[x]}(r),\quad 1\le x\le k-1,\quad 1\le r\le n-k+1$$
$$W^{[k]}(r)=g^{[k]}(r),\quad 1\le r\le n-k+1 \tag{19}$$

and the maintenance duration $t^{[x]}$ is set equal to $\beta^{[x]}$.

For each k, problem $1|p_jW^{[x]}(r),MP[k-1]|C_{\max}$ can be solved in $O(nk)$ time by matching the jobs taken in the LPT order to n smallest weights $W^{[x]}(r)$. This can be done by the running Algorithm 2 with $g^{[x]}(r)=W^{[x]}(r)$ for all relevant values of x and r. Thus, the overall running time for solving problem $1|p_jg^{[x]}(r),MP(\tau)|C_{\max}$ is $O(n^3)$.

Notice that problem $1|p_jg(r),MP(\tau)|C_{\max}$, with group-independent deterioration rates, i.e., the version in which $g^{[x]}(r)=g(r)$ for all $x,1\le x\le n$, can be solved faster. We can directly apply Algorithm 3 with $H(1)=(g(1),g(2),\ldots,g(n))$ in Step 1 and with $H(k)=((1+\alpha^{[k]})g(1),(1+\alpha^{[k]})g(2),\ldots,(1+\alpha^{[k]})g(n-k+1)$ for each k in Step 2. Thus, problem $1|p_jg(r),MP(\tau)|C_{\max}$ requires only $O(n^2)$ time.

Below we present a small numerical example that illustrates the application of Algorithms 2 and 3 to problem $1|p_jg^{[x]}(r),MP(\tau)|C_{\max}$ with job-independent, group-dependent positional factors with start-time dependent maintenance. This example demonstrates why the use of a faster Algorithm 3 does not guarantee an optimal solution in this case, and we have to rely on Algorithm 2.

Example 2. Consider problem $1|p_jg^{[x]}(r),MP(\tau)|C_{\max}$ with six jobs, with the following normal processing times listed in an LPT order $p_1=10,\ p_2=9,\ p_3=6,\ p_4=3,\ p_5=3,\ p_6=2$.

A schedule can have up to five MPs, with the following parameters

$$\alpha^{[1]}=2,\quad \beta^{[1]}=4,$$

$$\alpha^{[2]}=3,\quad \beta^{[2]}=0,$$

$$\alpha^{[3]}=3,\quad \beta^{[3]}=1,$$

$$\alpha^{[4]}=4,\quad \beta^{[4]}=5,$$

$$\alpha^{[5]}=4,\quad \beta^{[5]}=3,$$

where the α-values are non-decreasing as required. As a result of these MPs, a schedule can be divided in up to six groups, with the deterioration factors that obey (3) and (4) presented in Table 3.

Table 4 shows the details of the run of Algorithm 2 for the above instance. Since $\gamma_r(3)=\gamma_3(2)$ for each r, $1\le r\le 6$, the algorithm stops after the iteration $k=3$, so that $k'=2$. The algorithm outputs the minimum value of the makespan from the set $\{C_{\max}(S(k))|1\le k\le 2\}$, which is $C_{\max}(S(2))$. In an optimal schedule for $k=2$, the sequence of jobs (3,6) is processed in the first group, while the sequence of jobs (1,2,4,5) is processed in the second group, after an MP of duration $D^{[1]}(10)=24$. The makespan of the resulting schedule is 99.

If Algorithm 3 is applied to the same instance then for $k=3$ the values of $\gamma_r(3)$ would be obtained as the list of 6 smallest values in the merger of the sequences $\gamma_r(2)$, $1\le r\le 6$, and $W^{[3]}(r)$, $1\le r\le 4$, resulting into the list (2,2,2,2,3,4), which is wrong. Thus, since in the model with the group-dependent factors the values $W^{[x]}(r)$ change dynamically as k grows, the list of the best multipliers found in the previous iteration is not relevant and Algorithm 3 is not applicable. On the other hand, if the factors are group-independent Algorithm 3 handles the resulting problem correctly and faster than Algorithm 2.

Now, consider the general case, in which the deterioration factors depend on a job, a group and a position. We denote the corresponding problem by $1|p_jg_j^{[x]}(r),MP(\tau)|C_{\max}$. The durations of the MP are still determined by (18). The best schedule $S(k)$ with k groups can be found by solving the modified problem

Table 3
Deterioration factors for Example 2.

Position (r)	$g^{[1]}(r)$	$g^{[2]}(r)$	$g^{[3]}(r)$	$g^{[4]}(r)$	$g^{[5]}(r)$	$g^{[6]}(r)$
1	1	2	2	3	3	4
2	2	2	2	4	4	
3	3	4	4	5		
4	5	5	5			
5	8	8				
6	10					

Table 4
Run of Algorithm 2 for Example 2.

$k=1$	r	$W^{[1]}(r)$	$\gamma_r(1)$	$\gamma_r(1)p_r$
	1	1	1	10
	2	2	2	18
	3	3	3	18
	4	5	5	15
	5	8	8	24
	6	10	10	20

$C_{\max}(S(1)) = \sum_{r=1}^{6} \gamma_r(1)p_r = 105$

$k=2$	r	$W^{[1]}(r)$	$W^{[2]}(r)$	$\gamma_r(2)$	$\gamma_r(2)p_r$
	1	3	2	2	20
	2	6	2	2	18
	3	9	4	3	18
	4	15	5	4	12
	5	24	8	5	15
	6			6	12

$C_{\max}(S(2)) = \sum_{r=1}^{6} \gamma_r(2)p_r + \beta^{[1]} = 95+4 = 99$

$k=3$	r	$W^{[1]}(r)$	$W^{[2]}(r)$	$W^{[3]}(r)$	$\gamma_r(3)$	$\gamma_r(3)p_r$
	1	3	8	2	2	20
	2	6	8	2	2	18
	3	9	12	4	3	28
	4	15	20	5	4	12
	5				5	15
	6				6	12

$C_{\max}(S(3)) = \sum_{r=1}^{6} \gamma_r(3)p_r + \beta^{[1]} + \beta^{[2]} = 95+4+0 = 99$

$1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$, provided that the number of groups k is fixed and the positional weights $W_j^{[x]}(r)$ are given by

$$W_j^{[x]}(r) = (1+\alpha^{[x]})g_j^{[x]}(r), \quad 1 \le x \le k-1, \ 1 \le r \le n-k+1, \ j \in N;$$

$$W_j^{[k]}(r) = g_j^{[k]}(r), \quad 1 \le r \le n-k+1, \ j \in N.$$

Thus, problem $1|p_j g_j^{[x]}(r), MP(\tau)|C_{\max}$ reduces to a modified problem $1|p_j W_j^{[x]}(r), MP|C_{\max}$. The latter problem can be solved as a sequence of rectangular assignment problems, as described in Section 5. Each of these assignment problems corresponds to problem $1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$, and the value $\beta^{[x]}$ is treated as the maintenance duration $t^{[x]}$, $1 \le x \le k-1$. The resulting problem can be solved by Algorithm 5 in $O(n^5)$ time. However, in the case under consideration we cannot guarantee that the positions used in an optimal schedule with $k+1$ groups contain the positions used in an optimal schedule with k groups. This is due to the fact that the deterioration rates change dynamically as the number of groups grows. As a result, Theorem 5 does not hold and the faster Algorithm 6 cannot be adapted.

We now turn to the problem in which the deterioration rates are group-independent, i.e., the version in which $g_j^{[x]}(r) = g_j(r)$ for all $x, 1 \le x \le n$. Yang and Yang [30] consider a special case of problem $1|p_j g_j(r), MP(\tau)|C_{\max}$ with a polynomial job-dependent deterioration factors $g_j(r) = r^{a_j}$, $j \in N$, and a start time dependent maintenance, such that the duration of each MP is equal to $\alpha\tau+\beta$. They reduce the problem to a sequence of assignment problems and give an algorithm that requires $O(n^5)$ time. We show that a more general problem $1|p_j g_j(r), MP(\tau)|C_{\max}$ can be solved faster. It reduces to the sequence of problems of the form $1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$ to be solved for each k, $1 \le k \le n$, where the modified deterioration rates become

$$W_j^{[x]}(r) = (1+\alpha^{[x]})g_j(r), \quad 1 \le x \le k-1, \quad 1 \le r \le n-k+1, \quad j \in N;$$

$$W_j^{[k]}(r) = g_j(r), \quad 1 \le r \le n-k+1, \quad j \in N.$$

In turn, each problem $1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$ reduces to an assignment problem. Suppose that for some value of k, $1 \le k \le n-1$, we have found an optimal schedule $S(k)$ for problem $1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$, so that in each group x, $1 \le x \le k$, the number of consecutively used positions is $l^{[x]}$, where $\sum_{x=1}^{k} l^{[x]} = n$. While making a transition to solving the assignment problem associated with problem $1|p_j W_j^{[x]}(r), MP[k]|C_{\max}$ with $k+1$ groups, notice that the deterioration rates used in problem $1|p_j W_j^{[x]}(r), MP[k-1]|C_{\max}$ for groups from 1 up to $k-1$ remain the same, while the rates previously used in group k will now be used in group $k+1$. Additionally, for group k the previously used rates will be multiplied by $(1+\alpha^{[k]})$. By Theorem 5, this means that in an optimal schedule $S(k+1)$ at most $l^{[x]}$ positions will be used in each group x, $1 \le x \le k-1$, and at most $l^{[k]}$ positions will be used in group $k+1$; i.e., at most n positions in total can be used in these groups. Additionally, up to $n-k+1$ positions can be used in group k. This implies that we can adapt Algorithm 6 for solving problem $1|p_j g_j(r), MP(\tau)|C_{\max}$ with group-independent rates, so that the problem can be solved in $O(n^4)$ time.

7. Conclusion

The results of this paper on the enhanced single machine scheduling models that combine machine deterioration and maintenance are given in Tables 5 and 6. The developed algorithms are applicable to a wider range of models and either improve or match the running times known for less general models. We hope this study will initiate further research on machine scheduling with rate-modifying maintenance activities.

Table 5
Results on problem $1|p_j g_j^{[x]}(r), MP|C_{\max}$ and its variations.

Deterioration factors	Additional conditions	Running time	Reference
r^a, $a>0$	$t^{[x]} = t$	$O(n^2)$	Kou and Yang [28]
$g(r)$		$O(n^2)$	Algorithm 1
$g(r)$	$T(k)$-convex	$O(n \log n)$	Binary Search
$g^{[x]}(r)$		$O(n^2)$	Algorithm 3
r^{a_j}, $a_j>0$	$t^{[x]} = t$	$O(n^4)$	Zhao and Tang [29]
$g_j^{[x]}(r)$		$O(n^4)$	Algorithm 6

Table 6
Results on problem $1|p_j g_j^{[x]}(r), MP(\tau)|C_{\max}$ and its variations (see Section 6) with $D^{[x]}(\tau) = \alpha^{[x]}\tau + \beta^{[x]}$ to define the duration of the x-th maintenance period that starts at time τ.

Deterioration factors	Additional conditions	Running time	Reference or Algorithm to be Modified
$g(r)$		$O(n^2)$	Algorithm 3
$g^{[x]}(r)$		$O(n^3)$	Algorithm 2
r^{a_j}, $a_j>0$	$\alpha^{[x]} = \alpha, \beta^{[x]} = \beta$	$O(n^5)$	Yang and Yang [30]
$g_j(r)$		$O(n^4)$	Algorithm 6
$g_j^{[x]}(r)$		$O(n^5)$	Algorithm 5

Appendix A

Proof of Lemma 4. Suppose that the lemma does not hold, i.e., for some x there exists a column (x,r') that does not contain a 0*, where $r' < l^{[x]}$. Assume that a 0* appears in position $(j,(x,l^{[x]}))$. Since each covered line contains a 0*, it follows that the column (x,r') is uncovered.

Observe that the only way for a zero to lose its "star" label is Step 4 of the algorithm, but in this case a 0′ from the same column becomes a 0*. In short, once a column gets a 0*, then it will contain a 0* (possibly, in a different row) in all subsequent iterations. On the other hand, if a column does not have a 0*, then it has not contained a 0* in all preceding iterations. Thus, column (x,r') has not contained a 0* in all previous iterations, and this column has always been uncovered.

Suppose that in some iteration i, the element in position $(j,(x,l^{[x]}))$ is reduced to zero as the current minimal element (see Step 5). At the time the element is uncovered, i.e., in all previous iterations column $(x,l^{[x]})$ has not contained a 0* and has not been covered, exactly as column (x,r'). Thus, up to the i-th iteration both columns (x,r') and $(x,l^{[x]})$ have been subject to the same transformations in Step 5. In particular, the elements in positions $(j,(x,r'))$ and $(j,(x,l^{[x]}))$ either have been left the same in all previous iterations with row j covered or have been reduced by the value of the current minimal element in each previous iteration when row j was not covered. Since originally $p_j g_j^{[x]}(r') \le p_j g_j^{[x]}(l^{[x]})$, we deduce that in the beginning of iteration i the element in position $(j,(x,r'))$ is less or equal to the element in position $(j,(x,l^{[x]}))$. At the end of iteration i, we know that position $(j,(x,l^{[x]}))$ becomes zero. Now since the element in position $(j,(x,r'))$ cannot be negative, we can deduce that it must be zero at the end of iteration i. Therefore, all elements in the consecutive positions $(j,(x,r')),(j,(x,r'+1)),\ldots,(j,(x,l^{[x]}))$ of row j are equal to zero. We know that Step 3 of Algorithm 4 processes the zero in position $(j,(x,r'))$ earlier than all other uncovered zeros in this row. Thus, the zero in position $(j,(x,r'))$ will be primed.

If there is no 0* in row j, Step 4 of the algorithm will star the primed zero in position $(j,(x,r))$, as we know that column (x,r) does not contain any 0* either.

If there is a 0* in row j, then the corresponding column, say, column v, is covered. The algorithm in Step 3 will uncover column v and cover row j. If this uncovers a 0 in column v, say, in row $u \ne j$, then Step 4 of the algorithm will find a path that traverses through the three positions $(u,v),(j,v)$ and $(j,(x,r'))$, and redistribute the stars. As a result, a 0* would appear in position $(j,(x,r'))$.

Now we consider the situation when there are no uncovered zeros in column v, row j is covered, the zero in position (j,v) is starred, the zero in position $(j,(x,r'))$ is primed and the zeros in positions $(j,(x,r'+1)),\ldots,(j,(x,l^{[x]}))$ have no labels. By the lemma conditions, we know that eventually the zero in position $(j,(x,l^{[x]}))$ becomes starred. In the iterations that follow iteration i, the only way to get a 0* in row j in a position other than (j,v) is to start with some 0′ in the uncovered part of the current matrix, and to find a path (as described in Step 4) that starts with the chosen 0′ and finishes with the two positions (j,v) and $(j,(x,r'))$, that contain a 0* and a 0′, respectively. However, as a result of the corresponding redistributions of stars, a 0* will appear in position $(j,(x,r'))$.

We have proved that once column (x,r') gets a 0*, it will always contain a 0* in all subsequent iterations. Hence, our assumption that for some x, there exists a column (x,r') that does not contain a 0*, where $r' < l^{[x]}$, is false; thereby proving Lemma 4.

Appendix B

Proof of Lemma 5. Among all uncovered zeros, in Step 3 choose zero Z that appears in the earliest column, and prime it. If the row containing the primed zero contains a 0*, then we cover that row and uncover the column, so that Z does not become a 0* yet. Notice that as a result of this transformation, all zeros contained in the same row with Z are covered, including those found in Step 5, and they will not be considered in this iteration of the algorithm. Thus, the values $l^{[x]}$ for each x, $1 \le x \le k$, remain the same for all remaining zeros in this row.

If Z does not have any 0* in its row (see Step 4), a path is formed which is an alternating sequence of the primed and starred zeros that starts with the primed zero Z ends with another 0′. In such a situation, all 0*'s in the path are unstarred and each 0′ is converted to a 0*. Since the number of primed zeros in the path is always one greater than the number of starred zeros, it follows that once the swap is performed we have exactly one extra 0* that will replace the last 0′ in the path. This includes a situation which happens when for zero Z neither its row, nor its column contains a 0*, so that the path simply consists of Z alone and Z itself becomes a 0*. Thus, a new 0* will appear in the column that has not had a starred zero earlier, while all other columns will maintain the number of contained 0*'s. Suppose that the new 0* appears in position (x,r) for some group x, $1 \le x \le k$, and $r \ge 1$. If $r=1$, then the old value $l^{[x]} = 0$ grows by 1. Otherwise, we know from Lemma 4 that in a particular group x, all 0*'s appear in consecutive columns $(x,1),\ldots,(x,l^{[x]})$. Since column (x,r) is the next to the column $(x,l^{[x]})$, the value $l^{[x]}$ for group x grows by 1.

Whenever a new 0* is added to the matrix, the columns containing the 0*'s are covered and the remaining uncovered zeros are processed one by one in the same manner.

References

[1] Palmer D. Maintenance planning and scheduling handbook.2nd ed.. New York: McGraw Hill; 1999.
[2] Nyman D, Levitt J. Maintenance planning, scheduling and coordination.2nd ed.. New York: Industrial Press; 2010.
[3] Gopalakrishnan M, Ahire SL, Miller DM. Maximizing the effectiveness of a preventive maintenance system: an adaptive modeling approach. Management Science 1997;43:827–40.
[4] Cheng TCE, Ding Q, Lin BMT. A concise survey of scheduling with time-dependent processing times. European Journal of Operational Research 2004;152:1–13.
[5] Biskup D. A state-of-the-art review on scheduling with learning effects. European Journal of Operational Research 2008;188:315–29.
[6] Gordon VS, Potts CN, Strusevich VA, Whitehead JD. Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. Journal of Scheduling 2008;11:357–70.
[7] Gawiejnowicz S. Time-dependent scheduling. Berlin: Springer; 2008.
[8] Biskup D. Single-machine scheduling with learning considerations. European Journal of Operational Research 1999;115:173–8.
[9] Mosheiov G. Scheduling problems with a learning effect. European Journal of Operational Research 2001;132:687–93.
[10] Mosheiov G. A note on scheduling deteriorating jobs. Mathematical and Computer Modelling 2005;41:883–6.
[11] Bachman A, Janiak A. Scheduling jobs with position-dependent processing times. Journal of the Operational Research Society 2004;55:257–64.
[12] Lee C-Y. Machine scheduling with availability constraints. In: Leung JY-T, editor. Handbook of scheduling: algorithms, models and performance analysis. London: Chapman & Hall/CRC; 2004. p. 22-1–13.
[13] Ma Y, Chu C, Zuo C. A survey of scheduling with deterministic machine availability constraints. Computers and Industrial Engineering 2010;58:199–211.
[14] Qi X, Chen T, Tu F. Scheduling the maintenance on a single machine. Journal of the Operational Research Society 1999;50:1071–8.
[15] Liao CJ, Chen WJ. Single-machine scheduling with periodic maintenance and nonresumable jobs. Computers and Operations Research 2003;30:1335–47.

[16] Ji M, He Y, Cheng TCE. Single-machine scheduling with periodic maintenance to minimize makespan. Computers and Operations Research 2007;34:1764–70.
[17] Chen J-S. Optimization models for the tool change scheduling problem. Omega 2008;36:888–94.
[18] Chen W-J. Minimizing number of tardy jobs on a single machine subject to periodic maintenance. Omega 2009;37:591–9.
[19] Bar-Noy A, Bhatia R, Naor JS, Schiber B. Minimizing service and operation costs of periodic scheduling. Mathematics of Operations Research 2002;27:518–44.
[20] Grigoriev A, van de Klundert J, Spieksma FCR. Modelling and solving periodic maintenance problem. European Journal of Operational Research 2002;172:783–97.
[21] Kubzin MA, Strusevich VA. Two-machine flow shop no-wait scheduling with machine maintenance. Naval Research Logistics 2005;3:303–13.
[22] Kubzin MA, Strusevich VA. Planning machine maintenance in two-machine shop scheduling. Operations Research 2006;54:789–800.
[23] Lee C-Y, Leon VJ. Machine scheduling with a rate-modifying activity. European Journal of Operational Research 2001;128:119–28.
[24] Mosheiov G, Oron D. Due-date assignment and maintenance activity scheduling problem. Mathematics and Computer Modelling 2006;44:1053–7.
[25] Gordon VS, Tarasevich AA. A note: common due date assignment for a single machine scheduling with the rate-modifying activity. Computers & Operational Research 2009;36:325–8.
[26] Mosheiov G, Sarig A. Scheduling a maintenance activity and due-window assignment on a single machine. Computers & Operational Research 2009;36:2541–5.
[27] Mosheiov G, Sidney JB. Scheduling a deteriorating maintenance activity on a single machine. Journal of the Operational Research Society 2010;61:882–7.
[28] Kuo WH, Yang DL. Minimizing the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. Journal of the Operational Research Society 2008;59:416–20.
[29] Zhao C-L, Tang H-Y. Single machine scheduling with general job-dependent aging effect and maintenance activities to minimize makespan. Applied Mathematical Modelling 2010;34:837–41.
[30] Yang S-J, Yang D-L. Minimizing the makespan single-machine scheduling with aging effects and variable maintenance activities. Omega 2010;38:528–33.
[31] Lodree Jr EJ, Geiger CD. A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. European Journal of Operational Research 2010;201:644–8.
[32] Burkard R, Dell'Amico RM, Martello S. Assignment problems. Philadelphia: SIAM; 2009.
[33] Gordon VS, Strusevich VA. Single machine scheduling and due date assignment with positionally dependent processing times. European Journal of Operational Research 2009;198:57–62.
[34] Hardy GH, Littlewood JE, Polya G. Inequalities. London: Cambridge University Press; 1934.
[35] Mosheiov G, Sidney JB. Scheduling with general job-dependent learning curves. European Journal of Operational Research 2003;147:665–70.
[36] Rustogi K, Strusevich VA. Convex and *V*-shaped sequences of sums of functions that depend on ceiling functions. Journal of Integer Sequences 2011;14 Article 11.1.5 〈http://www.cs.uwaterloo.ca/journals/JIS/VOL14/Strusevich/strusevich2.html〉.
[37] Bourgeois F, Lassale J-C. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. Communications of ACM 1971;14:802–4.