University of Greenwich

Department of Mathematical Sciences

Statistics and Operational Research Group

Single Machine Scheduling with Time-Dependent Linear Deterioration and Rate-Modifying Maintenance

Kabir Rustogi

Vitaly A. Strusevich

Single Machine Scheduling with Time-Dependent Linear Deterioration and Rate-Modifying Maintenance

Abstract

We study single machine scheduling problems with linear time-dependent deterioration effects and maintenance activities. Maintenance periods (MPs) are included into the schedule, so that the machine, that gets worse during the processing, can be restored to a better state. We deal with a job-independent version of the deterioration effects, i.e., all jobs share a common deterioration rate. However, we introduce a novel extension to such models and allow the deterioration rates to change after every MP. We study several versions of this generalized problem and design a range of polynomial-time solution algorithms that enable the decision-maker to determine possible sequences of jobs and MPs in the schedule, so that the makespan objective can be minimized. We show that all problems reduce to a linear assignment problem with a product matrix and can be solved by methods very similar to those used for solving problems with positional effects.

Keywords: Scheduling; Sequencing; Maintenance; Deterioration; Time-dependent effects; Assignment problem

1 Introduction

Scheduling models in which the processing times are not constant but are subject to various effects have received considerable attention since the early 1990s. In most publications in this area, it is assumed that the processing conditions change over time and this affects the actual processing time of a job, which can get longer or shorter compared to its 'normal' processing time. To address this issue, two effects, known as deterioration and learning, have been studied. Under *deterioration*, the later a job starts, the longer it takes to process it; this effect is observed when the processing conditions get worse, i.e., a processing machine or a tool loses its qualities, or a human operator gets tired. Under *learning,* the later a job starts, the shorter it takes to process it; this effect occurs when human operators improve their skills by gaining experience.

Traditionally, researchers distinguish between several classes of effects that specify how exactly the actual processing time of a job is affected, depending on its place in a schedule. Most relevant to this paper is a *time-dependent* effect, in which the actual processing time of a job is a function of its start time. Another popular effect is a *positional* effect, in which the actual processing time of a job depends on its position in the processing sequence. Studies on time-dependent effects and positional effects have formed the bulk of publications in the area; see, e.g., a survey by Cheng, Ding and Lin (2004) and a book by Gawiejnowicz (2008), the latter providing a comprehensive exposition on scheduling with time-dependent processing times.

These two effects can be further classified as being either *job-dependent* or *job-independent.* A job-dependent effect implies that each job has a different (unique) effect on the state of the machine, while a job-independent effect implies the opposite.

Given a set $N = \{1, 2, \ldots, n\}$ of jobs, each job $j \in N$ is associated with a value p_j, which can be understood as its 'normal' processing time under perfect processing conditions. In the case of time-dependent effects, one of the most studied models is given by a linear function of the start-time of a job, so that the actual processing time $p_j(\tau)$ of a job $j \in N$ that starts at a time $\tau \geq 0$ is given by

$$p_j(\tau) = p_j + a_j \tau, \tag{1}$$

where a_j is a constant, which is strictly positive in the case of deterioration and strictly negative in the case of learning. In this paper, we mainly concentrate on effects that are derived from a job-independent version of (1) and are given by

$$p_j(\tau) = p_j + a\tau. \tag{2}$$

In the case of deterioration for the latter effect (2), the deterioration rate $a > 0$ is common for all jobs. Cheng, Ding and Lin (2004) mention that the above model provides a very realistic setting which reflects the fact that all processing times are increased by a common factor that mainly depends on the processing quality of the machine. A linear deterioration model arises in scheduling derusting operations performed by a single worker, in which not treated parts are subject to further corrosion, and therefore the time for their treatment would increase; see Gawiejnowicz et al. (2006). Additonal examples of practical use of linear time-dependent models can be found in the book by Gawiejnowicz (2008).

The difference between the time-dependent and the positional effects can be illustrated by the following example. In a manufacturing shop there are several parts that need a hole of the same diameter to be punched through by a pneumatic punching unit. A gas leakage after each punch inavodably occurs, due to which the punching unit loses pressure, so that the later a part is subject to punching the longer it takes to perform it. Here the position of a part in the queue, rather than its start time determines the actual time for punching.

This paper considers single machine scheduling models in which the processing times of the jobs are subject to deterioration effects similar to that given by (2). To avoid the growth of actual processing times to unacceptably large values, the decision-maker is allowed to perform maintenance activities in the schedule so that the processing conditions can be improved. Jobs scheduled after a *maintenance period* (MP) are processed faster, since the machine is restored to a better state. During an MP no jobs are processed on the machine.

A similar study on scheduling problems with positional deterioration effects and machine maintenance is conducted by Kuo and Yang (2008), Zhao and Tang (2010) and Rustogi and Strusevich (2012a). The latter paper not only handles an arbitrary positional deterioration effect, but also introduces the concept of group-dependence. In reality an MP will improve the processing conditions, but will not necessarily restore the initial perfect state of the machine. Rustogi and Strusevich (2012a) assume that the scheduled MPs are not identical, so that each of them leaves the processing conditions of the machine in a different state. Several MPs in a schedule split the jobs into several groups, one group before the first maintenance and one after each MP. Thus, the effects that change the actual processing time of a job may become additionally dependent on the group a job is sequenced in. Such effects are referred to as *group-dependent*. In this paper, we focus on single machine scheduling problems with start-time dependent effects similar to (2), provided that the deterioration rates are group-dependent.

In the problem that is of our primary concern in this paper, we are given K, the number of possible MPs, which is seen as a constant. The decision-maker determines how many of

	Linear Start-Time Dependent Group-Dependent Deterioration Effects	
	$\sum C_j$	$C_{\max}$
Group-Dependent Positional Effects	$O\left(n^{K+2}\right)$ Rustogi and Strusevich (2014)	$O\left(n^{K+1}\log n\right)$ Rustogi and Strusevich (2014)
No Positional Effects	$O\left(n^{K+1}\log n\right)$ Rustogi and Strusevich (2014)	$O\left(n2^{K}K\log K\right)$ This paper

Table 1: Running times of the algorithms for the relevant problems

K MPs to include into a schedule, in which order to perform them, and when to start each of them in order to minimize an objective function F of the job completion times. The jobs are subject to job-independent group-dependent deterioration effects, while the duration of an MP is either constant or also depends on its starting time. In this paper, we focus on $F = C_{\max}$, i.e., our goal is to minimize the *makespan* or the completion time of the last job to be processed. Another popular objective function is $F = \sum C_j$, the sum of the completion times or the *total flow time*. See Section 2 for a meaningful example of the problem studied in this paper. This example and its modifications are used throughout the paper to illustrate various developed algorithms. The example is formulated in terms of processing jobs on a machine by multiple cutting tools, but similar situations can be found in most manufacturing processes.

The first paper in which a time-dependent effect is combined with a single MP of a constant duration is Lodree and Geiger (2010). Yang and Yang (2010) study a problem of minimizing the total flow time for a linear time-dependent group-independent deterioration effect of the form (2) and a known number K of MPs. Yang (2010) and Yang (2012) address various versions of the problem, in which a linear time-dependent group-independent effect is combined with a polynomial positional effect. The algorithms employed to solve all such problems require the prior knowledge of the number of jobs in each group, and therefore, the resulting running times contain at least a multiple n^K, since the optimal number of jobs in each group is often found by full enumeration.

Rustogi and Strusevich (2012b, 2014) have recently shown that even models of the most general setting that combine simultaneous positional effects, time-dependent effects and various rate modifying activities (more general than just maintenance) can be solved in polynomial time. The corresponding scheduling problems in the case of job-independent effects reduce to a special form of the linear assignment problem with a product cost matrix. The resulting algorithms can be applied to handling simpler problems, with job-independent start-time effects only. However, as shown by Rustogi and Strusevich (2014), in the case of minimizing the total flow time $\sum C_j$ it is not possible to reduce the multiple n^K in the running time estimates, even if simpler problems, with start-time dependent effects only, are considered. For the problems in which is it required to determine how many of K available MPs should be inserted into a schedule and how to order the groups and the jobs within the groups, Table 1 demonstrates how the running times of known solution algorithms change for various effects and objective functions.

The main outcome of this paper is that it shows that various versions of the problem to minimize the makespan $C_{\max}$ with linear start-time dependent group-dependent deterioration effects, can be solved by algorithms, whose running times depend linearly on n, provided that K is a known constant and the sorted sequence of p_j's is available; see the bottom-right corner cell of Table 1. The improved running times are achieved because for the models under

Notation	**Description**
n	the number of jobs
p_j	'normal' processing time of job j
MP	a maintenance period
K	the number of available MPs
k	the number of groups created by using $k-1$ MPs in a schedule
τ	the start time of an MP or of a job
$a^{[x]}$ or a	the group-dependent/group-independent deterioration rate of jobs
$p_j^{[x]}(\tau) = p_j + a^{[x]}\tau$	the actual processing time of job j of group x, $1 \leq x \leq k$, that starts τ time units after the start of the group
$n^{[x]}$	the number of jobs in group x
$\pi^{[x]}$	the sequence of jobs in group x
$F_{(x,r)}$	the sum of actual processing times of the first r jobs in
$F_x = F_{\left(x,n^{[x]}\right)}$	the total processing time of the sequence $\pi^{[x]}$ of jobs
$T_x = \alpha^{[x]}F_x + \beta^{[x]}$	the duration of the $x-$th MP, where $\alpha^{[x]}$ and $\beta^{[x]}$ are known group-dependent constants
$S(k)$	a schedule with k groups
$C_{\max}(S(k))$	the makespan of schedule $S(k)$
$S^*(k)$	an optimal schedule with k groups
k^*	an optimal number of groups
$W^{[x]}(r)$	the positional weight of a job in position r of sequence $\pi^{[x]}$

Table 2: The list of notation

consideration we are able to determine the optimal number of jobs in each group without performing full enumeration. Moreover, under additional assumptions that are relevant to practice, even faster running times can be reached.

The remainder of this paper is organized as follows. Section 2 gives a formal statement of the problem under consideration and provides a brief review of the relevant results. The problem is reduced to the linear assignment problem with a product cost matrix that is defined by certain positional weights. Section 3 demonstrates how the required positional weights can be computed. A method for solving the problem in its general form is described and analyzed in Section 4. For several representative special versions of the problem, alternative faster methods are developed in Sections 5.1 and 5.2. Some concluding remarked can be found in Section 6.

2 Preliminaries

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. Each job $j \in N$ is associated with its normal processing time p_j. At time zero, the machine is assumed to be in a perfect processing state, and its conditions change with the course of processing. The machine can be subject to maintenance activities which will restore the processing conditions, either fully or at least partly. All these factors define the actual values of the processing time of each job j, that will depend on p_j, the group that the job is scheduled in, and on its start time within the group. The list of used pieces of notation in given in Table 2.

Given a feasible schedule S, the completion time of a job $j \in N$ is denoted by $C_j(S)$. In this paper, we mainly focus on the problems of minimizing the makespan, denoted by $C_{\max}(S) = \max\{C_j(S) | j \in N\}$, however, when appropriate we also review the results on minimizing the total flow time, denoted by $\sum_{j \in N} C_j(S)$. If it is clear from the content which schedule is used, we may drop the reference to S and simply write C_j, $C_{\max}$ or $\sum_{j \in N} C_j$, etc.

One of the main solution approaches widely used in the area of scheduling with changing processing times is based on reducing the original scheduling problem to a linear assignment problem with a product cost matrix (LAPr). Typically, an input for such a problem is determined by two arrays $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$. In such a problem, n items have to be assigned to n positions, the cost of assigning an item i to position j is given by $\lambda_i \mu_j$, and the total cost of the assignments is to be minimized. Provided that $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_n$, the problem consists in finding a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of the components of array α, such that for any permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, the inequality

$$\sum_{j=1}^{n} \lambda_{\varphi(j)} \mu_j \leq \sum_{j=1}^{n} \lambda_{\pi(j)} \mu_j \tag{3}$$

holds. The classical result by Hardy, Littlewood and Polya (1934) states that a permutation φ satisfies (3) if

$$\lambda_{\varphi(1)} \geq \lambda_{\varphi(2)} \geq \ldots \geq \lambda_{\varphi(n)},$$

provided that $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_n$. As a result, Problem LAPr can be solved by the following *matching algorithm*: permutation φ matches the larger components of one of the two given arrays with smaller components of the other array. The matching algorithm requires $O(n \log n)$ time. This algorithm is quite well-known and has become a popular technique for solving scheduling problems in this area; see, e.g., Gawiejnowicz (1996), Kuo and Yang (2007), Okołowski and Gawiejnowicz (2010) and Yang and Yang (2010). Also, see a survey by Rustogi and Strusevich (2012b), which demonstrates that many scheduling problems with changing processing times can be solved by the matching algorithm, but instead less suitable approaches were used in earlier papers.

In the past, although many papers have considered the problem of time-dependent deterioration with maintenance periods, most of them however, only see an MP as a fixed non-availability period, which does not necessarily improve the machine conditions. Only a handful of studies have considered problems in which an MP is actually used to restore the machine to its original state, so that the effects of time-dependent deterioration are repaired.

Lodree and Geiger (2010) study a problem of minimizing the makespan for a time-dependent effect of the form $p_j(\tau) = a_j \tau, a_j \geq 1$, and a single MP in the schedule. They provide an optimal policy to determine the number of jobs to be included in each of the two created groups. According to this policy, if n is even, both groups should contain $\left(\frac{n}{2} + 1\right)$ jobs, whereas if n is odd, one group should contain $\left(\frac{n+1}{2}\right)$ jobs and the other should contain $\left(\frac{n+3}{2}\right)$ jobs.

Yang and Yang (2010) study a problem of minimizing the total flow time for a linear time-dependent deterioration effect of the form (2) and a known number K of MPs. They prove that if the number of jobs in each group is known, the problem reduces to Problem LAPr, so that an optimal permutation of jobs can be found in $O(n \log n)$ time. Further, they enumerate all possible options in which n jobs can be divided into $K + 1$ groups, and provide an optimal solution to the problem in $O\left(n^{K+1} \log n\right)$ time.

The same method is extended by Yang (2010) and Yang (2012) to handle various versions of a problem, in which a linear time-dependent effect is combined with a polynomial positional effect, so that a combined effect of the form $p_j(\tau, r) = (p_j + a\tau) r^b, 1 \leq r \leq n$, is applied, where τ denotes the start time of job j, while r corresponds to its position in the processing sequence. Yang (2010) considers the case, in which a single MP ($K = 1$) is to be scheduled on a single machine that is subject to time-dependent learning ($a < 0$) and polynomial positional deterioration ($b > 0$) effects. Yang (2012) considers the case, in which K MPs are to be scheduled on a single machine that is subject to time-dependent deterioration ($a > 0$) and polynomial positional learning ($b < 0$) effects. For their respective models, both these papers claim to solve the problem of minimizing the makespan and the problem of minimizing the total flow time in $O\left(n^{K+1} \log n\right)$ time each. However, Rustogi and Strusevich (2014) show that both these papers underestimate the running time needed to solve the problem of minimizing the total flow time and calculate the running time to be $O\left(n^{K+2}\right)$ for both cases. Rustogi and Strusevich (2014) also show that the running times of $O\left(n^{K+1} \log n\right)$ and $O\left(n^{K+2}\right)$ are applied to a range of much more general problems with group dependent combined effects.

Notice that apart from the problem studied by Lodree and Geiger (2010), all other problems mentioned so far in this section require a full enumeration of the number of jobs in each group. To the best of our knowledge, there are no other algorithms which solve the problems related to time-dependent effects and maintenance activities in a different way. In particular, prior to this paper it has remained unknown whether the problem with a pure time-dependent deterioration effect and maintenance activities can be solved faster than the corresponding problem with a combined effect of the form $p_j(\tau, r) = (p_j + a\tau) r^b$.

In this paper, we solve the problem of minimizing the makespan for a pure linear time-dependent deterioration effect given by (2), which is enhanced by including various maintenance activities in the schedule. We show that even in the presence of distinct maintenance activities and group-dependence, the problem of minimizing the makespan can be solved by an efficient polynomial algorithm, which does not involve full enumeration.

Formally, the problem that we consider in this paper is as follows. A decision-maker is presented with a total of $K \geq 0$ possible maintenance activities, which can be either distinct or alike. We further generalize the effect (2) by allowing each MP to restore the machine to a different state, so that different deterioration rates are applied in different groups. The MPs are indexed arbitrarily. As part of the input, we know $a^{[1]}$, the deterioration rate applied to the group that starts at time zero, before the first scheduled MP. Also, for the MP with an index y, $1 \leq y \leq K$, we know $a^{[y+1]}$, a deterioration rate that would apply to the jobs sequenced in the group that follows the MP, provided it is included into a schedule.

Suppose that $k - 1$ MPs have been chosen from the available K, and a certain order to perform these maintenance activities has been fixed. In the resulting schedule the jobs are split into k groups, and we call such a schedule $S(k)$, $1 \leq k \leq K + 1$. In schedule $S(k)$, the actual processing time of a job $j \in N$ that starts at time τ in the $x-$th group is given by

$$p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\ \tau \geq 0,\ 1 \leq x \leq k, \tag{4}$$

where it is assumed that the timer is reset after every MP. The deterioration rates are renumbered in order of the occurrence of the corresponding MPs in the schedule, so that a group sequenced after the $x-$th MP is associated with a deterioration rate $a^{[x+1]} > 0$, $1 \leq x \leq k - 1$. Notice that these group-dependent deterioration rates are analogous to the group-dependent positional factors introduced by Rustogi and Strusevich (2012a).

Recall that during each MP no job processing takes place. In a very general setting, each of the K available MPs, are associated with its own set of duration parameters, $\alpha^{[y]}$ and $\beta^{[y]}$, $1 \le y \le K$. This allows us to further generalize the linear model for start-time dependent MP durations introduced by Kubzin and Strusevich (2005, 2006), so that the duration D_{MP} of an MP can be given as

$$D_{MP} = \alpha^{[y]}\tau + \beta^{[y]}, \tag{5}$$

where τ is the start-time of the MP, measured from the time the previous MP was completed, and $\alpha^{[y]}$ and $\beta^{[y]}$ are parameters that define the MP.

Extending the traditional three-field notation widely accepted in scheduling literature, we denote the problem of minimizing the makespan under the general settings defined by (4) and (5) by $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$. An optimal solution to problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ must deliver optimal choices for each of the following decisions:

Decision 1. *The number of MPs:* If $k-1$ MPs are included in the schedule, the jobs are divided into k groups. Determine the optimal value of k, $1 \le k \le K+1$.

Decision 2. *The choice of MPs:* From a given list of K MPs, choose $k-1$, $1 \le k \le K+1$, MPs to be included in the schedule.

Decision 3. *The sequence of MPs:* Determine the optimal order in which the selected $k-1$ MPs are scheduled on a machine.

Decision 4. *An optimal permutation of jobs:* For each job $j \in N$, determine the group and the position within that group, that it must be scheduled in.

Example 1. Six jobs should be processed by a human operator who has an equipment with multiple cutting tools. The processing times (in appropriate time units) of these jobs under perfect conditions of both the operator and the tool are given by

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

During the processing, the operator gets tired and the tool loses its sharpness, which leads to extending the actual processing time, i.e., to a deterioration effect. During the planning period, up to $K = 5$ types of maintenance activities can be performed. Some can be seen as rest periods for the operator (leaving the tools' conditions unchanged), some of them are related to improving the cutting capabilities of one or several tools (with the operator having rest as well). In any case, the duration of the $x-$th MP is defined in accordance with (5) and may include a constant term $\beta^{[x]}$, which can be seen as the duration of mandatory standard tests to be run for the maintenance activity of this type, as well as the start-time dependent term $\alpha^{[x]}\tau$, which is understood as the duration of repair activities that depends on the conditions of the tool(s). The rates $\alpha^{[x]}$ are different for different types of maintenance activities, since they involve work on different tools. Any MP improves the processing conditions, but not to initial perfect conditions, so that a different deterioration rate might be in effect after each MP. As a result, the jobs are split into several groups, one before the first MP and one after each scheduled MP. The actual processing times of the jobs follow the rule (4). For a group x, the job deterioration rate $a^{[x]}$ is set by the preceding $(x-1)-$th MP. The default deterioration rate of the machine is given as $a^{[1]} = 0.1$. The other numerical parameters are given by

$$\begin{array}{llll}
\text{Type 1 MP}: & \alpha^{[1]} = 0.05, & \beta^{[1]} = 10, & a^{[2]} = 0.15; \\
\text{Type 2 MP}: & \alpha^{[2]} = 0.10, & \beta^{[2]} = 8, & a^{[3]} = 0.20; \\
\text{Type 3 MP}: & \alpha^{[3]} = 0.025, & \beta^{[3]} = 6, & a^{[4]} = 0.25; \\
\text{Type 4 MP}: & \alpha^{[4]} = 0.15, & \beta^{[4]} = 2, & a^{[5]} = 0.20; \\
\text{Type 5 MP}: & \alpha^{[5]} = 0.2, & \beta^{[5]} = 0, & a^{[6]} = 0.15.
\end{array}$$

In the setting of this example, Decision 1 is to choose k types of maintenance activities to be included into the schedule, e.g., 3 out of 5. Then Decision 2 selects particular types of maintenance, e.g., types $1, 3$ and 4. Decision 3 determines the sequence in which the three chosen MPs are included into the schedule, i.e., type 3 followed by type 4 and then type 1. In the resulting schedule there will be 4 groups, with the job deterioration rates equal to $0.1, 0.25, 0.20$ and 0.15, respectively.

Given an instance of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, suppose that all four decisions are taken (not necessarily in an optimal way). This defines a schedule $S(k)$. The sequence of jobs in schedule $S(k)$ can be written as $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, where $\pi^{[x]}$ denotes a sequence of jobs in group x, $1 \leq x \leq k$. In turn, if a group x contains $n^{[x]}$ jobs, then the permutation of these jobs is given by $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$.

In what follows, we consider various versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ and provide polynomial-time algorithms for their solution. We distinguish between different versions of the problem based on three criteria: (i) deterioration rates are group-dependent or group-independent, (ii) MPs are identical or distinct, and (iii) duration of the MPs are constant or start-time dependent.

All versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ reduce to finding a permutation of jobs that delivers the minimum to a generic objective function of the form

$$F(k) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{6}$$

where $W^{[x]}(r)$ is a suitably defined positional weight and $\Gamma(k)$ is a sequence independent constant. The problem of minimizing function $F(k)$ can be qualified as Problem LAPr and can be solved by the matching algorithm.

3 Computing Positional Weights

To solve problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, we first assume that Decisions 1-3 are taken in advance, so that we know that some $k-1$ particular MPs are included into the schedule, in a chosen order. As a result, the jobs are split into k, $1 \leq k \leq K+1$, groups. Renumber the indices of duration parameters of the MPs in the order of their occurrence in the schedule, so that the duration of an MP scheduled after the $x-$th group is given by

$$T_x = \alpha^{[x]} F_x + \beta^{[x]},\ 1 \leq x \leq k-1, \tag{7}$$

where F_x is the total processing time of all jobs scheduled in the $x-$th group. Denote the resulting problem by $1\left|p_j + a^{[x]}\tau, MP(k-1)\right| C_{\max}$. In order to solve this problem we need to find out how to split the jobs into k groups and how to sequence them within each group; essentially, this corresponds to the best Decision 4 based on earlier taken Decisions 1-3.

Consider a schedule $S(k)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \dots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$. We now derive an expression for the total time F_x it takes to process all jobs in a group x, $1 \le x \le k$.

Let us denote the total duration of the first r jobs in a group x by $F_{(x,r)}$. It follows from (4) that $F_{(x,1)} = p_{\pi^{[x]}(1)}$. The actual processing time of the second job in the group is given by $p_{\pi^{[x]}(2)} + a^{[x]} F_{(x,1)}$, which implies that

$$F_{(x,2)} = F_{(x,1)} + \left(p_{\pi^{[x]}(2)} + a^{[x]} F_{(x,1)}\right) = \left(1 + a^{[x]}\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)}.$$

Similarly, for the third job in the actual processing time is given by $p_{\pi^{[x]}(3)} + a^{[x]} F_{(x,2)}$, so that

$$\begin{aligned} F_{(x,3)} &= F_{(x,2)} + \left(p_{\pi^{[x]}(3)} + a^{[x]} F_{(x,2)}\right) = \left(1 + a^{[x]}\right)\left(\left(1 + a^{[x]}\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)}\right) + p_{\pi^{[x]}(3)} \\ &= \left(1 + a^{[x]}\right)^2 p_{\pi^{[x]}(1)} + \left(1 + a^{[x]}\right) p_{\pi^{[x]}(2)} + p_{\pi^{[x]}(3)}. \end{aligned}$$

Extending, we deduce that for the jobs in the first r positions in group x, the formula

$$F_{(x,r)} = \sum_{u=1}^{r} \left(1 + a^{[x]}\right)^{r-u} p_{\pi^{[x]}(u)}, \ 1 \le r \le n^{[x]}, 1 \le x \le k.$$

Thus, the total time it takes to process all jobs in a group x can be given by

$$F_x = F_{\left(x, n^{[x]}\right)} = \sum_{r=1}^{n^{[x]}} \left(1 + a^{[x]}\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)}, \ 1 \le x \le k. \tag{8}$$

The makespan of a schedule $S(k)$ is the sum of the durations of all scheduled groups and the MPs, and is given by

$$C_{\max}(S(k)) = F_1 + T_1 + F_2 + T_2 + \dots + F_{k-1} + T_{k-1} + F_k,$$

where T_x is the duration of the MP scheduled after the $x-$th group. Substituting the value of T_x from (7) into the above equation we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1} (1 + \alpha^{[x]}) F_x + F_k + \sum_{x=1}^{k-1} \beta^{[x]}.$$

Now, substituting the value of F_x from (8) we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1} \sum_{r=1}^{n^{[x]}} (1 + \alpha^{[x]}) \left(1 + a^{[x]}\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}} \left(1 + a^{[k]}\right)^{n^{[k]}-r} p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1} \beta^{[x]}. \tag{9}$$

We see that in (9), a job j scheduled in position r of group x, $1 \le x \le k-1$, $1 \le r \le n^{[x]}$, will contribute its normal processing time $p_j = p_{\pi^{[x]}(r)}$ taken $(1 + \alpha^{[x]}) \left(1 + a^{[x]}\right)^{n^{[x]}-r}$ times.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-independent	Section 5.2	Section 5.2	Section 5.1	Section 4
Group-dependent	Section 5.1	Section 4	Section 4	Section 4

Table 3: Different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$

If job j is scheduled in the last group $x = k$, then its contribution is $\left(1 + a^{[k]}\right)^{n^{[k]}-r}$ times p_j. This implies that the objective function (9) can be written as a generic function (6), with the *positional weights*

$$W^{[x]}(r) = \begin{cases} \left(1+\alpha^{[x]}\right)\left(1+a^{[x]}\right)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ \left(1+a^{[k]}\right)^{n^{[k]}-r}, & 1 \le r \le n^{[x]},\ x = k, \end{cases} \tag{10}$$

and the constant term given by

$$\Gamma\left(k\right) = \sum_{x=1}^{k-1} \beta^{[x]}. \tag{11}$$

Thus, problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$ reduces to minimizing a linear form $\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}(r)p_{\pi^{[x]}(r)}$ over a set of all permutations. If the number of jobs in each group, $n^{[x]}$, $1 \le x \le k$, is known, an optimal solution can be obtained by the matching algorithm. To obtain a solution to the original problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, we (i) generate all possible instances of problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$ by modifying the outcomes of Decisions 1-3, (ii) solve each of the obtained instances, i.e., the corresponding problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$ and (iii) identify the instance with the smallest value of the objective function as an overall optimal solution.

In what follows, we consider eight different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$. For each version, the computed positional weights are found by making an appropriate adjustment in (10). Recall that (10) is computed for the most general version, in which the deterioration rates are group-dependent, MPs are distinct and their durations depend on the start-time. Notice that the positional weights defined by (10) are non-increasing within each group. Additionally, their values are dependent on the number of jobs $n^{[x]}$ in a group.

In the following sections, we present three solution approaches, which handle different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ by finding the best options for Decisions 1-4. The main advantage of these approaches is that the number of jobs, $n^{[x]}$, in each group x, $1 \le x \le k$, is not found by full enumeration but on the fly, which considerably reduces the running times of the resulting algorithms. Table 3 lists the different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ considered in this paper and gives references to the appropriate sections.

4 Solution Approach to the General Problem

In this section, we describe a solution approach which solves problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ in the most general setting, i.e., when the deterioration rates are group-dependent and the K MPs are known to be distinct with start-time dependent durations. Unlike many other problems in the area of scheduling with changing processing times as discussed in Section 3, the approach that we employ here does not require prior knowledge of the number of jobs $n^{[x]}$ in each group, and leads to a solution algorithm whose running time depends linearly on n, for a constant K.

Assume that Decisions 1-3 are taken in advance and denote the resulting problem as $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$. Set the value $n^{[x]} = n$, $1 \le x \le k$, and compute all possible positional weights $W^{[x]}\left(r\right)$, $1 \le r \le n$, $1 \le x \le k$, by (10), where for convenience we denote $U^{[x]} := \left(1+\alpha^{[x]}\right)$, $1 \le x \le k-1$, and $U^{[k]} := 1$. The resulting $n \times k$ matrix is given by

$$\begin{pmatrix} U^{[1]}\left(1+a^{[1]}\right)^{n-1} & U^{[2]}\left(1+a^{[2]}\right)^{n-1} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-1} \\ U^{[1]}\left(1+a^{[1]}\right)^{n-2} & U^{[2]}\left(1+a^{[2]}\right)^{n-2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ U^{[1]}\left(1+a^{[1]}\right)^{2} & U^{[2]}\left(1+a^{[2]}\right)^{2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{2} \\ U^{[1]}\left(1+a^{[1]}\right) & U^{[2]}\left(1+a^{[2]}\right) & \cdots & U^{[k]}\left(1+a^{[k]}\right) \\ U^{[1]} & U^{[2]} & \cdots & U^{[k]} \end{pmatrix}. \tag{12}$$

Each column of the above matrix represents all possible positional weights that can be associated with a particular group, the first element of column x representing a weight associated with the first position of group x, while the last element of column x, representing a weight associated with the last, i.e., the $n-$th position of group x, $1 \le x \le k$. The elements of each column form a non-increasing sequence of the weights, i.e., for problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$ we have that

$$W^{[x]}(1) \ge W^{[x]}(2) \ge \cdots \ge W^{[x]}(n^{[x]}),$$

provided that there are $n^{[x]}$ jobs in group x, $1 \le x \le k$.

The following statement explains how problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$ can be solved, without prior knowledge of the values $n^{[x]}$, $1 \le x \le k$.

Theorem 1 *Given problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$, where $1 \le k \le K+1$, compute the matrix (12) of all possible positional weights $W^{[x]}\left(r\right)$ and choose the n smallest among them. If in each column the chosen elements occupy consecutive positions starting from the last row, then assigning the jobs with the largest normal processing times to the positions associated with the smallest positional weights will ensure that the objective function (6) is minimized.*

Notice that Theorem 1 is only applicable to solving those scheduling problems with changing processing times, for which all possible positional weights can be computed in advance, which is not the case, e.g., for problems of minimizing the total flow time as in Yang and Yang (2010) or for problems with combined effects as in Yang (2010, 2012) and Rustogi and Strusevich (2014).

The proof of Theorem 1 is straightforward. Notice that at most n positions can be used in each group. In a schedule with k groups, we have a choice of at most nk positions, in which n jobs can be potentially scheduled. Each of these positions have a certain positional weight associated with them. Recall that the contribution of a job $j = \pi^{[x]}\left(r\right)$ to the objective function $F\left(k\right)$ is given by $W^{[x]}\left(r\right)p_j$. Thus, in order to ensure the smallest value of the objective function we must choose n positions that correspond to the smallest positional weights. Since all possible values of $W^{[x]}(r)$ are known, we can identify these positions. If within a group the found positions form a block of consecutive positions starting from position 1, then they may be used to create a feasible schedule. For a group x, the number of positions in such a block gives us the value $n^{[x]}$, $1 \leq x \leq k$.

Finally, to assign the jobs optimally to the chosen positions we use the matching algorithm. The easiest way to implement it, is to create a non-decreasing list of the chosen positional weights and an *LPT (Longest Processing Time)* list of jobs taken in non-increasing sequence of their normal processing times, and to match the $u-$th elements of the two lists, $1 \leq u \leq n$. A similar idea is used to design Algorithm 2 presented in Rustogi and Strusevich (2012a). We now apply this approach to finding an optimal solution to problem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right| C_{\max}$.

According to Theorem 1, an optimal schedule can be found by choosing the n smallest positional weights and assigning the jobs with the largest processing times to the positions corresponding to the smallest positional weights. Notice that the n smallest positional weights are found in consecutive positions of the columns at the bottom of the matrix (12). The smallest positional weight in a group is associated with the last position in that group, irrespective of the number of jobs in that group, i.e., for group x, in accordance with (10) the weight $W^{[x]}\left(n^{[x]}\right) = U^{[x]}$. The next smallest positional weight in group x is located immediately above in the same column, and so on.

The problem of finding the n smallest positional weights and matching them to the appropriate jobs is structurally similar to that of scheduling jobs on uniform parallel machines to minimize the total flow time. The latter problem can be solved by a method described by Conway, Maxwell and Miller (1967); see the book by Brucker (2007) for a good implementation. According to this method, the jobs are scanned in the LPT order and the machines are filled in the reversed order, from the last position to the first one.

Adapting this approach to our problem, consider the jobs in the LPT order. To assign the first job, compare the k multipliers $U^{[x]}$, $1 \leq x \leq k$, and assign the job to the last position of the group associated with the smallest value of $U^{[x]}$, $1 \leq x \leq k$. The next positional weight that can be taken from this group is computed and replaces the previously used one. The process continues, and for the current job the smallest of the k available positional weights determines the group and the position within the group, where the job should be assigned. This approach does not require any advance knowledge of the number of jobs $n^{[x]}$ in each group, or in fact, even an advance knowledge of the full matrix (12).

A formal description of the algorithm is given below.

Algorithm NSmall

INPUT: An instance of problem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right| C_{\max}$ with the jobs renumbered in the LPT order

OUTPUT: An optimal schedule $S^*(k)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k$

Step 1. For each group x, $1 \leq x \leq k$, define an empty processing sequence $\pi^{[x]} := (\varnothing)$ and the weight $W^{[x]} = U^{[x]}$. Create a non-decreasing list Ω of the values $W^{[x]}$, $1 \leq x \leq k$; to break ties, we place the weight associated with a group with a smaller index x earlier in the list.

Step 2. For each job j from 1 to n do

(a) Take the first element $W^{[v]}$ in list Ω, the smallest available positional weight.

j	p_j	Group 1		Group 2		Group 3		Group 4		Contribution
		$W^{[1]}$	$\pi^{[1]}$	$W^{[2]}$	$\pi^{[2]}$	$W^{[3]}$	$\pi^{[2]}$	$W^{[4]}$	$\pi^{[4]}$	of job j
		1.025	$\varnothing$	1.15	$\varnothing$	1.05	$\varnothing$	$\boxed{1}$	$\varnothing$	
1	10	$\boxed{1.025}$	$\varnothing$	1.15	$\varnothing$	1.05	$\varnothing$	$1 * 1.15$	(1)	$1 * 10$
2	9	$1.025 * 1.1$	(2)	1.15	$\varnothing$	$\boxed{1.05}$	$\varnothing$	1.15	(1)	$1.025 * 9$
3	6	$\boxed{1.1275}$	(2)	1.15	$\varnothing$	$1.05 * 1.2$	(3)	1.15	(1)	$1.05 * 6$
4	3	$1.1275 * 1.1$	$(4, 2)$	$\boxed{1.15}$	$\varnothing$	1.26	(3)	1.15	(1)	$1.1275 * 3$
5	3	1.24025	$(4, 2)$	$1.15 * 1.25$	(5)	1.26	(3)	$\boxed{1.15}$	(1)	$1.15 * 3$
6	2	1.24025	$(4, 2)$	1.4375	(5)	1.26	(3)	$1.15 * 1.15$	$(6, 1)$	$1.15 * 2$

Table 4: Run of Algorithm NSmall for Example 1 with the chosen sequence of MPs

(b) Assign job j to group v and place it in front of the current permutation $\pi^{[v]}$, i.e., update $\pi^{[v]} := (j, \pi^{[v]})$ and associate job j with the positional weight $W^{[v]}$. Remove $W^{[v]}$ from the list Ω. Update $W^{[v]} := W^{[v]}\left(1 + a^{[v]}\right)$ and insert the updated value $W^{[v]}$ into Ω, while maintaining the list Ω as non-decreasing.

Step 3. With the found permutation $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k]}\right)$, compute the optimal value of the objective function $C_{\max}\left(S^*\left(k\right)\right)$ by substituting appropriate values in (6).

In Step 1 of Algorithm NSmall, list Ω can be created in $O\left(k \log k\right)$ time. Each iteration of the loop in Step 2 requires $O\left(\log k\right)$ time, since the insertion of the updated weight into a sorted list can be done by binary search. Since $n \geq k$, the following statement holds.

Theorem 2 *Algorithm NSmall solves problem* $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$ *in* $O\left(n \log k\right)$ *time, provided that the LPT order of the jobs is known.*

Example 2. We illustrate Algorithm NSmall by solving the instance described in Example 1, provided that three MPs have been chosen to run in the order Type 3, Type 4 and Type 1. The parameters ($\alpha^{[x]}$, $\beta^{[x]}$ and $a^{[x]}$) are renumbered according to the order in which the groups appear in the schedule. After renumbering the parameters become

$$\begin{array}{lll} & & a^{[1]} = 0.10; \\ \alpha^{[1]} = 0.025, & \beta^{[1]} = 6, & a^{[2]} = 0.25; \\ \alpha^{[2]} = 0.150, & \beta^{[2]} = 2, & a^{[3]} = 0.20; \\ \alpha^{[3]} = 0.050, & \beta^{[3]} = 10, & a^{[3]} = 0.15. \end{array}$$

The computation is shown in Table 4. For each job the chosen positional weight is shown in a box in the previous row.

In the resulting schedule, the sequence of jobs $(4, 2)$ form Group 1, while Group 2 and Group 3 consist of job 5 and 3, respectively, and the last Group 4 processes the sequence of jobs $(6, 1)$. The makespan of this schedule can be computed by (6), where $\Gamma(3)$ is equal to $\beta^{[1]} + \beta^{[2]} + \beta^{[3]} = 18$, so that the resulting makespan is equal to $34.6575 + 18 = 52.6575$.

To determine an optimal solution for the general problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, all options associated with Decisions 1-3 must be enumerated and the solutions of the resulting sub-problems $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$ be compared. The best of these solutions is

	$K=3$	$K=5$	$K=7$	$K=9$	$K=11$
The number of generated subproblems	13	81	449	2305	11265
$n=100$					
$n=500$					
$n=1000$					
$n=5000$					
$n=10000$					

Table 5: The number of subporblems and computaion times (in sec)

chosen as an overall optimal solution for problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. For a known k, $1 \leq k \leq K+1$, the number of ways to select $k-1$ MPs from K available MPs (Decision 2) is equal to $\binom{K}{k-1}$. Notice that the positional weights (10) that are associated with the first $k-1$ groups, do not depend on the order of the MPs. However, the $k-$th group is differently structured from the others, since for this group $U^{[k]} = 1$, so it matters which MP is to be scheduled last, i.e., at the $(k-1)-$th position. Thus, the number of choices for Decision 3 is equal to $k-1$. Trying all possible values of k (Decision 1), $1 \leq k \leq K+1$, the total number of options to be evaluated is given by $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)$. Since Algorithm NSmall requires $O(n\log k)$ time to run for a given k, $1 \leq k \leq K+1$, the total running time required to solve the most general version of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ can be estimated as $O\left(\sum_{k=1}^{K+1} n\log k\binom{K}{k-1}(k-1)\right)$. Since $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1) = O\left(2^K K\right)$, the resulting running time can be written as $O(n2^K K\log K)$, which is linear in n for a constant K.

Notice that in an optimal solution for an instance of problem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right|C_{\max}$ it is possible that out of the k groups, certain groups are not assigned any jobs at all, i.e., $n^{[x]} = 0$. Such a situation can occur if an MP is not efficient in restoring the machine to a better state, and as a result the group that follows would generate rather large positional weights. Such an instance can never result in an overall optimal schedule for the general problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, since we unnecessarily spend time to perform an inefficient MP. This instance, if any, will be automatically eliminated from consideration if we try different combinations of Decision 1-3 to define problem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right|C_{\max}$.

We have performed computational experiments with problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, which demonstrate that our method solves the problem in an acceptable time for reasonable vaues of K, provided that Algorithm NSmall is used to solve each generated subpriblem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right|C_{\max}$.

In our settings, we have taken $n \in \{100, 500, 1000, 5000, 10000\}$ and $K \in \{3, 5, 7, 9, 11\}$. Notice that in reality K is the number of possible types of maintenance, and is therefore facility-dependent, i.e., it is unlikely to be very large and is not affected by the number of jobs.

For each pair (n, K) the input parameters have been sampled from the corresponding uniform distributions: p_j from $(0, 10]$, $\alpha^{[y]}$ from $[0, 0.25]$, $\beta^{[y]}$ from $[0, 10]$, $a^{[1]}$ from $[0, 0.2]$, while the $a^{[y+1]}$ values have been simulated as the sum of $a^{[1]}$ and a random number sampled from $[0, 0.25]$. A total of 10 instances have been generated for each pair (n, K). The algorithms has been coded in MATLAB 13.8 and run on XXXX. Table 5 reports average computation times achived for each pair (n, K).

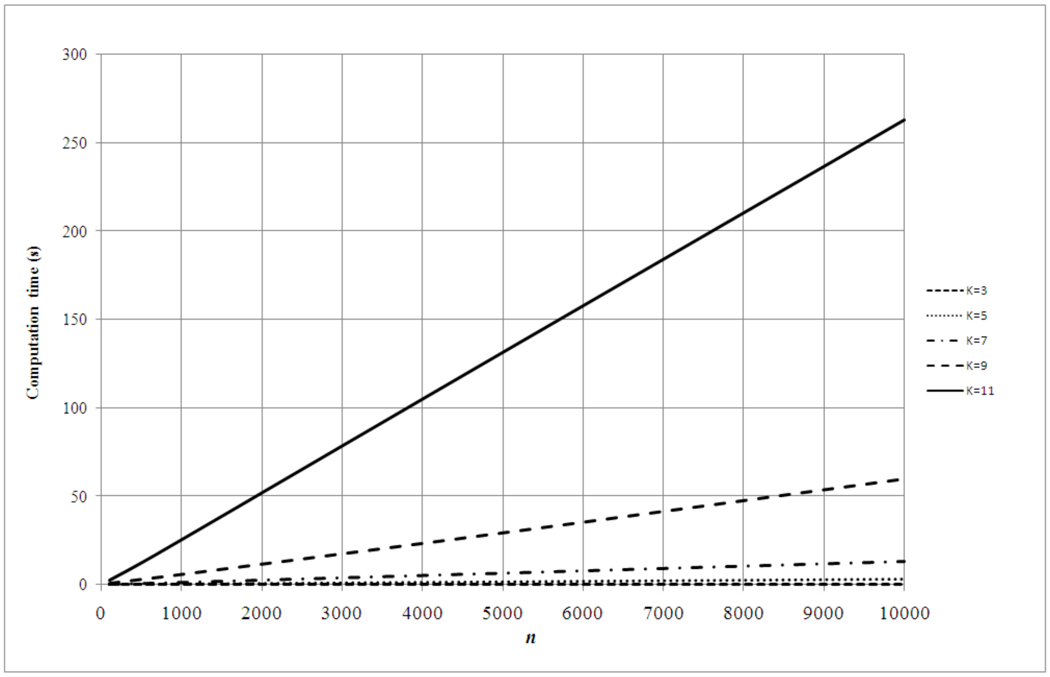

Figure 1: The computation times (in sec) as functions of n

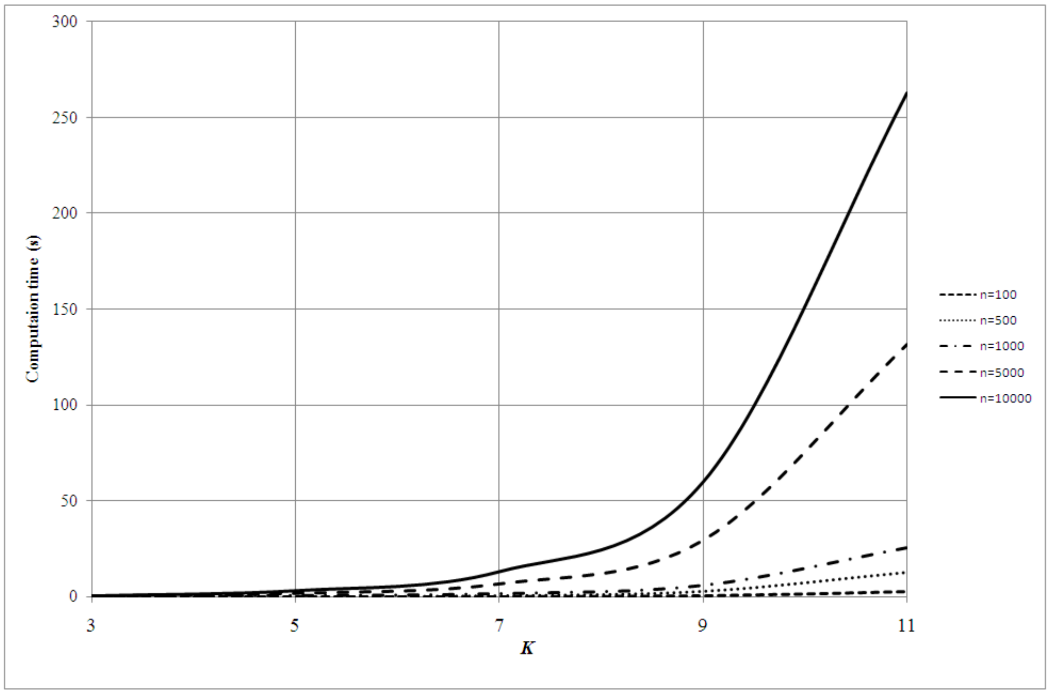

Figure 2: The computation times (in sec) as functions of K

Figures 1 and 2 plot the average computaion times as fucntions of n and K, respectively. We see that the times in Figure 1 increase linearly with n and those in Figure 2 increase exponentially in K, which complies with the running time estimation $O(n2^K K\log K)$.

In the remainder of this paper, we consider special cases of the general problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with a purpose of verifying whether they would admit faster solution algorithms.

First, consider a version of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with group-independent deterioration rates, i.e., $a^{[x]} = a$, $1 \le x \le k$, and distinct MPs of start-time dependent durations. This problem corresponds to a scenario in which distinct MPs are performed in the schedule, but they all restore the machine to the same state. Making relevant substitutions in (10) the computed positional weights can be written as

$$W^{[x]}(r) = \begin{cases} \left(1+\alpha^{[x]}\right)(1+a)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ (1+a)^{n^{[k]}-r}, & 1 \le r \le n^{[x]},\ x = k, \end{cases}$$

whereas the constant term $\Gamma(k)$ remains as given in (11). Since each MP restores the machine to the same state, Decision 2 is made only on the basis of the durations of the MPs. There are two parameters that define the MP durations, therefore, there is no easy way by which the best $k-1$ MPs can be selected out of the available K MPs. Thus, all possible selections need to be tried and this can be done in $\binom{K}{k-1}$ ways. Moreover, the found positional weights appear to be independent of the order of the MPs. Thus, Decision 3 is irrelevant. Trying all possible values of k (Decision 1), $1 \le k \le K+1$, the total number of options can be given by $\sum_{k=1}^{K+1}\binom{K}{k-1}$. Thus, the total running time required to solve this version of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ can be estimated as $O\left(\sum_{k=1}^{K+1} n\binom{K}{k-1}\log k\right) = O\left(n2^K \log K\right)$, a factor of K less than in the general case.

For the problem with group-dependent deterioration rates and distinct MPs of constant durations, i.e., $\alpha^{[x]} = 0$, $1 \le x \le k-1$, the computed positional weights can be written as

$$W^{[x]}(r) = \left(1 + a^{[x]}\right)^{n^{[x]}-r},\ 1 \le r \le n^{[x]},\ 1 \le x \le k.$$

Clearly, these positional weights are dependent on the type of the MP, but not on their order. Thus, as above the total number of options can be given by $\sum_{k=1}^{K+1}\binom{K}{k-1}$ and total running time required to solve this version of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ can be estimated as $O\left(n2^K\log K\right)$.

Consider now the problem with group-dependent deterioration rates and identical MPs of start-time dependent durations, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \le x \le k-1$. This problem corresponds to a scenario in which identical start-time dependent MPs are performed in the schedule, but they all restore the machine to a different state. Such a situation is possible when the MPs fail to repair the machine to the default standard. Since the MPs are identical, clearly Decision 2 and 3 do not have an effect on the optimal solution. Thus, only Decision 1 needs to be taken. Trying all possible values of k, $1 \le k \le K+1$, an optimal solution to this version of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ can be found in $O\left(\sum_{k=1}^{K+1} n\log k\right) = O\left(nK\log K\right)$ time.

Notice that although Algorithm NSmall is able to solve all eight versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ listed in Table 3, for some cases it is possible to make Decisions 1-3 on the fly by using another solution approach, which enables us to find an optimal solution faster. These alternative solution approaches are discussed in the following section.

5 Faster Solution Approaches for Special Cases

In this section, we distinguish between two solution approaches that are applicable to certain special cases of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, and lead to faster solution algorithms.

5.1 On the Fly Decision Making

The solution approach outlined in this subsection allows us to make Decisions 1-4 on the fly, without enumerating all possible options. This helps us to considerably reduce the running time for solving special cases of the original problem. We list out two versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ for which this is possible:

Problem 1: Problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ with group-dependent deterioration rates and distinct MPs of constant durations, i.e., $\alpha^{[x]} = 0$, $1 \le x \le K$, subject to the condition that the deterioration rates associated with all available MPs can be ordered such that

$$a^{[1]} \le a^{[2]} \le \cdots \le a^{[K+1]}, \tag{13}$$

and

$$\beta^{[1]} \le \beta^{[2]} \le \cdots \le \beta^{[K]}, \tag{14}$$

hold simultaneously. This version is a generalization of one of the cases found in Table 3, in which group-dependent deterioration rates are considered along with identical MPs of constant duration, i.e., $\alpha^{[x]} = 0$, $\beta^{[x]} = \beta$, $1 \le x \le K$. For the latter problem, it is safe to assume that (13) and (14) hold simultaneously, based on the following argument. If identical MPs of equal duration are performed on the machine, then after an MP the condition of the machine can be no better than its condition after the previous MP. In such a case, the deterioration rates do not decrease as more groups are created.

Problem 2: Problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ with group-independent deterioration rates, i.e., $a^{[x]} = a$, $1 \le x \le K+1$, and distinct MPs of start-time dependent durations, subject to the condition that the duration parameters of the MPs can be ordered such that

$$\alpha^{[1]} \le \alpha^{[2]} \le \cdots \le \alpha^{[K]}, \tag{15}$$

and (14) hold simultaneously. This version is a generalization of one of the cases found in Table 3, in which group-independent positional rates are considered along with identical MPs of start-time dependent duration, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \le x \le K$.

In order to solve both versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ described above, the optimal choice for Decisions 2 and 3 can be made easily. If Decision 1 is assumed to be taken, so that $k-1$, $1 \le k \le K+1$, MPs are to be included in the schedule, then for both problems, the MPs with the indices $1, 2, \ldots, k-1$ are chosen and scheduled in the same order. This is the optimal choice for Problem 1, as the MPs with the indices $1, 2, \ldots, k-1$ have the smallest durations and create groups that are associated with the smallest deterioration rates, since (13) and (14) hold simultaneously. This is the optimal choice for Problem 2 as well, as all MPs create identical groups and the ones with a smaller index have smaller values of the duration parameters, since (14) and (15) hold simultaneously. Notice that the order of the MPs is immaterial in both cases.

With Decisions 1-3 having been made (with an assumed value of k), denote the resulting problem as $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$. This problem can be solved by minimizing the generic objective function of the form (6). For Problem 1, obtain the required positional weights $W^{[x]}(r)$ by substituting $\alpha^{[x]} = 0$, $1 \le x \le k-1$, into (10) so that

$$W^{[x]}(r) = \left(1 + a^{[x]}\right)^{n^{[x]}-r},\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{16}$$

while for Problem 2, substitute $a^{[x]} = a$, $1 \le x \le k$, so that

$$W^{[x]}(r) = \begin{cases} \left(1+\alpha^{[x]}\right)(1+a)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ (1+a)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{17}$$

Set the value $n^{[x]} = n$, $1 \le x \le k$, and $k = K+1$, and compute all positional weights $W^{[x]}\left(r\right)$, $1 \le r \le n$, $1 \le x \le K+1$, for both problems by using the formulae above. Notice that the computed positional weights represent a set of all possible values of $W^{[x]}\left(r\right)$ across all possible groups. Further, notice that because of (13), the positional weights associated with Problem 1 are ordered so that for each k, $1 \le k \le K+1$, we have

$$W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k]}(r),\ 1 \le r \le n,$$

while because of (15), the positional weights for Problem 2 are ordered so that for each k, $1 \le k \le K+1$, we have

$$W^{[k]}(r) \le W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k-1]}(r),\ 1 \le r \le n.$$

Definition 1 *If for each position, the positional weight in group x is smaller than the positional weight in the same position in another group y, we say that group x dominates group y. If all available groups can be linearly ordered with respect to the introduced dominance relation, we refer to such a condition as '$K-domi$'.*

Notice that both Problems 1 and 2 satisfy the conditions of $K - domi$. For instances of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ that simultaneously satisfy (14) and $K - domi$, it is possible to compute the optimal values of $n^{[x]}$, $1 \le x \le k$, and take all Decisions 1-4 on the fly. Recall that for a fixed value of Decision 1, the optimal values for Decisions 2 and 3 are already known. Thus, to solve problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ we only need to worry about making Decisions 1 and 4. We base our methodology on Theorem 1.

The method we intend to use for finding an optimal schedule $S^*(k)$ with k groups, $2 \le k \le K+1$, searches for the n smallest positional weights, only by comparing the values in the two lists, which we denote by $G\left(k-1\right)$ and $H\left(k\right)$. List $G\left(k-1\right)$ contains all the positional weights corresponding to the positions used in the previously found schedule $S^*\left(k-1\right)$, while list $H\left(k\right)$ contains the positional weights that will be introduced if the $k-$th group is opened.

List $G(k)$ consists of n smallest positional weights that are available across all positions in groups $1, 2, \ldots k$, where $1 \le k \le K+1$. The list is sorted in non-decreasing order and defined similarly for both Problems 1 and 2. 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

$$C_{\max}\left(S^*\left(k\right)\right) = P(S^*\left(k\right)) + \Gamma(k) = \sum_{j=1}^{n} p_j \gamma_j(k) + \sum_{x=1}^{k-1} \beta^{[x]}. \tag{18}$$

where $P(S^*(k))$ denotes the sum of actual durations of the jobs in an optimal schedule with k groups, and $\Gamma(k)$ is a constant term as defined in (11).

List $H(v)$, $1 \leq v \leq K+1$, is defined differently for Problems 1 and 2. For Problem 1, $H(v)$ contains the positional weights $W^{[v]}(r)$, $v \leq r \leq n$, so that by (16) we have

$$H(v) := \begin{pmatrix} \left(1+a^{[v]}\right)^{n-v} \\ \left(1+a^{[v]}\right)^{n-(v+1)} \\ \vdots \\ \left(1+a^{[v]}\right)^{2} \\ \left(1+a^{[v]}\right) \\ 1 \end{pmatrix}, \ 1 \leq v \leq K+1.$$

For Problem 2, notice that the values of the positional weights given by (17) change dynamically as the value of k is changed. Thus, we define $H(v)$ so that this effect is incorporated: initialize

$$H(1) := \begin{pmatrix} (1+a)^{n-1} \\ (1+a)^{n-2} \\ \vdots \\ (1+a)^{2} \\ (1+a) \\ 1 \end{pmatrix}$$

and define

$$H(v) := \begin{pmatrix} U^{[v-1]}(1+a)^{n-v} \\ U^{[v-1]}(1+a)^{n-(v+1)} \\ \vdots \\ U^{[v-1]}(1+a)^{2} \\ U^{[v-1]}(1+a) \\ U^{[v-1]} \end{pmatrix}, \ 2 \leq v \leq K+1,$$

where $U^{[v-1]} = \left(1+\alpha^{[v-1]}\right)$, $2 \leq v \leq K+1$.

Notice that for both problems, list $H(v)$ has at most $n-v+1$, $1 \leq v \leq K+1$, elements sorted in non-increasing order. It suffices to consider only $n-v+1$ positions in list $H(v)$, since condition $K-domi$ guarantees that each of the $v-1$ earlier groups will have at least one job scheduled in them.

The following algorithm solves an instance of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ and returns the optimal number of MPs, k^*-1, to be included in the schedule (Decision 1) along with the optimal schedule $S^*(k^*)$ with k^* groups (Decision 4).

Algorithm NSmall2

INPUT: An instance of either Problem 1 or Problem 2 with the jobs renumbered in the LPT order

OUTPUT: An optimal schedule $S^*(k^*)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k^*$

Step 1. For $k=1$, find list $H(1)$ for the problem at hand. Define a sorted list $G(1)$ by reordering the elements of $H(1)$ in the opposite order. Compute $C_{\max}(S^*(1))$ by formula (18). Define $k' := K+1$.

Step 2. For k from 2 to k' do

(a) Create the list $G(k) = (\gamma_1(k), \gamma_2(k), \dots, \gamma_n(k))$ that contains n smallest elements in the merger of the lists $G(k-1)$ and $H(k)$.

(b) Compute $C_{\max}(S^*(k))$ by formula (18). 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^*, $1 \leq k^* \leq k'$, such that

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

Step 4. Run Algorithm NSmall for the found value of k^* to obtain the optimal processing sequence $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k^*]}\right)$.

Notice that similar to Algorithm NSmall, Step 2 of the above algorithm also follows Theorem 1 and searches for the n smallest positional weights for a given k, $1 \leq k \leq K+1$, and assigns the jobs with largest values of p_j to the positions corresponding to the smallest positional weights. The main difference between the two algorithms lies in the way the list of n smallest positional weights is found. For each k, $1 \leq k \leq K+1$, Algorithm NSmall searches for the n smallest positional weights by comparing the positional weights across all groups. On the other hand, Algorithm NSmall2 in each iteration compares the elements in only two lists, $G(k-1)$ and $H(k)$. Our method is justified, because list $G(k-1)$ already contains the n smallest positional weights coming from the first $k-1$ groups. Thus, to search for the n smallest weights needed for schedule $S(k)$, there is no need to scan the first $k-1$ groups again. In other words, we utilize the fact, that if a certain position in the first $k-1$ groups is not used in schedule $S^*(k-1)$, it will not be used in schedule $S^*(k)$ either.

In Step 2, for every k both lists $G(k-1)$ and $H(k)$ have at most n elements sorted in a non-decreasing order; therefore each Step 2(a) and Step 2(b) can be completed in $O(n)$ time. If the loop in Step 2 is not broken throughout the run of Algorithm NSmall2, the final value of k' remains equal to $K+1$, i.e., it is possible all MPs will be run and all groups will be opened in an optimal schedule. The loop in Step 2 may be stopped when in Step 2b the condition $P(S^*(k)) = P(S^*(k-1))$ is achieved. This condition implies that the opening of the $k-$th group does not provide any positional weights smaller than those contained in the list $G(k-1)$. If this happens for the $k-$th group, all groups that could be opened after this would provide even worse positional weights, because list $H(k+1)$ is dominated by list $H(k)$, $1 \leq k \leq K$. Thus, the makespan cannot be reduced by running more MPs after the $k'-$th group is opened, so that there is no need to examine further values of k. With the found value of k', the overall optimal schedule will be found among the schedules $S^*(k), 1 \leq k \leq k'$.

Theorem 3 *Algorithm NSmall2 solves an instance either of Problem 1 or of Problem 2 in $O(nK)$ time, provided that the LPT order of the jobs is known.*

Example 3. We illustrate the working of Algorithm NSmall2 by an instance of Problem 2, obtained by modifying the generic instance of Example 1. For Problem 2, the deterioration rate is group-independent and is known to be equal to $a = 0.1$. Such a situation arises when each MP is able to restore the processing conditions to an "as good as new" state. We also modify the duration parameters of the MPs to make them obey (14) and (15) simultaneously, so that they are now given as

$$\begin{array}{lll}\text{Type 1 MP}: & \alpha^{[1]} = 0.025, & \beta^{[1]} = 2; \\ \text{Type 2 MP}: & \alpha^{[2]} = 0.05, & \beta^{[2]} = 4; \\ \text{Type 3 MP}: & \alpha^{[3]} = 0.15, & \beta^{[3]} = 4; \\ \text{Type 4 MP}: & \alpha^{[4]} = 0.25, & \beta^{[4]} = 6; \\ \text{Type 5 MP}: & \alpha^{[5]} = 0.25, & \beta^{[5]} = 6.\end{array}$$

The rest of the setting of this example remains similar to Example 1, with the normal processing times given by

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

Table 6 shows the details of the run of Algorithm NSmall2 for the above instance. Since $\gamma_r(4) = \gamma_r(3)$ for each r, $1 \leq r \leq 6$, the algorithm stops after iteration $k = 4$, so that $k' = 3$. The algorithm outputs the minimum value of the makespan from the set $\{C_{\max}(S^*(k)) | 1 \leq k \leq 3\}$, which is $C_{\max}(S^*(2))$. Optimal sequences of jobs in the two groups can be obtained by running Algorithm NSmall and are found to be $\pi^{[1]} = (5, 3, 1)$ and $\pi^{[2]} = (6, 4, 2)$. The makespan of the resulting schedule is 37.318.

Algorithm NSmall2 can also be applied to solve problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with group-independent deterioration rates and constant duration MPs (both identical and distinct). This is possible since both conditions $K - domi$ and (14) can be satisfied simultaneously. The required running time is again $O(nK)$. We do not discuss the solution of this problem here, since it is possible to solve it even faster using another solution approach, discussed in Section 5.2.

Now consider a version of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ in which the conditions $K - domi$ and (14) do not hold simultaneously. In principle, Algorithm NSmall2 can still be used to obtain a solution for Decisions 1 and 4, but to make Decisions 2 and 3, a full enumeration of options might be required. As a result, the overall running time to solve problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ turns out to be no smaller than that obtained by using the general solution approach presented in Section 4.

We view the material of this subsection as a strong evidence of similarity between the problems with time-dependent effects and positional effects. These two models have been traditionally considered as different in nature. However, Algorithm NSmall2 which solves problems with time-dependent effects shares the same design principles as Algorithm 3 in the recent paper by Rustogi and Strusevich (2012a) on scheduling with positional group-dependent effects. The main point of difference is how to compute the lists $G(k)$ and $H(k)$, but that feature is problem-dependent in any case.

5.2 Binary Search in Convex Sequences

In this subsection, we deal with problems in which the computed positional weights are group-independent, i.e., are of the form $W^{[x]}(r) = W(r)$, $1 \leq x \leq k$, and additionally, they are ordered in a way such that $W(1) \geq W(2) \geq \cdots \geq W(n)$. Such a situation arises for the versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, in which the deterioration rates are group-independent, i.e., $a^{[x]} = a$, $1 \leq x \leq K + 1$, while MPs are of constant duration, i.e., $\alpha^{[x]} = 0$,

$k=1$	r	$H(1)$	$G(1)$	$\gamma_r(1)p_r$
	1	1.61051	1	10
	2	1.4641	1.1	9.9
	3	1.331	1.21	7.26
	4	1.21	1.331	3.993
	5	1.1	1.4641	4.3923
	6	1	1.61051	3.22102
	$C_{\max}(S^*(1)) = \sum_{r=1}^{6} \gamma_r(1)p_r = 38.76632$			

$k=2$	r	$G(1)$	$H(2)$	$G(2)$	$\gamma_r(2)p_r$
	1	1	1.500703	1	10
	2	1.1	1.364275	1.025	9.225
	3	1.21	1.24025	1.1	6.6
	4	1.331	1.1275	1.1275	3.3825
	5	1.4641	1.025	1.21	3.63
	6	1.61051		1.24025	2.4805
	$C_{\max}(S^*(2)) = \sum_{r=1}^{6} \gamma_r(2)p_r + \beta^{[1]} = 35.318 + 2 = 37.318$				

$k=3$	r	$G(2)$	$H(3)$	$G(3)$	$\gamma_r(3)p_r$
	1	1	1.53065	1	10
	2	1.025	1.3915	1.025	9.225
	3	1.1	1.265	1.1	6.6
	4	1.1275	1.15	1.1275	3.3825
	5	1.21		1.15	3.45
	6	1.24025		1.21	2.42
	$C_{\max}(S^*(3)) = \sum_{r=1}^{6} \gamma_r(3)p_r + \beta^{[1]} + \beta^{[2]} = 35.0775 + 2 + 4 = 41.0775$				

$k=4$	r	$G(3)$	$H(4)$	$G(4)$	$\gamma_r(4)p_r$
	1	1	1.5125	1	10
	2	1.025	1.375	1.025	9.225
	3	1.1	1.25	1.1	6.6
	4	1.1275		1.1275	3.3825
	5	1.15		1.15	3.45
	6	1.21		1.21	2.42
	$C_{\max}(S^*(4)) = \sum_{r=1}^{6} \gamma_r(4)p_r + \beta^{[1]} + \beta^{[2]} + \beta^{[3]} = 35.0775 + 2 + 4 + 4 = 45.0775$				

Table 6: Run of Algorithm NSmall2 for Example 3

$1 \leq x \leq K$. Formally, we denote the described problem by $1\left|p_j + a\tau, MP\left[0\right]\right|C_{\max}$. We focus on the problem that MPs are distinct, since the problem with identical MPs is not known to admit a faster solution algorithm.

Notice that for problem $1\left|p_j + a\tau, MP\left[0\right]\right|C_{\max}$, the optimal choice for Decisions 2 and 3 can be made easily. Assume that for problem $1\left|p_j + a\tau, MP\left[0\right]\right|C_{\max}$ an optimal schedule includes $k-1$ MPs, so that the jobs are divided into k, $1 \leq k \leq K+1$, groups. Since it is known that the MPs create groups associated with the same deterioration rates, it follows that the order in which the MPs are performed is immaterial. Further, it is obvious that in order to choose $k-1$ MPs out of the available K ones, the MPs with smaller durations must be given priority. To ensure that the smallest $k-1$ MPs are chosen in an optimal schedule, we renumber the K available MPs in a way that (14) holds and select the ones with indices $1, 2, \ldots, k-1$, and include them into a schedule in the order of their numbering.

With Decisions 1-3 having been made (for an assumed value of k), the resulting problem $1\left|p_j + a\tau, MP\left[0\right](k-1)\right|C_{\max}$ can be solved by minimizing the generic objective function (6). Obtain the required positional weights $W^{[x]}(r)$ by substituting $a^{[x]} = a$, $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (10) so that we have

$$W^{[x]}(r) = (1+a)^{n^{[x]}-r},\ 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k.$$

Notice that, unlike for most previously considered models, here the positional weights in the last group are computed similarly to all other groups.

To solve an instance of problem $1\left|p_j + a\tau, MP\left[0\right](k-1)\right|C_{\max}$, below we outline a solution approach which is again based on Theorem 1.

First, set the value $n^{[x]} = n$, $1 \leq x \leq k$, and from the resulting set of positional weights

$$\begin{pmatrix} (1+a)^{n-1} & (1+a)^{n-1} & \cdots & (1+a)^{n-1} \\ (1+a)^{n-2} & (1+a)^{n-2} & \cdots & (1+a)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ (1+a)^{2} & (1+a)^{2} & \cdots & (1+a)^{2} \\ (1+a) & (1+a) & \cdots & (1+a) \\ 1 & 1 & \cdots & 1 \end{pmatrix},$$

choose the n smallest of them. Obviously, the n smallest weights are found in consecutive positions at the bottom of the matrix. The smallest k positional weights are due to the last positions of each of the k groups. The next smallest k positional weights are due to the second last positions of each of the k groups, and so on. Assuming that $n = \lambda k + \mu$, where λ and μ are non-negative integers, $\mu \leq k-1$, the optimal number of jobs in each group can be given by

$$n^{[x]} = \begin{cases} \left\lceil \frac{n}{k} \right\rceil = \lambda + 1, & 1 \leq x \leq \mu \\ \left\lfloor \frac{n}{k} \right\rfloor = \lambda, & \mu + 1 \leq x \leq k. \end{cases}$$

With known values of $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, and $n^{[x]}$, $1 \leq x \leq k$, an optimal makespan $C_{\max}(S^*(k))$ for problem $1\left|p_j + a\tau, MP\left[0\right](k-1)\right|C_{\max}$ can be found in $O(n)$ time by running the matching algorithm.

To determine the optimal solution for problem $1\left|p_j + a\tau, MP\left[0\right]\right|C_{\max}$, all options associated with Decisions 1-3 must be enumerated. Decisions 2 and 3 have already been taken optimally, thus, we only need to determine the optimal value of the number of MPs. We can do this by solving problem $1\left|p_j + a\tau, MP\left[0\right]\left(k-1\right)\right|C_{\max}$ for all values of k, $1 \leq k \leq K+1$, and choosing the instance that delivers the smallest value of $C_{\max}(S^*\left(k\right))$. Thus, problem $1\left|p_j + a\tau, MP\left[0\right]\left(k-1\right)\right|C_{\max}$ can be solved in $O\left(nK\right)$ time. However, we prove that the sequence $C_{\max}(S^*(k))$, $1 \leq k \leq K+1$, is in fact $V-$shaped and thus, in order to search for the smallest value of $C_{\max}(S^*(k))$, we only need to evaluate $\lceil \log_2\left(K+1\right)\rceil$ options of k, $1 \leq k \leq K+1$. Recall that a sequence $A(k)$ is called $V-$*shaped* if there exists a k_0, $1 \leq k_0 \leq K+1$, such that

$$A(1) \geq \cdots \geq A(k_0-1) \geq A(k_0) \leq A(k_0+1) \leq \cdots \leq A(K+1).$$

Lemma 1 *For problem $1\left|p_j + a\tau, MP\left[0\right]\left(k-1\right)\right|C_{\max}$, if the jobs are numbered in the LPT order, then the makespan of the optimal schedule can be written as*

$$C_{\max}(S^*\left(k\right)) = P(S^*\left(k\right)) + \Gamma(k) = \sum_{j=1}^{n} p_j h\left(\left\lceil \frac{j}{k}\right\rceil\right) + \sum_{x=1}^{k-1}\beta^{[x]},\ 1 \leq k \leq K+1. \tag{19}$$

where we denote $h\left(q\right) := \left(1+a\right)^{q-1},\ 1 \leq q \leq n.$

Proof: The value $C_{\max}(S\left(k\right))$ can be seen as $P(S\left(k\right)) + \Gamma(k)$, where $P(S\left(k\right))$ denotes the sum of the actual durations of the jobs in a schedule $S\left(k\right)$, and $\Gamma(k)$ is the total duration of all $k-1$ MPs defined by (11). If the jobs are numbered in the LPT order, then to minimize the value $P(S(k))$, we need to assign the jobs one by one to an available position with the smallest positional weight. Irrespective of the number of jobs in each group, this can be done by distributing the first k jobs to the last positions in each of the k groups, then the next k jobs to the second last positions in each of the k groups, and so on, until all jobs have been sequenced.

If $j = \lambda k$ then the predecessors of j are placed into the last λ positions of groups $1, 2, \ldots, k-1$ and the last $\lambda - 1$ positions of group k, so that job j is assigned to group k and gets position $\lambda = \left\lceil \frac{j}{k}\right\rceil$ from the end. If $j = \lambda k + \mu$ for $1 \leq \mu \leq k-1$, then the predecessors of j take the last λ positions in each group and additionally the $\left(\lambda+1\right)-$th last position in each of the groups $1, 2, \ldots, \mu - 1$, so that job j gets position $\lambda + 1 = \left\lceil \frac{j}{k}\right\rceil$ from the end in group μ.

It follows that in an optimal schedule $S^*\left(k\right)$, the contribution of a job $j \in N$ to the objective function is equal to $p_j\left(1+a\right)^{\left\lceil \frac{j}{k}\right\rceil - 1}$, and the total processing time for all jobs is equal to

$$P(S^*\left(k\right)) = \sum_{j=1}^{n} p_j h\left(\left\lceil \frac{j}{k}\right\rceil\right). \tag{20}$$

■

Theorem 4 *For problem $1\left|p_j + a\tau, MP\left[0\right]\right|C_{\max}$, the sequence $C_{\max}\left(S^*(k)\right)$, $1 \leq k \leq K+1$, given by (19), is $V-$shaped.*

Proof: The proof is based on the concept of a convex sequence, a discrete analog of a convex function. Recall that a sequence $A(k)$, $1 \leq k \leq K+1$, is called *convex* if

$$A(k) \leq \frac{1}{2}\left(A(k-1) + A(k+1)\right),\ 2 \leq k \leq K.$$

A convex sequence is also $V-$shaped.

It is proved by Rustogi and Strusevich (2011, 2012a), that the sequence $P(S^*(k)) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right)$, $1 \leq k \leq n$, is convex provided that the jobs are ordered in LPT and the sequence $g(r)$, $1 \leq r \leq n$, is non-decreasing. Since the sequence $h(r) = (1+a)^{r-1}$, $1 \leq r \leq n$, is non-decreasing as well, the convexity of the sequence $P(S^*(k))$, $1 \leq k \leq K+1$, of the form (20) follows immediately.

The sequence $\Gamma(k) = \sum_{x=1}^{k-1} \beta^{[x]}$, $1 \leq k \leq K+1$, can also be proved to be convex. Indeed, (14) corresponds to the inequalities $\beta^{[x-1]} \leq \beta^{[x]}$, which can be rewritten as $\beta^{[x-1]} \leq \frac{1}{2}\beta^{[x-1]} + \frac{1}{2}\beta^{[x]}$, $2 \leq x \leq K$. Taking a k, $2 \leq k \leq K$, and summing up, we deduce

$$\begin{aligned}
\sum_{x=2}^{k} \beta^{[x-1]} &\leq \frac{1}{2}\left(\sum_{x=2}^{k} \beta^{[x-1]} + \sum_{x=2}^{k} \beta^{[x]}\right); \\
\sum_{x=1}^{k-1} \beta^{[x]} &\leq \frac{1}{2}\left(\sum_{x=1}^{k-2} \beta^{[x]} + \sum_{x=1}^{k} \beta^{[x]}\right) + \frac{1}{2}\beta^{[k-1]} - \frac{1}{2}\beta^{[1]} \\
&\leq \frac{1}{2}\left(\sum_{x=1}^{k-2} \beta^{[x]} + \sum_{x=1}^{k} \beta^{[x]}\right).
\end{aligned}$$

The last inequality is true as (14) holds, so that

$$\Gamma(k) \leq \frac{1}{2}\left(\Gamma(k-1) + \Gamma(k+1)\right), \ 2 \leq k \leq K,$$

and the sequence $\Gamma(k), 1 \leq k \leq K+1$, is convex. The latter inequalities become equalities if the MPs have identical durations, which happens, e.g., in the problem studied by Yang and Yang (2010). Since the sum of two convex sequences is convex, the sequence $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \leq k \leq K+1$, is convex and, as a result, is also $V-$shaped. ■

Theorem 4 allows us to find an optimal schedule with an optimal number of groups k^* by performing binary search in sequence $C_{\max}(S^*(k))$, $1 \leq k \leq K+1$, with respect to k. As a result at most $\lceil \log_2(K+1) \rceil$ values of k need to be explored, so that the optimal solution for problem $1\left|p_j + a\tau, MP[0]\right|C_{\max}$ can be found in $O(n \log K)$ time.

As in Section 5.1, the approach we use for the problem with group-independent start-time dependent effects is very similar to the one used by Rustogi and Strusevich (2012a) for scheduling with positional group-independent effects. This is another evidence that the two types of models with changing processing times are closer than has been perceived.

6 Conclusion

In this paper, we solve several problems with time-dependent deterioration effects and maintenance activities. We are not aware of any other papers that study the problem of minimizing the makespan, for a model in which a time-dependent effect of the form (2) is combined with multiple MPs. In our paper, we study enhanced models that allow the deterioration rates to be group-dependent and the MPs to have different duration parameters. We propose three solution approaches that solve different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$.

Notice that in scheduling literature, problems with positional effects and start-time dependent effects have been perceived as methodologically different. Not denying the meaningful

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-independent	$O(n \log K)$	$O(n \log K)$	$O(nK)$	$O\left(n2^K \log K\right)$
Group-dependent	$O(nK)$	$O\left(n2^K \log K\right)$	$O(nK \log K)$	$O\left(n2^K K \log K\right)$

Table 7: Computational complexities of different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$.

difference between the two types of effects, we would like to stress the corresponding problems can be handled by ideologically similar solution approaches; please compare this paper with Rustogi and Strusevich (2012a) that studies problems with positional deterioration with multiple maintenance activities.

Table 7 summarizes all the problems considered in this paper along with the running times needed to solve them, provided that the jobs are taken in the LPT order of their normal processing times.

This paper, together with Rustogi and Strusevich (2012a, 2014), form a comprehensive study on single machine models with maintenance activities and changing processing times (subject to positional effects, or start-time dependent effects or their combination), provided that the effects are job-independent and the objectives are the makespan and the total flow time. The problems with other objective functions and of other settings (due date assignment, group technology, batching, parallel machines) are worth investigating. A good problem to consider in the future would involve the study of rate-modifying maintenance activities for models with linear job-dependent time-dependent effects of the form (1). Extended start-time dependent effects, other than linear, have not received sufficient attention, and those of practical relevance deserve to be more actively studied.

References

Brucker P (2007). *Scheduling Algorithms.* Springer: Guildford, Surrey.

Cheng T C E, Ding Q and Lin B M T (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research* **152**: 1–13.

Conway R W, Maxwell W L and Miller L W (1967). *Theory of Scheduling.* Addison Wesley: Reading (MA).

Gawiejnowicz S (1996). A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters* **57**: 297–300.

Gawiejnowicz S (2008). *Time-Dependent Scheduling.* Springer: Berlin.

Gawiejnowicz S, KurcW, and Pankowska L (2006). Pareto and scalar bicriterion scheduling of deteriorating jobs. *Computers and Operations Research* **33**: 746–767.

Hardy G H, Littlewood J E, and Polya G (1934). *Inequalities.* Cambridge University Press: London.

Kubzin M A and Strusevich V A (2005). Two-machine flow shop no-wait scheduling with machine maintenance. *4OR* **3**: 303–313.

Kubzin M A and Strusevich V A (2006). Planning machine maintenance in two-machine shop scheduling. *Operations Research* **54**: 789–800.

Kuo W H and Yang D L (2007). Single machine scheduling with past-sequence-dependent setup times and learning effects. *Information Processing Letters* **102**: 22–26.

Kuo W H and Yang D L (2008). Minimising the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society* **59:** 416–420.

Lodree Jr. E J and Geiger C D (2010). A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. *European Journal of Operational Research* **201**: 644–648.

Okołowski D and Gawiejnowicz S (2010). Exact and heuristic algorithms for parallel-machine scheduling with DeJong's learning effect. *Computers and Industrial Engineering* **59**: 272–279.

Rustogi K and Strusevich V A (2011). Convex and V-shaped sequences of sums of functions that depend on ceiling functions. *Journal of Integer Sequences* **14**: Article 11.1.5, http://www.cs.uwaterloo.ca/journals/JIS/VOL14/Strusevich/strusevich2.html.

Rustogi K and Strusevich V A (2012a). Single machine scheduling with general positional deterioration and rate-modifying maintenance. *Omega* **40**: 791–804.

Rustogi K and Strusevich V A (2012b). Simple matching vs linear assignment in scheduling models with positional effects: A critical review. *European Journal of Operational Research* **222**: 393–407.

Rustogi K and Strusevich V A (2014). Combining time and position dependent effects on a single machine subject to rate-modifying activities. *Omega* **42:** 166–178.

Yang S J (2010). Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. *Applied Mathematics and Computation* **217**: 3321–3329.

Yang S J (2012). Single-machine scheduling problems simultaneously with deterioration and learning effects under deteriorating multi-maintenance activities consideration. *Computers and Industrial Engineering* **62**: 271–275.

Yang S J and Yang D L (2010). Minimising the total completion time in single-machine scheduling with ageing/deteriorating effects and deteriorating maintenance activities. *Computers and Mathematics with Applications* **60**: 2161–2169.

Zhao C L and Tang H Y (2010). Single machine scheduling with general job-dependent aging effect and maintenance activities to minimise makespan. *Applied Mathematical Modelling* **34**: 837–841.