Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 39 (2019) 1119–1128

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

25th International Conference on Production Research Manufacturing Innovation:
Cyber Physical Manufacturing
August 9-14, 2019 | Chicago, Illinois (USA)

Column Generation Algorithms for a Single Machine Problem with Deteriorating Jobs and Deterioration Maintenance Activities

Young-Bin Woo[a], Byung Soo Kim[b], Ilkyeong Moon[a,c,*]

[a]*Department of Industrial Engineering, Seoul National University, Seoul, Republic of Korea*
[b]*Department of Industrial and Management Engineering, Incheon National University, Incheon, Republic of Korea*
[c]*Institute for Industrial Systems Innovation, Seoul National University, Seoul, Republic of Korea*

Abstract

This article addresses the single machine scheduling problem in which the actual processing time of jobs change depending on an elapsed time between its start time and a released time or completion time of a recent maintenance activity. The introduced problem reflects production industry of chemical and metallurgical processes. The objective of this problem is to determine a sequence of jobs and the number and positions of maintenance activities on a single machine to minimize the makespan. We introduce a mixed integer linear programming (MILP) model for the problem. We then propose column generation (CG) algorithms by using Dantzig-Wolfe decomposition approach. The performances of the CG algorithms are evaluated with randomly generated examples.

Keywords: scheduling; time-dependent deterioration; deterioration-maintenance activity; column generation.

1. Introduction

The scheduling problems with deteriorating jobs and deterioration-maintenance activities (DMAs) have been received increasing attention in operation research and arising in applications such as steel production with metallurgical processes [1]. The deteriorating job is defined as a job of which processing time increases depending on

* Corresponding author. Tel.: +82-02-880-7151; fax: +82-02-889-8560.
E-mail address: ikmoon@snu.ac.kr

10.1016/j.promfg.2020.01.358

a state of the job such as temperature. For example, in executing a steel product, a temperature of a material to be processed may get out of a workable range due to its self-cooling. In this case, thus the state of the material necessarily be adjusted causing additional processing time, called deterioration time, before processing the corresponding job. In this study, a deterioration time of a job linearly increases based on its waiting time. The deterioration of job has an important effect on a scheduling criterion of a manufacturing system because the total deterioration time of the entire system rapidly increases as the waiting times of the jobs increase. Meantime, there is an activity, called DMA, that restores conditions of sub-sequential jobs. In real manufacturing situations, a DMA can be interpreted as a decision to release a batch of some jobs at a certain time point in a schedule to save waiting times of the jobs. Therefore, simultaneous determination of two decisions, i.e., the schedule of jobs and the assignments of DMAs, is important to control the manufacturing system in minimizing completion-time-based criteria such as the makespan.

A comprehensive literature review on scheduling problems and models, in which deteriorating jobs and DMA(s) are considered, was described by Cheng, Ding, and Lin [2] and Woo and Kim [3]. Since Gupta and Gupta [4] initially introduced the scheduling problem to sequencing deteriorating jobs on a single machine, many researchers have considered extended problems under several criteria such as the makespan, total completion time and total tardiness. These problems were solved using exact methodologies, i.e., polynomial-time algorithm, optimal policy, and mathematical programming [5–10]. Meanwhile, maintenance is an important activity in production systems because the activity affects the efficiency of production or product quality. Recently, the studies on scheduling problems considering both deteriorating jobs and maintenance for recovering deterioration have been popular topics to researchers [11]. Lodree and Geiger [12] addressed a single machine scheduling problem with simple linear deteriorations of jobs and a DMA. They proposed an optimal policy for providing optimal schedules, but they allowed only one DMA. Zhang et al. [13] addressed an extended problem in which multiple DMAs are allowed and a duration time of DMA is also dependent on its start time. They introduced a polynomial time algorithm referring to the optimality property of Gupta and Gupta [4]. The algorithm can provide optimal schedules for the problem if the number of DMAs is fixed. Yang and Yang [11,14] considered scheduling problems with deteriorating jobs and DMAs whose time is relevant to its start time. They provided a polynomial time algorithm for minimizing the makespan and total completion time of the single machine system, respectively. However, their algorithms were only reasonable for a special case of the problems where the maximum number of DMAs, that is less than the number of jobs, is already known. When there is no limit of the number of DMAs, the algorithms cannot find optimal solutions within the polynomial time. Meanwhile, Woo and Kim [3,15] proposed metaheuristic algorithms to solve the scheduling problem with deteriorating jobs and multiple DMAs for minimizing the makespan.

To the best of our knowledge, even if many researchers addressed scheduling problems with deteriorating jobs and multiple DMAs, no researchers studied an efficient algorithm for obtaining optimal solutions of the problem in which no limit of the number of DMAs is allowed. In this study, we propose a column generation (CG) algorithm. We also present three advance approaches enhancing the performance of the CG algorithm in terms of efficiency and develop three variant CG algorithms. The performances of the algorithms are evaluated with randomly generated instances.

The remaining of the article is organized as follows. In Section 2, we describe the scheduling problem with deteriorating jobs and DMAs and introduce a mixed integer linear programming (MILP) model. In Section 3, we propose a CG algorithm based on Dantzig-Wolfe decomposition approach. In Section 4, we describe the advance approaches for enhancing CG algorithm and introduce variant CG algorithms. In Section 5, numerical experiments are conducted to evaluate the performances of the CG algorithms. Finally, conclusions and further research are presented in Section 6.

2. Problem statement

Fig. 1 describes a single machine schedule of deteriorating jobs and multiple DMAs. There is a given set $I = \{1,2,\ldots,n\}$ $I = \{1,2,...,n\}$ of jobs to be scheduled. Each job is available at time zero and must be scheduled without preemption. Original processing time α_i and linear deterioration rate β_i are given associated with a job $i \in I$. The deterioration times of the jobs are assumed to be dependent on its start time. Accordingly, an actual processing time of every job is $\beta_i s_i + \alpha_i$ where s_i is a start time of job i. More precisely, the deterioration time of job i, $\beta_i s_i$, gradually increases based on its start time if there is no proceeding DMA. We assume that any job cannot be processed during a DMA. The single machine can include one or more DMAs. The objective is to minimize the makespan. For

the described problem, we develop a MILP formulation, named *OP,* to find the optimal schedule. Decision variables for *OP* are as follows:

Decision variables

s_i start time of job $i \in I$
e_i temporary variable representing a completion time of the recent DMA for a sequence of job $i \in I$
x_{ij} equals to 1, if job $i \in I$ precedes job; $j \in I$; 0, otherwise
y_i equals to 1, if job $i \in I$ accompanies a DMA; 0, otherwise
z_i equals to 1, if the state of job $i \in I$ is refreshed; 0, otherwise
C_i completion time of job $i \in I$
$C_{\max}$ makespan

The MILP formulation *OP* is formulated as follows:

$$\textbf{\textit{OP}}: \text{minimize } C_{\max} \tag{1}$$
$$\text{subject to } \sum_{j \in I} x_{ji} = 1, \qquad \forall i \in I \tag{2}$$
$$\sum_{i \in I: i \neq j} x_{ij} \leq \sum_{j \in I} x_{ji}, \qquad \forall i \in I \tag{3}$$
$$\sum_{i \in I} x_{ii} = 1 \tag{4}$$
$$e_i \leq M\left(1 - x_{ii}\right), \qquad \forall i \in I \tag{5}$$
$$e_i \leq e_j + M\left(1 + y_i - x_{ji}\right), \qquad \forall i, j \in I : i \neq j \tag{6}$$
$$e_i \leq s_i + \gamma + M\left(1 - y_i\right), \qquad \forall i \in I \tag{7}$$
$$(1 + \beta_i) s_i + \alpha_i \leq C_i + \gamma z_i, \qquad \forall i \in I \tag{8}$$
$$s_i + \left(s_i - e_i\right)\beta_i + \alpha_i \leq C_i + \gamma\left(1 - z_i\right), \qquad \forall i \in I \tag{9}$$
$$C_i + \gamma y_j \leq s_j + M\left(1 - x_{ij}\right), \qquad \forall i, j \in I : i \neq j \tag{10}$$
$$C_i \leq C_{max}, \qquad \forall i \in I \tag{11}$$

The objective function (1) seeks to minimize the makespan. Constraint (2) describes that each job must be assigned with exactly one preceding job. We note that index j for a preceding job can be the same wih index i for a succeding job. The corresponding variable represents a decision on an assignment of the job to the beginning at a schedule. Constraint (3) shows that at most one succeeding job can exist for each job. Constraint (4) ensures that only one job can be assigned to the beginning of a schedule. Constraints (5)-(7) update the recent completion time of a DMA for each succeeding job. Constraints (8) and (9) calculate the completion time of a job. If the job is accompanied by a DMA, the completion time is defined by Constraint (8). Otherwise, the completion time is defined by Constraint (9). Constraint (10) ensures the precedence relationship of jobs and defines the start time of a succeeding job. Constraint (11) defines the makespan for a schedule.

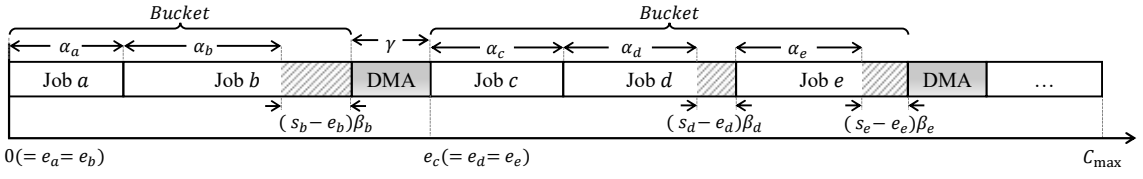

Fig. 1. Single machine schedule of deteriorating jobs and multiple DMAs.

3. Decomposition approach

In this section, we propose an extended formulation *MP* where a set of partitioned jobs, called buckets, is considered as a column of the formulation. We show that the formulation is valid for the addressed problem and it can be a master problem for CG approach. Next, we derive a pricing sub-problem for generating new feasible buckets applying Dantzig-Wolfe decomposition and then introduce a formulation for the sub-problem named *SP1*. Finally, the procedure of a CG algorithm for providing a dual or optimal solution for the addressed problem is organized.

3.1. Extended formulation

In the addressed problem, if there are partial schedules called bucket that consist of partitioned jobs and no DMA, then the completion time of a schedule comprised of the buckets and several DMAs is the same regardless of its sequence of buckets [3]. Applying Dantzig-Wolfe decomposition [16], we now decompose the addressed problem into a master problem for selecting buckets and a sub-problem corresponding to a partial schedule.

Let Ω denote the set of all feasible buckets. Let f_p be a processing time for executing jobs associated with bucket $p \in \Omega$. For each job $i \in I$, let $a_{ip} = 1$ if bucket p contains job i, and $a_{ip} = 0$, otherwise. Define binary variables for $p \in \Omega$: $w_p = 1$ if bucket p is a part of a schedule, $w_p = 1$, otherwise. Then, we obtain an extended formulation for the addressed problem:

$$\textbf{\textit{MP}}\text{: minimize } \sum_{p\in\Omega}\left(f_p+\gamma\right)w_p-\gamma \tag{12}$$

$$\text{subject to } \sum_{p\in\Omega} a_{ip}w_p = 1, \qquad \forall i \in I \tag{13}$$

Objective (12) corresponds to the original Objective (1). The number of DMAs in a complete schedule for the problem exactly equals to the number of buckets partitioning all jobs minus one. Constraint (13) means that each job must be covered by exactly one feasible bucket.

3.2. Column generation approach and bucket sub-problem

As the number of buckets $|\Omega|$ can be considerably large, it is intractable to solve *MP* by enumerating all the columns. Hence, we apply the CG approach, a procedure to efficiently solve a master problem by reiteratively adding a necessary column into the relaxed mater problem of restricted columns, called as a restricted master problem [17,18]. CG approach has been applied to handle large-scale decision problem in various fields such as cutting stock [19], vehicle routing [20], and lot sizing and scheduling [18,21].

When we solve a linear program by using the simplex algorithm, the algorithm proceeds a current basic solution to the next better one with a feasible direction for the relevant objective function. This procedure is achieved by replacing a basic variable with a non-basic one with a negative reduced cost to the basis. The reduced cost is a unit cost consisting of the cost of related variable and cost from compensating a change in the basic variables. Exploiting this property, CG approach reiteratively handles a pricing sub-problem of finding a feasible column that can improve the objective value of the current restricted master problem. Accordingly, the objective of the pricing sub-problem is to find a new feasible column with negative reduced cost.

For the master problem *MP*, a pricing sub-problem is to find a feasible bucket with negative reduced cost. Let π_i be the dual price associated with job i updated by the linear programming (LP) relaxation of the current restricted *MP* relevant to Ω. Then, we obtain a reduced cost r_p corresponding to a candidate bucket p as follows:

$$r_p = f_p + \gamma - \sum_{i\in I} a_{ip}\pi_i \tag{14}$$

By using the function of the reduced cost, more precisely, a bucket sub-problem is to find a subset of jobs i.e., bucket and the sequence of the jobs such that the cost of the corresponding partial schedule minus the total dual variable values of these jobs is minimized. A formulation for the sub-problem, named *SP1,* can be obtained by transforming Constraints (8) and (10) as follows:

$$\textbf{\textit{SP1}}\text{: minimize } C_{max} + \gamma - \sum_{i \in I} \sum_{j \in I} \pi_i x_{ji} \tag{15}$$

subject to Constraints (2)-(4) and (11)

$$\left(1+\beta_i\right) s_i + \alpha_i \le C_i, \qquad \forall i \in I \tag{16}$$

$$C_i \le s_j + M\left(1 - x_{ij}\right), \qquad \forall i, j \in I : i \ne j \tag{17}$$

Objective (15) represents the reduced cost of a column for the restricted *MP*.

3.3. Overall procedures of the proposed CG algorithm

The CG algorithm, named CG_SP1, is implemented as follows: Initialize a restricted *MP* with columns in Ω. If Ω is an empty set, then initialize Ω with $|I|$ columns that contain each job; Solve the restricted *MP*; Next, update coefficients of the sub-problem by using the dual variables for jobs; Solve the sub-problem; Add a new column to the restricted *MP* if the column is found with a negative reduced cost, terminate, otherwise. These procedures are repeated until there is a column that can save the cost for the *MP*.

In the sub-problem, meanwhile, the same column that already exists in the restricted *MP* can be obtained. To control adding the same column in the *MP*, we reiteratively impose a constraint, known as cut-set inequality, that makes a solution point relevant to the existing bucket infeasible. The cut-set inequality for generating new valid buckets is as follows:

$$\sum_{i \in I} \sum_{j \in B_p} x_{ij} \le \sum_{i \in I} \sum_{j \in I \setminus B_p} x_{ij} + \left(\left|B_p\right| - 1\right), \qquad \forall p \in \Omega \tag{18}$$

where B_p is a set of jobs which are included in bucket p.

4. Advanced approaches for CG algorithm

Most of the computation (CPU) times of CG algorithm are resulting from searching feasible buckets in sub-problems. Actually, when we solved some instances of bucket sub-problems for executing 12 or more jobs, the calculation time of single *SP1* took more than 1,800 seconds. To handle this intractability of *SP1,* we introduce three advanced approaches for saving CPU times of the CG algorithm. First, we propose another formulation for the bucket sub-problem. Next, valid constraints are imposed on the formulation to make generated bucket tight. Lastly, bucket heuristic for providing some feasible buckets to restricted *MP* is developed.

4.1. Formulation for the bucket sub-problem with an optimality property

There is a useful lemma for sequencing jobs whose processing times linearly increase with its start time if it is no need to determine positions of DMAs.

Lemma 1. For the scheduling problem $M1 \mid p_i = \alpha_i + \beta_i s_i \mid C_{\max}$, a schedule of jobs that are sequenced in non-decreasing order of α_i / β_i is an optimal solution.

Proof. This can be verified by observing the difference between the makespans of an arbitrary schedule contain two consecutive jobs and the modified schedule where the only two jobs are swapped (see Gupta and Gupta [4]). □

Under this lemma, we no longer have to search the sequence of partial jobs in finding feasible buckets. In other words, if we choose some jobs to be associated with a new bucket, then the sub-schedule, that equals to the nondecreasing order of the relevant jobs, dominates other sub-schedules of the jobs. To tighten regions of valid feasible buckets, we propose another valid formulation for the sub-problem by using Lemma 1 as follows:

$$\textbf{\textit{SP2}}\text{: minimize } C_n + \gamma - \sum_{i=1}^{n} u_i y_i \tag{19}$$

$$\text{subject to } C_0 = 0 \tag{20}$$

$$C_{i-1}(1+\beta_i) + \alpha_i y_i \le C_i + M(1-y_i), \qquad \forall i = 1\ldots n \tag{21}$$

$$C_{i-1} \le C_i + M y_i, \qquad \forall i = 1\ldots n \tag{22}$$

$$\beta_{i+1} C_i \le \gamma + M(1-x_{i+1}), \qquad \forall i = 1\ldots n \tag{23}$$

$$\sum_{i \in B_p} y_i \le \sum_{i \in I,\ B_p} y_i + \left(|B_p| - 1\right), \qquad \forall p \in \Omega \tag{24}$$

Objective (19) represents the reduced cost of a column for the restricted *MP*. Constraint (20) means that available time is zero. Constraints (21) and (22) describe that the completion time of the succeeding index can be defined based on that of the preceding index, and if the job of the succeeding index is not included in the bucket solution, the completion time of it is defined as that of the preceding index with no additional cost. Constraint (23) means that the deterioration time of a succeeding job cannot exceed the DMA time. Constraint (24) corresponds to Constraint (18).

4.2. Generation of tight buckets

In CG, a feasible bucket is generated to be added into columns of the restricted *MP* until there is no feasible subset of jobs with negative reduced cost. However, the bucket is not necessarily relevant to an optimal schedule while the reduced cost of its relevant column is negative. For example, there is an instance where four jobs and DMA of 10 unit time exist as described in Table 1. If we initialize columns with buckets of only one job, then dual variable values are updated as $\alpha_i + \gamma$ for each job i at the first run of the restricted *MP*. We can get a feasible bucket of all the jobs, i.e., {1, 2, 3, 4} and completion times of the jobs by Eq. (16). However, it can be easily seen that the bucket is not relevant to an optimal solution: if we exclude job 4 from the bucket and add a bucket with job 4 such as {1, 2, 3} and {4}. In other words, if we add a DMA before the last job, we can save the cost by 10.89 minus 10 unit time. In fact, in the proposed CG algorithm, some buckets can be generated that are valid but not relevant to an optimal solution due to significant deterioration times.

Table 1. Schedule of four-jobs example.

Job	Processing time	Deterioration rate	Ratio	Start time	Deterioration time	Completion time
1	28	0.18	155.56	0	0	28
2	33	0.17	194.12	28	4.76	65.76
3	26	0.11	236.36	65.76	7.23	98.99
4	35	0.11	318.18	98.99	10.89	144.88

Hence, it is valid that jobs whose deterioration time exceeds the DMA time can be stipulated in generating a bucket. Thus, we add the following constraint to the sub-problem:

$$\beta_j C_i \le \gamma + M\left(1 - x_{ij}\right) \qquad \forall i, j \in I : i \ne j \tag{25}$$

Constraint (25) stipulates a deterioration time occurred by a succeeding job must be less than the DMA time.

4.3. Bucket generation heuristic

It can be a burden to find feasible buckets in an instance with large-sized jobs because we handle an integer programming for the sub-problem. Hence, we present a heuristic for providing feasible buckets (or columns) to CG algorithm as initial buckets. It is not a new approach to use some columns obtained from a heuristic as the initial column of CG algorithm. But it is known an efficient way to reduce iterations for generating columns in CG procedure [20].

The Longest Processing Time first (LPT) rule is an effective rule-based heuristic for balancing loads of parallel machines [22]. Meanwhile, suppose that the number of DMAs required for a schedule is already known as q. Then, the addressed problem is exactly the same as the problem to assign all jobs to $q+1$ buckets. Considering a bucket as a machine for the load-balancing problem, we propose a bucket generation heuristic by adopting the feature of the LPT heuristic. The procedure of the bucket generation heuristic is as follows: First, set the number of buckets be as large integer number as possible such as $|I|$; Next, select a job in order of the ratio of jobs, and then assign this job to a bucket where its expected completion time is minimized; Decrease the number of buckets by one, then repeat the procedures until the number reaches zero. As mentioned in Section 3.4.1, this procedure also generates some buckets that are valid but not relevant to an optimal solution due to considerable deterioration times. Hence, we add a termination condition that a deterioration time of a single job exceeds the DMA time into the heuristic.

Applying the approaches that were introduced in Section 4.1-4.3, we finally organize three variants of the CG algorithm. First one, named CG_SP2, uses formulation *SP2* instead of *SP1* for generating buckets. The second one, named CG_SP2/C, generates buckets tightened by Constraint (25). The last one, named CG_SP2/C/H, initializes columns of the restricted *MP* with the buckets resulting from the bucket generation heuristic.

5. Numerical Experiments

Here we report the performances of the four proposed CG algorithms, i.e., CG_SP1, CG_SP2, CG_SP2/C, and CG_SP2/C/H. The numerical experiments are conducted on 3.4 GHz Intel Core i5-3570 CPU. We set a CPU time limit of 1,800 seconds to a termination condition for each run of an algorithm even if a CG algorithm has not yet found all feasible buckets with negative reduced costs. To obtain solutions of mathematical programs, we used a commercial solver, known as ILOG IBM CPLEX 12.8. The CPU time limit was also imposed when an *OP* model was solved.

Referring to the experiment design of previous study [3], all instances for the experiments were randomly generated with a set of problem settings characterized by the following three factors: the number of jobs $|I|$, DMA time γ, and mean of deterioration rate μ_β. Original processing times were generated by $U\left(0.8\times 480/|I|, 1.2\times 480/|I|\right)$. Note that to reflect the production plan for a working day, we assumed the working hour is 480 min. Deterioration rates were generated by $U\left(\mu_\beta-5, \mu_\beta+5\right)/100$. Because the complexity of MILP highly depends on the number of jobs to be scheduled, we set two groups of problem settings for small-sized jobs and large-sized jobs as described in Tables 2 and 3. Then, we randomly generated 10 instances for each problem setting. Therefore, we had 120 small-sized instances and 120 large-sized instances.

We run CPLEX solver and all the proposed CG algorithms for the instances of the first group. The results for the instances are summarized in Table 3. The average number of buckets, average CPU times, average dual objectives for ten instances in each problem setting, and the number of optimal solutions are reported for each CG algorithm. CPLEX solver found optimal solutions for the problem instances under 8 jobs. However, for the instances over 12 jobs CPLEX solver just provided feasible solutions with dual bounds. For all the instances, the CG algorithm where *SP2* formulation is applied, i.e., CG_SP2, CG_SP2/C, and CG_SP2/C/H completed their procedures within the CPU time limit and could provide the same dual or optimal solutions. But, CG_SP1 algorithm could not complete the procedure for the instances over 12 jobs. In this case, only the primal solution with the initial 12 buckets was outputted because *SP1* did not find a next feasible bucket within the CPU time limit. It is shown that generating a bucket by using the formulation *SP2* is more efficient than *SP1*. For the number of buckets, CG_SP2/C algorithm, in which Constraint (25) is imposed in the sub-problem, could provide the same solution with less searching for an average of 14 percent buckets compared to CG_SP2 algorithm. This means that tightening feasible buckets by using Constraint (25) dose not cut off the optimal solutions and helps to efficiently search the feasible bucket relevant to the optimal solutions.

Similarly, CG_SP2/C/H algorithm, in which initial columns (or buckets) are provided by the bucket heuristic, provided the same solution with less searching for an average of 18 percent buckets compared to CG_SP2/C algorithm. This possibly means that initial buckets obtained by the bucket heuristic are somewhat related optimal solution. The CPU time of CG_SP2/C/H was the smallest, followed by that of CG_SP2/C and CG_SP2.

Table 2. The result for the instance setting of small-sized jobs.

Inst. Set.			CPLEX		CG_SP1				CG_SP2				CG_SP2/C				CG_SP2/C/H			
$\|I\|$	γ	μ_β	Avg. CPU.	# of Opt.	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.
4	20	10	0.06	10	8	0.13	541.7	9	8	0.07	541.7	9	8	0.09	541.7	9	6.5	0.05	541.7	9
4	20	20	0.05	10	4	0.03	561.8	10	4	0.03	561.8	10	4	0.04	561.8	10	4	0.06	561.8	10
4	40	10	0.05	10	11.1	0.18	568.6	7	11	0.11	568.6	7	9.4	0.1	568.6	7	7.3	0.06	568.6	7
4	40	20	0.06	10	9.5	0.15	608.8	10	9.4	0.08	608.8	10	9.5	0.08	608.8	10	7.3	0.05	608.8	10
8	20	10	64.13	10	27.6	92.45	594.6	8	27.2	0.36	594.6	8	23.2	0.25	594.6	8	17.8	0.17	594.6	8
8	20	20	74.35	10	22.6	41.41	613.3	10	22.8	0.23	613.3	10	22.7	0.21	613.3	10	17.4	0.14	613.3	10
8	40	10	44.04	10	34	161.06	647.9	0	34.1	0.47	647.9	0	29	0.35	647.9	0	25.7	0.31	647.9	0
8	40	20	85.04	10	26.9	69.12	693.9	4	27	0.31	693.9	4	23.7	0.21	693.9	4	18	0.14	693.9	4
12	20	10	1,800	0	12	1,800	715.8	0	52.5	1.62	617.6	3	43.1	1.29	617.6	3	36.3	1.04	617.6	3
12	20	20	1,800	0	14.1	1,800	702.9	0	43.5	0.95	650.1	10	38.2	0.83	650.1	10	28.6	0.52	650.1	10
12	40	10	1,800	0	12	1,800	955.6	0	61.3	1.83	692.6	3	49.1	1.58	692.6	3	44.9	1.47	692.6	3
12	40	20	1,800	0	12	1,800	954.7	0	55.1	1.56	745.7	8	47.3	1.35	745.7	8	38.5	1.01	745.7	8
Avg.			622.32	6.7	16.4	630.38	678	4.8	29.7	0.64	628	**6.8**	25.6	0.53	628	**6.8**	21.0	**0.42**	628	**6.8**

We excluded CG_SP1 algorithm in the following experiments on solving the instances of the second group because the algorithm had not completed some instances over 12 jobs within the time limit. Hence, we run CG_SP2, CG_SP2/C, and CG_SP2/C/H for the instances with large-sized jobs. Table 3 summarizes the results for the instances. We additionally recorded the last negative price, i.e., the reduced cost of a bucket recently generated, for a single run of an algorithm. Less value of the price implies that more feasible buckets should be searched in running of an algorithm. When all the algorithms completed their procedure within the CPU time limit, the same solutions were provided for an instance. While, not all algorithms were able to complete the procedures within the time limit for some instances associated with 9th, 11th, and 12th instance settings in Table 3. This is because the branch and bound algorithm solving the sub-problems relatively have to more branch variables as a valid bucket can contain more jobs. Note that if the DMA time increases or the deterioration rates decrease, then relatively many jobs can be included in a valid bucket with satisfying Constrain (23). Particularly, the average numbers of buckets found by CG_SP2 for the instances associated with the 9th, 11th, and 12th instance settings were less than those found by CG_SP2/C and CG_SP2/C/H (see Table 3). This possibly means that Constraint (25) not only made the CG algorithm to efficiently search feasible buckets but also shortened the CPU time incurred by the sub-problems for the instances. Meanwhile, the average numbers of buckets found by CG_SP2/C/H were larger than those found by CG_SP2/C. The difference between both average numbers was approximately equal to the number of the initial buckets obtained by the bucket heuristic.

Table 3. The result for the instance setting of large-sized jobs.

Inst. Set.			CG_SP2					CG_SP2/C					CG_SP2/C/H				
$\|I\|$	γ	μ_β	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. price	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. price	Avg. $\|\Omega\|$	Avg. CPU.	Avg. Obj.	# of Opt.	Avg. price
15	10	15	56.3	2.08	591.11	1	-	47.7	1.59	591.11	1	-	44.3	1.53	591.11	1	-
15	20	25	50.7	1.55	615.78	4	-	50.8	1.57	615.78	4	-	50.8	1.8	615.78	4	-
15	20	15	78.1	4.32	639.88	10	-	63.4	3.73	639.88	10	-	53.7	2.7	639.88	10	-
15	20	25	57	2.16	673.29	3	-	48.2	2.74	673.29	3	-	41.2	2.57	673.29	3	-
30	10	15	176.1	121.95	642.64	6	-	136.7	85.31	642.64	6	-	113.3	66.2	642.64	6	-
30	10	25	121.7	24.37	684.81	3	-	103.6	13.7	684.81	3	-	84	10.02	684.81	3	-
30	20	15	214.8	687.29	711.65	0	-	157.8	498.2	711.65	0	-	147.7	454.63	711.65	0	-
30	20	25	175.4	125.26	778	4	-	135.4	100.03	778	4	-	117.4	87.43	778	4	-
45	10	15	68.1	1,800	802	0	-28.43	183.2	1,797.75	653.36	0	-0.05	171	1,779.61	653.34	0	-0.05
45	10	25	235.5	609.01	702.97	0	-	193.1	682.85	702.97	0	-	163.8	544.69	702.97	0	-

45	20	15	45	1,800	1361.2	0	N/A	79.5	1,800	993.4	1	-43.3	105.6	1,799.2	762.05	5	-1.75
45	20	25	45	1,800	1358.5	0	N/A	168	1,800	844.65	0	-0.11	175.2	1,800	844.6	0	-0.03
Avg.			110.3	581.5	684.21	2.1	-28.43	114	565.62	710.96	2.3	-14.49	105.7	545.87	691.68	2.6	**-0.61**

6. Conclusions

We addressed the scheduling problem with deteriorating jobs and multiple DMAs. The objective of this problem was to determine a sequence of jobs and the number and positions of DMAs on a single machine to minimize the makespan. We developed a MILP model for the problem. We then proposed an extended formulation on subsets of jobs called buckets. Sequentially we proposed a CG algorithm named CG_SP1, in which the sub-problems are reiteratively solved to generate new feasible buckets. To enhance the performances of the algorithm in terms of efficiency, we suggested three approaches: Using another valid formulation for the sub-problems where an optimality property is applied; Imposing constraints for tightening feasibility of buckets; Generating initial columns with bucket generation heuristic. We then developed three variant CG algorithms, CG_SP2, CG_SP2/C, and CG_SP2/C/H. The performances of the proposed CG algorithms were evaluated by conducting numerical experiments through randomly generated instances. Our major findings are as follows: Generating a bucket by using the formulation *SP2* was more efficient than that by using the formulation *SP1*; Constraint (25) for tightening feasibility of buckets could save the CPU time in solving the sub-problems and was effective in generating buckets relevant to optimal solutions; The proposed bucket generation heuristic could provide some buckets relevant to optimal solutions to the CG algorithm.

Further research will develop a branch and price algorithm referring to the CG algorithms proposed in this paper and will extend the addressed problem to a parallel machine scheduling (PMS) problem with ready time or unrelated PMS problem.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning [Grant no. 2017R1A2B2007812].

References

[1] A.S. Kunnathur, S.K. Gupta, Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem, Eur. J. Oper. Res. (1990).

[2] T.C.E. Cheng, Q. Ding, B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, Eur. J. Oper. Res. 152 (2004) 1–13.

[3] Y.-B. Woo, B.S. Kim, Matheuristic approaches for parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities, Comput. Oper. Res. 95 (2018).

[4] J.N.D. Gupta, S.K. Gupta, Single facility scheduling with nonlinear processing times, Comput. Ind. Eng. 14 (1988) 387–393.

[5] A. Bachman, A. Janiak, M.Y. Kovalyov, Minimizing the total weighted completion time of deteriorating jobs, Inf. Process. Lett. 81 (2002) 81–84.

[6] M.Ji, T.C.E. Cheng, Parallel-machine scheduling with simple linear deterioration to minimize total completion time, Eur. J. Oper. Res. 188 (2008) 342–347.

[7] J.B. Wang, L.H. Sun, L.Y. Sun, Single-machine total completion time scheduling with a time-dependent deterioration, Appl. Math. Model. 35 (2011) 1506–1511.

[8] J.-B. Wang, L.-Y. Wang, D. Wang, X.-Y. Wang, Single-machine scheduling with a time-dependent deterioration, Int. J. Adv. Manuf. Technol. 43 (2009) 805–809.

[9] X.Y. Wang, J.J. Wang, Scheduling deteriorating jobs with a learning effect on unrelated parallel machines, Appl. Math. Model. 38 (2014) 5231–5238.

[10] H. Zhu, M. Li, Z. Zhou, Machine scheduling with deteriorating and resource-dependent maintenance activity, Comput. Ind. Eng. 88 (2015) 479–486.

[11] S.J. Yang, D.L. Yang, Minimizing the makespan on single-machine scheduling with aging effect and variable maintenance activities, Omega. (2010).

[12] E.J. Lodree, C.D. Geiger, A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration, Eur. J. Oper. Res. (2010).
[13] X. Zhang, Y. Yin, C.-C. Wu, Scheduling with non-decreasing deterioration jobs and variable maintenance activities on a single machine, Eng. Optim. (2016) 1–14.
[14] S.J. Yang, D.L. Yang, Minimizing the total completion time in single-machine scheduling with aging/deteriorating effects and deteriorating maintenance activities, Comput. Math. with Appl. (2010).
[15] Y.-B. Woo, S. Jung, B.S. Kim, A rule-based genetic algorithm with an improvement heuristic for unrelated parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities, Comput. Ind. Eng. 109 (2017).
[16] G.B. Dantzig, P. Wolfe, Decomposition Principle for Linear Programs, Oper. Res. (2008).
[17] L.S. Lasdon, Optimization theory for large systems, Courier Corporation, 2002.
[18] Z.L. Chen, W.B. Powell, Solving parallel machine scheduling problems by column generation, INFORMS J. Comput. (1999).
[19] P.C. Gilmore, R.E. Gomory, A Linear Programming Approach to the Cutting Stock Problem - Part II, Oper. Res. (1961).
[20] M. Mourgaya, F. Vanderbeck, Column generation based heuristic for tactical planning in multi-period vehicle routing, Eur. J. Oper. Res. (2007).
[21] D. Cattrysse, M. Salomon, R. Kuik, L.N. Van Wassenhove, A Dual Ascent and Column Generation Heuristic for the Discrete Lotsizing and Scheduling Problem with Setup Times, Manage. Sci. (2008).
[22] M.L. Pinedo, Scheduling: Theory, algorithms, and systems, 2008.