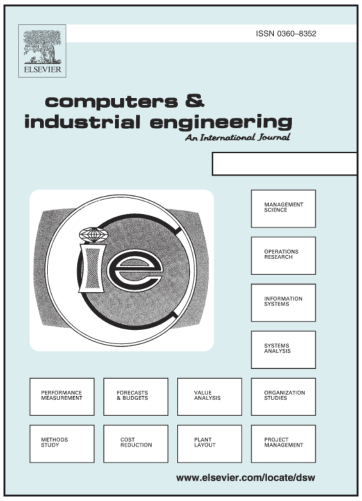


A scatter simulated annealing algorithm for the bi-objective scheduling problem for the wet station of semiconductor manufacturing

Jihong Pang, Hongming Zhou, Ya-Chih Tsai, Fuh-Der Chou

PII: S0360-8352(18)30289-4
DOI: https://doi.org/10.1016/j.cie.2018.06.017
Reference: CAIE 5280

To appear in: *Computers & Industrial Engineering*

Received Date: 5 July 2017
Revised Date: 3 June 2018
Accepted Date: 11 June 2018

A scatter simulated annealing algorithm for the bi-objective scheduling problem for the wet station of semiconductor manufacturing

Jihong Pang[1], Hongming Zhou[1], Ya-Chih Tsai[2], Fuh-Der Chou[1]

College of Mechanical and Electronic Engineering, Wenzhou University, Wenzhou 325035, Zhejiang, China[1]

Department of Hotel Management, Vanung University, Jung-Li, Tao Yuan, Taiwan, R.O.C.[2]

Abstract

This paper studies a single-machine scheduling problem where flexible maintenance is required in the planning horizon. The problem observed is in the clean operation of semiconductor manufacturing, where jobs are associated with release times, processing times, due dates, penalty weight of tardy jobs and the amount of dirt left on the machine during processing. The machine, named the wet station, should be stopped for cleaning or maintenance before a maximum amount of dirt has accumulated; the cleaning time is fixed and is known in advance. However, the starting point of each maintenance activity is a decision variable. Preemptive operation is not allowed. The bi-objective of minimizing total weighted tardiness and total completion time is considered in this paper. An intuitive threshold method and a dynamic programming approach are proposed for scheduling jobs and PMs under a given job sequence. A mixed-integer programming formulation is developed to obtain all efficient solutions for the small-size problems. In addition, this paper also develops a scatter simulated annealing (SSA) algorithm in which a scatter-search mechanism leads SSA to explore more potential solutions. Computational experiments are conducted to examine the efficiency of the SSA algorithm. For the small-size problems, SSA could obtain all non-dominated solutions except for a tiny fraction of the instances. For the large-size problems, SSA performs considerably well compared with SMOSA and NSGA-II, demonstrating its potential to efficiently solve bi-objective single problems with flexible maintenance activities.

Keywords: Semiconductor manufacturing; dynamic release time; scatter search mechanism; preventive maintenances; simulated annealing

Corresponding author: Fuh-Der Chou Tel: +86-13676582151; Fax: +86-577-86689138
E-mail: fdchou@tpts7.seed.net.tw

A scatter simulated annealing algorithm for the bi-objective scheduling problem for the wet station of semiconductor manufacturing

This paper studies a single-machine scheduling problem where flexible maintenance is required in the planning horizon. The problem observed is in the clean operation of semiconductor manufacturing, where jobs are associated with release times, processing times, due dates, penalty weight of tardy jobs and the amount of dirt left on the machine during processing. The machine, named the wet station, should be stopped for cleaning or maintenance before a maximum amount of dirt has accumulated; the cleaning time is fixed and is known in advance. However, the starting point of each maintenance activity is a decision variable. Preemptive operation is not allowed. The bi-objective of minimizing total weighted tardiness and total completion time is considered in this paper. An intuitive threshold method and a dynamic programming approach are proposed for scheduling jobs and PMs under a given job sequence. A mixed-integer programming formulation is developed to obtain all efficient solutions for the small-size problems. In addition, this paper also develops a scatter simulated annealing (SSA) algorithm in which a scatter-search mechanism leads SSA to explore more potential solutions. Computational experiments are conducted to examine the efficiency of the SSA algorithm. For the small-size problems, SSA could obtain all non-dominated solutions except for a tiny fraction of the instances. For the large-size problems, SSA performs considerably well compared with SMOSA and NSGA-II, demonstrating its potential to efficiently solve bi-objective single problems with flexible maintenance activities.

Keywords: Semiconductor manufacturing; dynamic release time; scatter search mechanism; preventive maintenances; simulated annealing

1. Introduction

Scheduling plays a key role in allocating resources, such as machines, labors, and materials, to tasks effectively for enterprises to retain/improve competitiveness in the fast-changing market. A variety of scheduling problems in different manufacturing environments, including single machine, parallel machines, flow shop, and job shop have been investigated in the past decades. In addition, numerous realistic requirements or constraints have also been incorporated into scheduling problems. For example, dynamic job release times and machine unavailability are the most common and significant constraints in the real world. Machine unavailability due to preventive maintenance (PM) is found prevalently in manufacturing systems. PM is a scheduled downtime planned for the requirements of inspection, repair, or cleaning and is performed periodically. An important implication of PM is that when PM is performed, no jobs can be processed on the machine, even if there are many available jobs that need to be processed. Thus, it is necessary to consider the planning of PM and the scheduling of jobs in an integrated way.

The job's release time, which is related to dynamic scheduling, is the time at which the job is available for processing. The majority of scheduling problems with machine availability assume that all jobs simultaneously arrive for processing, i.e., zero release times. In this paper, an extended version of the static single-machine scheduling problem, with a dynamic job release time and flexible maintenance, is investigated. The considered problem is a real-life problem that appears in the wet station of semiconductor wafer fabrication. All jobs do not arrive simultaneously for processing, and the machine, named the wet station, is not always available. The machine is designed to be stopped for cleaning of the residual chemical materials (dirt) left by the wafers to avoid damaging the quality of the wafers. Therefore, maintenance is performed so that the accumulated dirt in the machine does not exceed the pre-specified limitation. Moreover, the amount of dirt left by the wafers being processed is not the same. As a result, the starting time of maintenance is flexible, which is determined by the accumulated dirt. Note that these two characteristics make the single machine scheduling problem more realistic but increase its complexity drastically.

The objective in this paper is to minimize the total weighted tardiness and the total completion time simultaneously. On the one hand, minimizing the total weighted tardiness involves meeting due dates and avoiding delay penalties. On the other hand, the minimization of total completion time can effectively reduce the work in process. For many industries, such as chemistry, electronics, and semiconductor manufacturing, this bi-objective minimization could optimize the overall performance of the system and improve the competitiveness of the company.

Although this work is motivated by the wet station, it actually represents a class of single-machine scheduling problems with the combined-characteristics of dynamic job release time, flexible maintenance, and bi-objective. Those characteristics are common in the real-life manufacturing system, and disregarding any of them could lead to an ineffective production schedule. Based on 3-field denotation, the problem can be defined as $1\left|r_j, cleaning\right|\sum w_jT_j\,, \sum C_j$ (Pinedo 1995), where the first field denotes a single machine, and the second field (r_j) and 'cleaning' denotes the job release time and clean activity, respectively. The last field ($\sum w_jT_j$ and $\sum C_j$) denotes the total weighted tardiness (*TWT*) and total completion time (*TCT*), respectively. To the best of our knowledge, this is the first work to consider a single machine with unavailable periods while optimizing the total weighted tardiness and total completion time simultaneously. As the first attempt at addressing this problem, we construct a mixed-integer linear programming model. In addition to the presentation of the model, we propose two heuristic algorithms and a scatter simulated annealing (SSA) algorithm to solve the large-sized problems. SSA incorporates a dynamic programming technology and a novel search method to find non-dominant solutions. Extensive computational results and comparisons show that the proposed SSA algorithm is effective for the $1|r_j, cleaning|\sum w_j T_j, \sum C_j$ problem.

This paper is organized as follows. In Section 2, the previous research involving a different scheduling problem with PM is reviewed. Then, the characteristics of the problem are described, and two methods for scheduling jobs and PMs are introduced in Section 3. Additionally, a mixed-integer programming (MIP) model is also addressed. In Section 4, the details of the scatter simulated annealing (SSA) algorithm are explained, including the parameter settings and the search scheme. In Section 5, the results of the computational experiments are presented. Finally, Section 6 contains our conclusions and directions of future research.

2. Literature Review

With the enthusiastic adoption of the Total Productive Maintenance (TPM) philosophy during the last five decades, many industries perform preventive maintenance (PM) plans to keep their machines in good working condition. However, conducting PM on the machine means that the machine is unavailable for processing jobs, resulting in a decrease of machine capacity. Thus, scheduling models with machine availability have received considerable attention from many researchers and practitioners. In the relevant scheduling studies, three types of problems involving machine availability can be found: (1) fixed maintenance, where the period of maintenance is known in advance; (2) flexible maintenance, in which the start time and/or the maintenance duration must be determined during the process of scheduling; (3) and stochastic maintenance, in which the machines breakdown randomly (Wang and Liu 2013).

In the first case, Lee (1996) proved that the single machine scheduling problem with one machine unavailability period in terms of weighted completion time is NP-hard. For the same problem, Kacem and Chu (2008) provided an efficient branch-and-bound algorithm to minimize the weighted sum of completion times. Meanwhile, Kacem et al. (2008) compared the performance of three algorithms, including the MIP model, dynamic programming (DP) and branch and bound (BAB) methods. They showed that the DP and BAB methods are better than the MIP model. Mor and Mosheiov (2012) studied a single machine scheduling problem with an unavailability constraint, where the processing times of jobs are position-dependent. They developed three heuristic algorithms minimizing makespan, total completion time, and number of tardy jobs, respectively. Moncel et al. (2014) considered the objective of minimizing total completion time and provided an improved relative error for the heuristic algorithm (MSPT-k) addressed by He et al. (2006). Detienne (2014) considered the release time of jobs and maintenance constraints at the same time for the single machine scheduling problem and proposed a MIP model to minimize the number of late jobs. Low et al. (2010) proposed a modified particle swarm optimization algorithm to minimize the makespan. Other related important studies and some reviews considered the fixed maintenance in production scheduling with different optimization objectives and are listed in Lee (1991), Lee and Liman (1992), Schmidit (2000), and Ma et al. (2010).

Regarding the scheduling problem with flexible maintenance, Kubzin and Strusevich (2006) were the first to investigate variable maintenance in a two-machine flow shop and two-machine open shop. In their model, the maintenance period varies depending on the starting time *t* of the maintenance, that is, the later the maintenance activity begins, the longer maintenance duration will be, this kind of maintenance activity is also called deteriorating maintenance activity in the literature. Luo et al. (2015) considered a variable maintenance activity on a single-machine problem in the case of deteriorating maintenance activity and where the maintenance activity must start before a given deadline. They found that the problem, with the minimizing of the makespan, sum of completion times, maximum lateness, and number of tardy jobs, could be solved by their proposed polynomial-time algorithms. Ying et al. (2016) considered the same problem of Luo et al. (2015), but they considered four different objectives of minimizing mean lateness, maximum tardiness, total flow time and mean tardiness. The four problems could be solved by their proposed exact algorithms in $O(n^2)$ time, respectively. Cheng et al. (2011) considered an unrelated parallel machine scheduling problem with deteriorating maintenance activities, where each machine has at most one maintenance activity, and the maintenance activity could be performed at any time throughout the planning horizon. They considered the two objectives of minimizing the total completion time and the total machine load, respectively, and showed that the two problems could be optimally solved in polynomial time. A deteriorating maintenance activity was involved in several studies, including these works by Mosheiov and Sidney (2010), Yang and Yang (2010), and Yang (2012).

Another researcher assumed that the maintenance activity must be executed in predefined intervals [u, v]. u (v) is defined as the earliest (latest) time at which the machine starts (stops) its maintenance, and w is the maintenance time, which must be less than or equal to the period [u, v]. Yang et al. (2002) were the first to investigate this case. For the single machine scheduling problem with a flexible maintenance activity and the objective of minimizing makespan, they proved that the problem is *NP*-hard and proposed an O($n$log$n$) time algorithm to solve the problem. For the case of multiple maintenance activities in a single-machine problem, Chen (2008) proposed two mixed integer programming models and an efficient heuristic to minimize makespan. Xu et al. (2009) showed that the worst-case performance bound for the heuristic proposed by Chen (2008) is 2. Lee and Chen (2000) considered a parallel machine where each machine must be maintained once, and the objective is to minimize the total weighted completion time. Several important studies involving this case can be found in Chen (2006), Yang et al. (2011), Xu and Yin (2011) etc.

Qi et al. (1999) considered another type of flexible maintenance activity where the machine must be maintained after it continuously works for a maximum allowed continuous working time. They proved the single machine scheduling problem to be *NP*-hard and developed heuristics and a BAB method to minimize the sum of completion time. Graves and Lee (1999) considered a single machine to minimize either the total weighted completion time or the maximum lateness. Cui and Lu (2017) extended the static problem into a dynamic case, that is, where the release times of all jobs are not equal to zero. They proposed an MIP model and a BAB method to minimize the makespan. Su and Wang (2017) investigated a single-machine scheduling problem with clean activities. In their study, the machine must be maintained once the amount of dirt in the machine exceeds the maximum allowed accumulated dirt. The problem is motivated by a specified cleaning machine in a wafer fabrication process. The objective is to minimize the total absolute deviation of the job completion time. They proposed a MIP model that could solve small problems with $n \leq 50$ and a heuristic combined with the DP method with the V-shaped rule of Kanet (1981) for obtaining near-optimal solutions quickly.

These studies considered only one scheduling objective. However, in a practical situation, decision-makers may consider multiple objectives simultaneously. Wang and

Liu (2015) dealt with parallel-machine problems with multi-resource preventive maintenance and proposed NSGA-II algorithm to simultaneously minimize the makespan and unavailability of the machine and mold. Jin et al. (2008) extended the study of Cassady and Kutanoglu (2005) to a multi-objective optimization problem to minimize maintenance costs, makespan, total weighted completion time of jobs, total weighted tardiness and machine unavailability. For solving the problem, a multi-objective genetic algorithm is proposed. Berrichi et al. (2009, 2010) considered the same problem with preventive maintenance to simultaneously minimize the makespan and system unavailability. The multi-objective ant colony optimization of Berrichi et al. (2010) improved the quality of solutions obtained by NSGA-II of Berrichi et al. (2009). Recently, an interval number theory was applied to model uncertain processing conditions; this model uses the lower and upper bounds of the interval to indicate uncertain processing conditions. Lei (2013) considered the interval job shop scheduling with flexible maintenance and presented a multi-objective artificial bee colony algorithm to minimize the interval makespan and the total interval tardiness.

Based on the related literature review, no research simultaneously considers multiple objectives and job release times on a single machine scheduling problem with PMs. Thus, we extended the study of Su and Wang (2017) to the dynamic case and consider two objective functions, i.e., the $1\left|r_j, cleaning\right|\sum w_jT_j\,,\sum C_j$ problem is considered here. To the best of our knowledge, this problem is investigated for the first time. For the problem, we developed a MIP model to find Pareto solutions for a small-size problem. In addition, the SSA algorithm with a scatter search mechanism is developed to efficiently solve the considered problem.

The considered problem is a deterministic case. As for the stochastic case, the reader may refer to the studies by Cassady and Kutanoglu (2005) and Wang and Liu (2015) for more information.

3. The operation process of a wet station in semiconductor manufacturing

In a wet station, different wafers are soaked or splashed with chemicals in the machine for a given time and subsequently cleaned with ultra-pure water. In this process, residual dirt or pollutants are left in the machine, and once the amount of accumulated dirt exceeds the tolerance limit, the quality of the products will be damaged. As a result, the wet station needs to implement maintenance or cleaning activities periodically, wherein the start of each maintenance activity is determined by the amount of accumulated dirt. Based on these characteristics of the wet station, the problem is defined as a set of jobs $\{J_1, J_2, \dots, J_n\}$ with a processing time p_j, release time r_j, due date d_j, penalty weight w_j of tardy jobs and the amount of dirt t_j left on a machine during processing. The wet station is available for processing jobs as long as the accumulated dirt does not exceed the threshold value TV. The threshold value and cleaning time or maintenance time (MT) is constant and has a known priori. Figure 1 illustrates the schedule of a problem, in which the cleaning activity occurs after a sequence of jobs have been processed on a machine, and the sequence of jobs is denoted as batch B_i. Thus, a schedule π can be treated as $\pi = (B_1, MT, B_2, MT, \dots, B_L, MT, B_{L+1})$, where L is a decision variable and $(\sum_{j=1}^{n} t_j / TV) - 1 \le L \le n - 1$. For this problem, there are two decision points: the time when a machine finished processing and the time when a job arrives. At either of these points, a decision must be made whether to implement the cleaning activity or process a current job or wait for an oncoming job.

Insert Figure 1 here

3.1 Threshold method vs dynamic programming (DP) method

This section presents a threshold method and a DP method for the execution of a cleaning activity or processing jobs. The threshold method is simple and intuitive because it assigns jobs on the machine continuously as long as the accumulated dirt does not exceed the threshold value, and it is a common method used in real-life production systems. The steps of the threshold method are described below.

- Threshold method

Step 1: Arrange the jobs in a pre-specified order; i.e., $JS = \{J_{[1]}, J_{[2]}, \dots, J_{[n]}\}$. Let i=1 and accumulated dirt (AD)=0.

Step 2: Assign jobs into the machine as long as the accumulated dirt of these scheduled jobs does not exceed the threshold value. Remove these scheduled jobs from the list of JS and denote the scheduled jobs as batch B_i.

Step 3: If there are unscheduled jobs, i.e., $JS \neq \{\emptyset\}$, let AD=0, assign a cleaning activity to the machine after batch B_i, wherein i=i+1, and go to Step 2. Otherwise, stop the method.

The threshold method gives the minimum number of PMs for the pre-specified order (JS), where jobs could be sequentially processed when the accumulated dirt in the machine is less than the threshold value. Hence, the number of cleaning activities (or PMs) is minimized, which is an advantage to the machine's utilization. However, this method is myopic because it only uses information about the current dirt residue of a job for making decisions, which may lead to poor schedules. To resolve this deficiency, there is a need to develop a procedure that is capable of considering current objective values, job status and dirt limits for making decisions. This is the motivation from the

DP method developed by Wang and Uzsoy (2002) and Chou (2007) for the batching problems. The proposed DP method in this paper is addressed below.

- The proposed DP method

First the jobs are sorted in a pre-specified order (JS), and the top job of JS are placed into the set of π initially. The partial schedule π means that k jobs are scheduled in the set of π, where $k < n$. For the partial schedule π, its solution or state could be calculated by the following formula.

$$Z(\Delta^k) = Min_{1 \le g \le k} \left\{ \left[f_g^0\left(\nabla_g^k, Z_0(\Delta^{k-g})\right), f_g^1\left(\nabla_g^k, Z_1(\Delta^{k-g})\right), f_g^2\left(\nabla_g^k, Z_2(\Delta^{k-g})\right) \right] \right\} \quad ,$$

where Δ^k is a sub-sequence of the first k jobs in partial schedule π, that is, $\Delta^k = \{J_{[1]}, J_{[2]}, \dots, J_{[k]}\}$, ∇_g^k is a sub-sequence of the last g jobs in Δ^k, i.e., $\nabla_g^k = \{J_{[k-g+1]}, J_{[k-g+2]}, \dots, J_{[k]}\}$, $Z_0(\Delta^k)$, $Z_1(\Delta^k)$ and $Z_2(\Delta^k)$ indicates the minimum makespan, total weighted tardiness (*TWT*), and total completion time (*TCT*) obtained based on the sub-sequence of Δ^k, respectively. The proposed DP method is recursively calculated by the following formulas $f_g^0\left(\nabla_g^k, Z_0(\Delta^{k-g})\right), f_g^1\left(\nabla_g^k, Z_1(\Delta^{k-g})\right), f_g^2\left(\nabla_g^k, Z_2(\Delta^{k-g})\right)$ described as follows:

$$f_\delta^0(\nabla_g^k, Z_0(\Delta^{k-g})) = \begin{cases} Max(Z_0(\Delta^0), r_{[1]}) + p_{[1]} & if \quad \delta = 1, k = g, \sum_{\rho \in \nabla_g^k} t_\rho \le TV \\ Max(Z_0(\Delta^{k-g}) + TC, r_{[k-g+1]}) + p_{[k-g+1]} & if \quad \delta = 1, k > g, \sum_{\rho \in \nabla_g^k} t_\rho \le TV \\ Max(f_{\delta-1}^0(\nabla_g^k, Z_0(\Delta^{k-g})), r_{[k-g+\delta]}) + p_{[k-g+\delta]} & if \quad \delta > 1, \delta \le g, \sum_{\rho \in \nabla_g^k} t_\rho \le TV \\ \infty & if \quad \delta > 1, \delta \le g, \sum_{\rho \in \nabla_g^k} t_\rho > TV \end{cases}$$

$$f_{\delta}^{1}(\nabla_{g}^{k}, Z_{1}(\Delta^{k-g}))$$
$$= \begin{cases} Z_{1}(\Delta^{k-g}) + Max\{f_{1}^{0}(\nabla_{g}^{k}, Z_{0}(\Delta^{k-g})) - d_{[k-g+1]}, 0\} \times w_{[k-g+1]} & if \;\; \delta = 1 \\ f_{\delta-1}^{1}(\nabla_{g}^{k}, Z_{1}(\Delta^{k-g})) + Max\left\{f_{\delta}^{0}(\nabla_{g}^{k}, Z_{0}(\Delta^{k-g})) - d_{[k-g+\delta]}, 0\right\} \times w_{[k-g+\delta]} & if \;\; \delta > 1, \delta \le g \\ \infty & if \;\; \sum_{\rho \in \nabla_{g}^{k}} t_{\rho} > TV \end{cases}$$

$$f_{\delta}^{2}(\nabla_{g}^{k}, Z_{2}(\Delta^{k-g}))$$
$$= \begin{cases} Z_{2}(\Delta^{k-g}) + f_{1}^{0}(\nabla_{g}^{k}, Z_{0}(\Delta^{k-g})) & if \;\; \delta = 1 \\ f_{\delta-1}^{2}(\nabla_{g}^{k}, Z_{2}(\Delta^{k-g})) + f_{\delta}^{0}(\nabla_{g}^{k}, Z_{0}(\Delta^{k-g})) & if \;\; \delta > 1, \delta \le g \\ \infty & if \;\; \sum_{\rho \in \nabla_{g}^{k}} t_{\rho} > TV \end{cases}$$

An example shown in Table 1 illustrates the threshold method and the proposed DP method. Suppose a sequence of jobs is $\{J_7, J_2, J_6, J_3, J_1, J_5, J_8, J_4, J_9\}$, the final schedule is $\{(J_7, J_2, J_6)$,*MT*, (J_3, J_1), *MT*, (J_5, J_8), *MT*, $(J_4, J_9)\}$ obtained by the threshold method, and objective values of $\sum w_j T_j$ and $\sum C_j$ are 381 and 533, respectively. Under the same sequence of jobs, the final schedule is $\{(J_7)$,*MT*, (J_2, J_6, J_3, J_1),*MT*, (J_5, J_8),*MT*, $(J_4, J_9)\}$ obtained by the proposed *DP* method, and the objective values of $\sum w_j T_j$ and $\sum C_j$ are 276 and 497, respectively. This result reveals that the proposed DP is superior to the threshold method because the solution obtained by the latter is dominated. Furthermore, the solutions obtained by the proposed DP method also belong to the set of Pareto solutions. Although the proposed DP method is more complicated than the threshold method, the obvious merit of it is that the DP method can predict a better solution because more information from the current stage and previous stage are used.

Table 1. An instance with nine jobs where *TV*=15 and *MT*=10

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
r_j	11	15	18	12	6	10	0	11	19
p_j	5	6	9	7	10	10	8	8	10
d_j	40	39	43	38	42	40	28	44	48
t_j	3	3	6	3	8	2	9	7	5
w_j	3	4	5	1	3	4	4	2	1

3.2 Mixed-integer programming (MIP) model

This section describes the MIP model for the $1|r_j, cleaning|\sum w_j T_j, \sum C_j$ problem. Relative known variables are defined below.

n	: number of jobs.
TV	: threshold value of accumulated dirt.
MT	: maintenance time (clean time).
M	: a sufficiently large integer.
r_j	: release time of job J_j; $j = 1,2,3, \dots n$.
p_j	: processing time of job J_j.
d_j	: due date of job J_j.
t_j	: dirt left in a wet station after processing job J_j.
w_j	: penalty weight of job J_j if the job is tardy.

Some auxiliary variables and two sets of 0-1 decision variables are defined as follows.

ST_k	: start time for processing on position k in the sequence.
PT_k	: processing time at position k in the sequence.
CT_k	: completion time at position k in the sequence.
C_j	: completion time of job J_j.

T_j : tardiness time of job j; that is, $T_j = \max(0, C_j - d_j)$.

Q_k : accumulated dirt left at the point of position k in the sequence.

$$X_{jk} = \begin{cases} 1 & \mathit{if} \text{ job } J_j \text{ is assigned on position } k \text{ in the sequence} \\ 0 & \mathit{otherwise}. \end{cases}$$

$$Y_k = \begin{cases} 1 & \text{if clean activity occurs on position } k \text{ in the sequence} \\ 0 & \mathit{otherwise}. \end{cases}$$

The MIP formulation is detailed below.

$$Min. \quad \sum_{j=1}^{n} w_j T_j \ , \ \sum_{j=1}^{n} C_j \tag{1}$$

Subject to:

$$\sum_{k=1}^{n} X_{jk} = 1 \qquad \forall \quad j = 1,2,3,\dots,n \tag{2}$$

$$\sum_{j=1}^{n} X_{jk} = 1 \qquad \forall \quad k = 1,2,3,\dots,n \tag{3}$$

$$ST_k \geq \sum_{j=1}^{n} r_j \cdot X_{jk} \qquad \forall \quad k = 1,2,3,\dots,n \tag{4}$$

$$PT_k = \sum_{j=1}^{n} p_j \cdot X_{jk} \qquad \forall \quad k = 1,2,3,\dots,n \tag{5}$$

$$ST_k \geq CT_{k-1} + MT \cdot Y_{k-1} \qquad \forall \quad k = 2,3,4,\dots,n \tag{6}$$

$$CT_k \geq ST_k + PT_k \qquad \forall \quad k = 1,2,3,\dots,n \tag{7}$$

$$Q_1 = \sum_{j=1}^{n} t_j \cdot X_{j1} \tag{8}$$

$$\begin{cases} Q_{k-1} + \sum_{j=1}^{n} t_j \cdot X_{jk} \leq Q_k + M \cdot Y_{k-1} \\ \sum_{j=1}^{n} t_j \cdot X_{jk} \leq Q_k + M \cdot (1 - Y_{k-1}) \end{cases} \qquad \forall \quad k = 2,3,4,\dots,n \tag{9}$$

$$Q_k \leq TV \qquad \forall \quad k = 1,2,3,\dots,n \tag{10}$$

$$C_j \geq CT_k + M \cdot (X_{jk} - 1) \qquad \forall \quad \begin{cases} j = 1,2,3,\dots,n \\ k = 1,2,3,\dots,n \end{cases} \tag{11}$$

$$T_j \geq C_j - d_j \qquad \forall \quad j = 1,2,3,\dots,n \tag{12}$$

$$X_{jk}, Y_{jk} = \{0,1\} \qquad \forall \quad j,k = 1,2,3,\dots,n \tag{13}$$

$$C_j, T_j \geq 0 \qquad \forall \quad j = 1,2,3,\dots,n \tag{14}$$

In this model, the objective function simultaneously minimizes the total weighted tardiness and the total completion time, as shown in equation (1). Constraint sets (2) and (3) ensure that each job can only be placed at one position, and each position can only be occupied by one job for processing. Constraint sets (4) and (5) define the start time for the processing of the *k*-position job in the sequence and the processing time of the *k*-position job, respectively. Constraint set (6) ensures that the start time for processing of the *k*-position job should be logically equal to or greater than the sum of the completion time of the (*k*-1)-position job and the cleaning time if the cleaning activity is performed immediately after completion of the (*k*-1)-position job. Constraint set (7) defines the completion time of the *k*-position job. Constraint sets (8) and (9) define the dirt accumulation between the completion of the last cleaning activity and the completion of the *k*-position job. Constraint set (10) ensures that dirt accumulation in each batch cannot exceed the threshold value. Constraint set (11) defines the completion time of job j. Constraint set (12) defines the tardy job when the completion time of job j is greater than its due date. Constraint set (13) and (14) represent the binary and non-negativity constraints.

3.2.1 Complexity of the problem

The traditional single machine scheduling problem with job release time that considers either the total weighted tardiness or the total completion time minimization is proven to be *NP*-hard (Pinedo 2008); therefore, the problem in this paper is also *NP*-hard. In our model, a total of $(n^2 + 10n - 1)$ constraints, and a total of $(n^2 + 7n + 2)$ variables exist, where $(n^2 + n)$ is binary.

3.2.2 The solving procedure of bi-objective MIP model

To find all non-dominant solutions, we applied the ILOG script in the bi-objective MIP model to approach the solution space. The procedure of the bi-objective MIP model implemented using the ILOG script is explained in the following steps.

Step 1: To find the optimal *TWT* value without considering the objective of *TC*, and let the optimal *TWT* be equal to *twt_lb*.

Step 2: To find the optimal *TC* value without considering the objective of *TWT*, and let the optimal *TC* be equal to *tc_lb*.

Step 3: Add the extra constraint of *TC*=*tc_lb* using the script function into the MIP model, and find the *TWT* value. Let the *TWT* value be equal to *twt_ub*, and one of the non-dominated solutions is (*twt_ub*, *tc_lb*), which is also an extrema point.

Step 4: Add the extra constraint of *TWT*=*twt_lb* using the script function into the MIP model and find the *TC* value. Let the *TC* value be equal to *tc_ub*, and one of the non-dominated solutions is (*twt_lb*, *tc_ub*), which is also an extrema point.

Step 5: When the two extrema points obtained by steps 3 and 4 are the same, that is, (*twt_ub*, *tc_lb*)=(*twt_lb*, *tc_ub*), stop the procedure, and there is only one non-dominated solution.

Step 6: Based on the two extrema points, add the extra constraint of $((k - tc\_lb) \times TWT + (twt\_ub - twt\_lb) \times TC \leq (k \cdot twt\_ub) - (tc\_lb \cdot twt\_lb)\ )$ into the MIP model and implement the MIP model to find all non-dominated solutions using each of the *k* values in the range [*tc_lb*, *tc_ub*] with increments of 1.

4. SA for multi-objective problems

The SA algorithm is a computational stochastic technique to obtain optimal or near-optimal solutions by applying a probability function that helps it explore solutions to avoid falling into a local optimal trap on a search space (Kirkpatrick et al. 1983).

Recently, SA algorithms have been powerful tools in solving complex optimization problems, including multi-objective combinatorial problems. Czyzak and Jaszkiewicz (1998) proposed a Pareto simulated annealing (PSA) to generate a Pareto front for a multiple-objective knapsack problem. Varadharajan and Rajendran (2005) considered a flowshop scheduling problem wherein the objective is to simultaneously minimize the makespan and the total flowtime. For solving the problem, an SA is developed wherein two parameters are used to control search directions. Tekinalp and Karsli (2007) developed a novel multi-objective SA by using a fitness of population and an adaptive cooling schedule to generate a Pareto front for continuous optimization problems. Bandyopadhyay et al. (2008) developed an SA-based multi-objective optimization algorithm (AMOSA) that incorporates the concept of an archive to provide Pareto-solutions. Their results showed that the performance of AMOSA was better than that of the two well-known multi-objective evolutionary algorithms, namely, PAES (Knowles and Corne, 2000) and NSGA-II (Deb et al. 2002).

Inspired by the successful applications of SA in multi-objective optimization problems, a novel multi-objective SA framework, named scatter SA (SSA), was proposed for the $1\left|r_j, cleaning\right|\sum w_j T_j, \sum C_j$ problem. SSA comprises two main parts: the basic framework of SA, with a scatter search mechanism and a Pareto archive. The scatter search mechanism intends for each solution to simultaneously search different objective spaces by changing the search direction, as shown in Figure 2. From Figure 2, each current sequence comes with 11 different composite solutions by the weights of α and β, and the combination of α and β implies the directions on the search space. When the current sequence generates a neighbor sequence, each comparison for the current composite solution and neighborhood composite solution is on the same the combination of α and β (for example, α=0.1 and β=0.9). If the replacement occurs, then the neighbor sequence is better than the current sequence in this direction. From this process, it is evident that we could maintain better potential solutions for each combination of α and β or for each direction on the search space. Regarding the Pareto archive, it is a dynamic and auto-updating mechanism that is similar to the archive investigated by Bandyopadhyay et al. (2008), who attempted to consider each solution so that non-dominated solutions obtained during the search are inserted into this archive.

Insert Figure 2 here

Applying SA to solve the $1\left|r_j, cleaning\right|\sum w_jT_j, \sum C_j$ problem, the representation of a solution and the acceptance probability must be individually introduced, whereas the neighbor solutions generation and parameter settings are included in the entire SSA procedure.

4.1 Solution representation and neighborhood generation

As mentioned in Section 3, the $1\left|r_j, cleaning\right|\sum w_jT_j, \sum C_j$ problem is solved in two stages: first, sort jobs in some arbitrary order, and then apply the proposed DP method to form batches. As a result, for any sequence of jobs, the schedule and objective values to the corresponding sequence of jobs are obtained by the proposed DP method as shown in Figure 3. Neighbor solution generation is based on the job sequence that adopts a swap method for generating a new sequence of jobs and then applies the proposed DP method to obtain objective solutions (Figure 4).

Insert Figure 3 here

Insert Figure 4 here

4.2 Prompt archive and acceptance probability

In applying SA to solve bi-objective scheduling problems with minimum objectives, three cases may be encountered when comparing the generated neighbor solution (π^n) with the current solution (π^c), as follows:

Case 1: For all objectives where $f^i(\pi^n) \leq f^i(\pi^c)$

Case 2: An objective *i* exists at least, where $f^i(\pi^n) < f^i(\pi^c)$

Case 3: For all objectives where $f^i(\pi^n) > f^i(\pi^c)$

In case 1, the neighbor solution shall be accepted based on the definition of Pareto-optimal solution by Tamaki et al (1996), and the current solution is replaced by the neighbor solution, that is, $\pi^c = \pi^n$. Once replacement occurs, the new current solution is compared with all solutions in the archive. If some solutions in the archive are dominated by the new current solution, then the new current solution is added into the archive, and all inferior solutions are deleted. When the new current solution cannot be dominated by any solution in the archive, the new current solution is also added into the archive. Moreover, if the new current solution can be dominated by any solution in the archive, the archive does not need to be changed. The actions for case 1 are described in Table 2. Using this prompt process, the archive always contains the final non-dominated solutions. For case 2, a compound objective function is used to calculate the objective value, that is, $Z \quad = \alpha f^1(\pi \quad) + \beta f^2(\pi \quad)$, and then compares the current solution with the new one. The details for case 2 are listed in Table 3.

With respect to case 3, SA uses the acceptance probability to decide whether the worse solution is accepted or not, as mentioned in Table 4. This feature is important to enable the SA to explore solutions to prevent being trapped in a local optimum. For multi-objective solutions, different formulas for the acceptance probability are proposed by SMOSA (Suppapitnarm et al., 2000), UMOSA (Ulungu et al. 1998) and PSA (Czyzak and Jaszkiewicz, 1998). Given that SSA uses different values of (α, β) to guide the search direction, the conflicting objectives are integrated into a weighted objective solution according to (α, β), as follows:

$$\text{Probability } (p) = \exp\left(\frac{-\Delta E}{T}\right)$$

$$\text{where } \Delta E = f(\pi^n) - f(\pi^c) \text{ and } f(\pi^n) = \alpha f^1(\pi^n) + \beta f^2(\pi^n)$$

Table 2. The actions for case 1 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 1
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) < f^1(\pi^c)$ and $f^2(\pi^n) < f^2(\pi^c)$
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	
Action 1			$\pi^c = \pi^n$
Action 2	Compare π^c with all solutions in the archive, and one of the following three results is satisfied		
	Result 1	Delete all solutions inferior to π^c in the archive and add π^c to the archive	
	Result 2	Add π^c to the archive if π^c is a new non-dominated solution	
	Result 3	The archive is not changeable if π^c is dominated	

Table 3. The actions for case 2 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 2
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) \le f^1(\pi^c)$ and $f^2(\pi^n) > f^2(\pi^c)$ or
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	$f^1(\pi^n) > f^1(\pi^c)$ and $f^2(\pi^n) \le f^2(\pi^c)$
Action 1	$Z^n = \alpha f^1(\pi^n) + \beta f^2(\pi^n)$, $Z^c = \alpha f^1(\pi^c) + \beta f^2(\pi^c)$		
Action 2	if $(Z^n < Z^c)$ implement Action 2 in the Case 1		
	if $(Z^n \ge Z^c)$ implement Action 1 in the Case 3		

Table 4. The actions for case 3 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 3
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) > f^1(\pi^c)$ and $f^2(\pi^n) > f^2(\pi^c)$
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	
Action 1	Use acceptance probability to decide whether $\pi^c = \pi^n$ or not		
Action 2	None		

4.3 The proposed SSA

In the SSA, the cooling scheme includes (1) the cooling rate and (2) the number of iterations at the same temperature. A smaller cooling rate value indicates that the temperature decreases quickly which will lead the SSA to converge quickly to a poor result. In contrast, a higher cooling rate value causes a high SSA computation time to obtain the results. Regarding the number of iterations at the same temperature, its purpose is to make the SSA sufficiently disturbed at each temperature. Based on our preliminary experiments, the cooling rate and the number of iterations at the same temperature is 0.98 and 100, respectively. The proposed SSA algorithm is introduced using a step-by-step procedure as follows.

Step 0: Generate a sequence (π^0) randomly and add the solution into the archive. Set *x*=0 and the cooling rate is equal to 0.98.

Setp 1: Calculate the objective values, total weighted tardiness and total completion time, for sequence π^0 using the proposed DP method, that is, $f^1(\pi^0)$ and $f^2(\pi^0)$, respectively.

Step 2: Do while $x \leq 10$

Step 2.1: Set the initial sequence π^0 be the current sequence π_x^c

```
                Step 2.1.1: Set $\alpha = 0.0 + 0.1x$, $\beta = 1.0 - \alpha$
                Setp 2.1.2: Scatter the solution by doing the following
                    {
                    Calculate the weighted objective value for the sequence $\pi^0$, that is,
                    $Z_x^c = \alpha f^1(\pi^0) + \beta f^2(\pi^0)$.
                    Set the initial temperature $T_x$ equal to $1.618 \times Z_x^c$, where 1.618 is the
                    golden ratio.
                    }
        Step 2.2: $x = x + 1$
        End do
Step 3: Set $x$=0, temp_counter=0
Step 4: Do while $x \leq 10$
        Step 4.1: Generate a neighbor sequence $\pi_x^n$ from $\pi_x^c$ using the swapping method,
                temp_counter= temp_counter+1
        Step 4.2: Apply the proposed DP method for the sequence $\pi_x^n$ to obtain objective
                values of $f^1(\pi_x^n)$ and $f^2(\pi_x^n)$.
        Step 4.3: Compare the objective values of $f^1(\pi_x^n)$ and $f^2(\pi_x^n)$ with each of
                Pareto-optimal solution in the archive. If Cases 1 and 2 are satisfied,
                then add the sequence $\pi_x^n$ with objectives $f^1(\pi_x^n)$ and $f^2(\pi_x^n)$ into the
                archive.
        Step 4.4: Set $y$=0
        Step 4.5: Do while $y \leq 10$
                $\alpha = 0.0 + 0.1y, \beta = 1.0 - \alpha$
                Calculate the weighted objective value by $(\alpha, \beta)$ for the sequence $\pi_x^n$,
                that is, $Z_y^n = \alpha f^1(\pi_x^n) + \beta f^2(\pi_x^n)$.
                If ($Z_y^n < Z_y^c$), then
                        $Z_y^c = Z_y^n, \quad f^1(\pi_y^c) = f^1(\pi_x^n),$
                        $f^2(\pi_y^c) = f^2(\pi_x^n), \quad \pi_y^c = \pi_x^n$
                Else
                        Generate a random number $\gamma$
                        $\Delta\mathrm{E} = Z_y^n - Z_y^c$
                        $p = \exp\left(\frac{-\Delta E}{T_y}\right)$
                        If ($\gamma \le p$), then
                                $Z_y^c = Z_y^n, \quad f^1(\pi_y^c) = f^1(\pi_x^n),$
                                $f^2(\pi_y^c) = f^2(\pi_x^n), \quad \pi_y^c = \pi_x^n$
                        Endif
                        If temp_counter $\ge$100, then
                                $T_y = 0.98 \times T_y$  /* 0.98 is cooling rate
                                temp_counter =0
                        Endif
                Endif
                $y$=$y$+1
            End Do
        Step 4.6: $x = x + 1$
        End Do
Step 5: Go to Step 3 until the criterion for stopping is true
```

5. Computational experiments

To verify the performance of the proposed SSA algorithm, two sets of computational experiments are conducted: one involving small-size problems with different job numbers (*n*=5, 7, 9, 11 and 13) and the other involving large-size problems (*n*=20, 30, 40,…, and 100). Six algorithms, namely, MIP, enumerative approach with the threshold method (EAT), enumerative approach with DP (EAD), SMOSA (Suppapitnarm et al. 2000), NSGA-II (Deb et al. 2002), and SSA, are compared in the first experiment. SMOSA uses a return-to-base strategy to restart the search by randomly selecting a solution from the archived solutions, and the acceptance probability formulation is based on an annealing schedule with multiple temperatures. In the second experiment, SMOSA, NSGA-II, and SSA are compared. For each instance, the three meta-heuristic algorithms (NSGA-II, SMOSA and SSA) are executed five times, and the best result is obtained after five runs. The MIP model is implemented in IBM ILOG CPLEX Optimization Studio Version 12.6.1. The algorithms were coded in C++ and implemented on a PC Xeon E5-1620 CPU with 3.6 GHz and 12 GB RAM.

In both experiments, all test problems are generated randomly. For each job, processing time, release time, due date, weights, and dirt amount of the jobs are determined from a discrete uniform distribution in the following ranges: [4, 12], [0, 20], $(r_j + p_j) + \left(\left\lceil {\sum t_j}/{TV} \right\rceil \times 10 + \sum p_j\right) \times 0.75$, [1, 9], and [2, 10], respectively. The *TV* of the accumulated dirt and the maintenance time (*MT*) are 15 and 10, respectively. For each problem, a sample of 50 instances was provided, thus a total of 700 instances are generated.

In this work, three performance measures to examine the algorithms are adopted. These performance indexes are generally considered in the literature on multi-objective optimization problems and are defined as follows:

- *RNI* (Ratio of Non-dominated Individuals, Tan et al. 2002): the quality performance of the algorithms is defined as $|P_{A(B)} \cap P_C|/|P_C|$, where P_A and P_B are the set of Pareto-solutions obtained by algorithms *A* and *B*, respectively. For the small-size problem, P_C is the set of non-dominated solutions obtained by the MIP model, whereas for the large-size problem, P_C is the set of non-dominated solutions obtained after combining P_A and P_B. $|X|$ denotes the number of Pareto-solutions in set *X*.
- *HV* (Hypervolume) (Veldhuizen and Van and Lamont 1999): implies the distance of the solutions obtained by algorithm *A* and *B* are. This measure is modified from Kashan et al. (2010), and the concept is shown in Figure 5. The Pareto-solutions are obtained by the MIP model or after combining algorithms A and B. Each of the two extreme points from the Pareto-solutions is multiplied by 1.25, and are the reference points R1 and R2, respectively. *HV* is defined as $HV(P_{A(B)})/HV(P_c)$ and $0 < HV \leq 1$.

Insert Figure 5 here

- *GD* (Generational distance) (Veldhuizen et al. 1999): evaluates the average distance of the solutions of algorithm *A* from P^*, where P^* is the set of Pareto-solutions, as illustrated in Figure 6. The algorithm with a small *GD* value is preferred.

Insert Figure 6 here

Table 5 compares the average *RNI*, *HV* and *GD* obtained by EAT, EAD, SMOSA, NSGA-II, and SSA with those obtained by the MIP model for the small-size problems. In this comparison, the set of Pareto-solutions was obtained by the MIP model except for *n*=13. For *n*=13, the reference set is all non-dominated solutions obtained after combining EAT and EAD. Overall, the total average values obtained by EAD were 0.993, 1.000 and 0.489 for *RNI*, *HV*, and *GD*, respectively, and the corresponding average values of 0.993, 1.000, and 0.489 were obtained by SSA; the performances of EAD and SSA are the same. This result was anticipated because scatter search mechanisms by changing values of (α, β) allow the proposed SSA algorithm to explore more potential solutions. Furthermore, EAT is expected to be slightly worse than the other algorithms. This finding shows that EAD benefits from the advantages of the proposed DP method described in Section 3.1.

Considering the computational times, NSGA-II, SMOSA and SSA are terminated as the computation time reaches $n \times 0.1$ second, whereas the computational effort required by MIP, EAT, and EAD considerably increase as the number of jobs increases because the three methods belong to the exhausted method.

Table 5. Average results obtained by the four algorithms compared with those obtained by the MIP model

Jobs	Ave. *RNI*					Ave. *HV*				
	EAT	EAD	SMOSA	NSGA-II	SSA	EAT	EAD	SMOSA	NSGA-II	SSA
5	0.911	0.978	0.978	0.978	0.978	0.974	0.999	0.999	0.999	0.999
7	0.927	0.990	0.990	0.990	0.990	0.989	1.000	1.000	1.000	1.000
9	0.914	0.998	0.998	0.997	0.998	0.997	1.000	1.000	1.000	1.000
11	0.910	0.998	0.935	0.998	0.998	0.996	1.000	0.999	1.000	1.000
13	0.947	1.000	0.746	0.995	1.000	1.000	1.000	0.995	1.000	1.000
Global Ave.	0.922	0.993	0.929	0.992	0.993	0.991	1.000	0.999	1.000	1.000

Table 5. Average results obtained by the four algorithms compared with those obtained by the MIP model (Cont.)

Jobs	Ave. *GD*				
	EAT	EAD	SMOSA	NSGA-II	SSA
5	1.794	0.360	0.360	0.360	0.360
7	1.062	1.440	1.440	1.440	1.440
9	16.120	0.082	0.082	0.082	0.082
11	7.239	0.564	14.920	0.564	0.564
13	8.252	0.000	55.306	1.118	0.000
Global Ave.	6.893	0.489	14.422	0.713	0.489

For these two different versions of SA algorithms (SMOSA and SSA), we chose an instance from problems with n=20, and the sets of approximate non-dominated solutions obtained by SMOSA and SSA for the chosen instance are shown in Figure 7. Two Pareto fronts obtained by SMOSA and SSA are displayed by means of dots and a line, respectively. From Figure 7, it is apparent that some solutions obtained by SMOSA were dominated by those of SSA, and some dots that are on the curve indicate that the solutions of SMOSA are equal to those of SSA. Both SSA and SMOSA use the $n \times 0.1$ secnods time limit as the terminal condition. According to this finding, the scatter search mechanism of SSA is beneficial in exploring solutions.

Insert Figure 7 here

We also examine the average performances of SMOSA, NSGA-II, and SSA using the indexes of average *RNI*, *HV* and *GD*. The benchmark set of the Pareto front is obtained after combining those solutions obtained by SMOSA, NSGA-II, and SSA. Table 6 shows that the global average of RNI is 0.711, 0.822, and 0.955 for SMOSA, NSGA-II, and SSA, respectively; this result implies that the great majority of non-dominated solutions in the benchmark set of the Pareto front could be obtained by SSA. For *HV*, the global averages for SMOSA, NSGA-II, and SSA are 0.983, 0.998, and

1.000, respectively. For *GD*, the global averages for SMOSA, NSGA-II, and SSA are 4155.430, 6317.209, and 1451.823, respectively. Both *HV* and *GD* measure the difference between the obtained Pareto front and the benchmark set of Pareto front. However, the *GD* is more sensitive than the *HV* because the *GD* calculation is based on distances between individual solutions, whereas the *HV* calculation is based on areas where identifying slight differences between solutions is difficult. Whether *RNI*, *HV* or *GD* or not, we observed that SSA is clearly superior to SMOSA and NSGA-II. This finding is expected because the SSA search covers a large range of solution spaces via search-direction changing compared with SMOSA.

Table 6. Comparison of results of SMOSA and SSA for the large-size problems

n	Ave. *RNI*			Ave. *HV*			Ave. *GD*		
	P_{SMOSA}	NSGA-II	P_{SSA}	P_{SMOSA}	NSGA-II	P_{SSA}	P_{SMOSA}	NSGA-II	P_{SSA}
20	0.457	0.8432	0.997	0.979	0.999	1.000	336.259	163.308	0.615
30	0.320	0.6039	0.951	0.971	0.998	1.000	2179.440	1163.776	57.912
40	0.501	0.645	0.928	0.982	0.998	1.000	3500.320	2410.338	175.517
50	0.690	0.7565	0.921	0.982	0.997	1.000	6110.820	5116.24	684.456
60	0.824	0.864	0.951	0.983	0.998	1.000	4819.559	4922.443	668.196
70	0.872	0.904	0.940	0.984	0.998	1.000	4888.314	6433.976	2522.108
80	0.907	0.924	0.972	0.985	0.998	1.000	4678.073	10350.480	1571.080
90	0.906	0.917	0.963	0.987	0.997	1.000	4151.309	11440.360	3691.800
100	0.928	0.938	0.971	0.990	0.997	1.000	6734.772	14853.960	3694.720
Global Ave.	0.711	0.822	0.955	0.983	0.998	1.000	4155.430	6317.209	1451.823

6. Conclusions

Motivated by the wet station operation in semiconductor fabrications, this is the first work to consider the bi-objectives of minimizing total weighted tardiness and total completion time simultaneously on a single machine with a dynamic job release time and flexible PM. For resolving the problem, we addressed the intuitive threshold method and DP method to exhibit the influence of PM on scheduling. We also proposed an MIP model and the SSA algorithm, where the proposed DP method is incorporated to obtain better objective values for each sequence of jobs. In addition, the SSA applied the scatter search mechanism by changing the search direction to be able to explore solutions extensively. Two tests were used to examine the performances of the different algorithms. In the first experiment, five approaches of EAT, EAD, SMOSA, NSGA-II, and SSA are compared with MIP, and the results showed that EAD is better than EAT. This is because the former applied the proposed DP method, and non-dominated solutions could be obtained by EAD and SSA. For large-size problems, SSA was compared with SMOSA and NSGA-II. The three meta-heuristic algorithms are terminated when the computational time reaches n×0.1 seconds. The computational result shows that the proposed SSA is superior to SMOSA and NSGA-II in terms of the three well-known performance measures mentioned in the literature. The majority of non-dominated solutions in the set of approximate Pareto front were obtained by SSA. Furthermore, the results also revealed that using GD as a performance index is more sensitive than using HV.

Based on the computation results, the proposed SSA successfully solves the considered problem. With the efficiency of SSA, it could also be modified easily to address different shop environments, such as parallel machines or flow shop, in the future. Different multiple objectives can also be investigated for this considered scheduling problem. Furthermore, different meta-heuristic algorithms, such as particle swarm optimization (PSO) algorithms, genetic algorithms (GAs), etc., can be developed for the considered problem.

Appendix

Table A.1 All non-dominated solutions obtained by six algorithms for the tenth instance of the small-size problem

		n=5	n=7	n=9	n=11
MIP	No.	2	9	4	9
	The set of	(78, 154);	(202, 296);	(217, 427);	(729, 619);
	Pareto	(86, 149)	(206, 293);	(233, 421);	(736, 590);
	solutions		(208, 286);	(248, 420);	(749, 585);
			(216, 284);	(284, 419)	(766, 584);
			(260, 282);		(769, 581);
			(463, 281);		(786, 580);
			(490, 280);		(787, 566);
			(494, 277);		(813, 560);
			(514, 274)		(828, 558)
EAT	No.	2	9	4	6
	The set of	(78, 154);	(202, 296);	(217, 427);	**(753, 625)**;
	Pareto	(86, 149)	(206, 293);	(233, 421);	**(761, 619);**
	solutions		(208, 286);	(248, 420);	**(780, 592);**
			(216, 284);	(284, 419)	(787, 566);
			(260, 282);		(813, 560);
			(463, 281);		(828, 558)
			(490, 280);		
			(494, 277);		
			(514, 274)		
EAD	No.	2	9	4	9
	The set of	(78, 154);	(202, 296);	(217, 427);	(729, 619);
	Pareto	(86, 149)	(206, 293);	(233, 421);	(736, 590);
	solutions		(208, 286);	(248, 420);	(749, 585);
			(216, 284);	(284, 419)	(766, 584);
			(260, 282);		(769, 581);
			(463, 281);		(786, 580);
			(490, 280);		(787, 566);
			(494, 277);		(813, 560);
			(514, 274)		(828, 558)
SMOSA	No.	2	9	4	9
	The set of	(78, 154);	(202, 296);	(217, 427);	(729, 619);
	Pareto	(86, 149)	(206, 293);	(233, 421);	(736, 590);
	solutions		(208, 286);	(248, 420);	(749, 585);
			(216, 284);	(284, 419)	(766, 584);
			(260, 282);		(769, 581);
			(463, 281);		(786, 580);
			(490, 280);		(787, 566);
			(494, 277);		(813, 560);
			(514, 274)		(828, 558)
NSGA-II	No.	2	9	4	9
	The set of	(78, 154);	(202, 296);	(217, 427);	(729, 619);
	Pareto	(86, 149)	(206, 293);	(233, 421);	(736, 590);
	solutions		(208, 286);	(248, 420);	(749, 585);
			(216, 284);	(284, 419)	(766, 584);
			(260, 282);		(769, 581);
			(463, 281);		(786, 580);
			(490, 280);		(787, 566);
			(494, 277);		(813, 560);
			(514, 274)		(828, 558)
SSA	No.	2	9	4	9
	The set of	(78, 154);	(202, 296);	(217, 427);	(729, 619);
	Pareto	(86, 149)	(206, 293);	(233, 421);	(736, 590);
	solutions		(208, 286);	(248, 420);	(749, 585);
			(216, 284);	(284, 419)	(766, 584);

		(260, 282); (463, 281); (490, 280); (494, 277); (514, 274)	(769, 581); (786, 580); (787, 566); (813, 560); (828, 558)

Remark: Dominated solutions is bold font in Table A.1

Table A.1 All non-dominated solutions obtained by six algorithms for the tenth instance of the small-size problem (Cont.)

		n=13
MIP	No.	48
	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); (1108, 927); (1119, 926); (1125, 924); (1141, 923); (1145, 922); (1151, 920); (1162, 919); (1178, 918); (1217, 917) (1242, 916); (1250, 914); (1264, 911); (1265, 909); (1274, 906); (1286, 905); (1292, 900); (1303, 899); (1309, 897); (1310, 895); (1327, 893); (1362, 892); (1401, 891); (1480, 890)
EAT	No.	46
	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); **(1108, 929); (1119, 928); (1125, 926); (1145, 924); (1151, 922); (1162, 921); (1201, 920); (1231, 918);** (1250, 914); (1264, 911); (1265, 909); (1274, 906); (1286, 905); (1292, 900); (1303, 899); (1309, 897); (1310, 895); (1327, 893); (1362, 892); (1401, 891); (1480, 890)
EAD	No.	48
	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); (1108, 927); (1119, 926); (1125, 924); (1141, 923); (1145, 922); (1151, 920); (1162, 919); (1178, 918); (1217, 917) (1242, 916); (1250, 914); (1264, 911); (1265, 909); (1274, 906); (1286, 905); (1292, 900); (1303, 899); (1309, 897); (1310, 895); (1327, 893); (1362, 892); (1401, 891); (1480, 890)
SMOSA	No.	40

	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); **(1108, 930); (1122, 929); (1131, 928); (1145, 924); (1217, 922); (1226, 921); (1242, 918); (1264, 915); (1270, 913); (1290, 909); (1291, 908); (1333, 903); (1393, 897); (1414, 892); (1493, 891)**
NSGA-II	No.	47
	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); (1108, 927); (1119, 926); (1125, 924); (1141, 923); (1145, 922); (1151, 920); (1162, 919); (1178, 918); **(1242, 917)**; (1250, 914); (1264, 911); (1265, 909); (1274, 906); (1286, 905); (1292, 900); (1303, 899); (1309, 897); (1310, 895); (1327, 893); (1362, 892); (1401, 891); (1480, 890)
SSA	No.	48
	The set of Pareto solutions	(825, 1054); (828, 1053); (837, 989); (843, 987); (860, 984); (869, 983); (894, 981); (897, 980); (906, 979); (946, 978); (952, 976); (953, 965); (959, 963); (976, 960); (982, 949); (983, 947); (989, 945); (1005, 944); (1006, 942); (1015, 941); (1054, 940); (1095, 939); (1100, 938); (1101, 937); (1106, 936); (1108, 927); (1119, 926); (1125, 924); (1141, 923); (1145, 922); (1151, 920); (1162, 919); (1178, 918); (1217, 917); (1242, 916); (1250, 914); (1264, 911); (1265, 909); (1274, 906); (1286, 905); (1292, 900); (1303, 899); (1309, 897); (1310, 895); (1327, 893); (1362, 892); (1401, 891); (1480, 890)

References

Bandyopadhyay, S., Saha, S., Maulik, U. & Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: AMOSA. *IEEE Transactions on Evolutionary Computation*, 12, 269-283.

Berrichi, A., Amodeo, L., Yalaoui, F., Chatelet, E., & Mezghiche, M. (2009). Bi-objective optimization algorithms for joint production and maintenance scheduling: application to the parallel machine problem. *Journal of Intelligent Manufacturing*, 20, 389-400.

Berrichi, A., Yalaoul, F., Amodeo, L., & Mezghiche, M. (2010). Bi-objective ant colony optimization approach to optimize production and maintenance scheduling. *Computers and Operations Research*, 37, 1584-1596.

Cassady, C. R., & Kutanoglu, E. (2005). Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Transactions on Reliability*, 54, 304-309.

Chen, J.S. (2006). Optimization models for the machine scheduling problem with a single flexible maintenance activity. *Engineering Optimization*, 38, 53-71.

Chen, J. S. (2008). Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research*, 190, 90-102.

Cheng, T.C.E., Hsu, C.-J., & Yang, D.-L. (2011). Unrelated parallel-machine scheduling with deteriorating maintenance activities. *Computers and Industrial Engineering*, 60, 602-605.

Chou, F. D. (2007). A joint GA+DP approach for single burn-in oven scheduling problems with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, 35, 587-595.

Cui, W. W., & Lu, Z. (2017). Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates. *Computers and Operations Research*, 80, 11-22.

Czyzak, P., & Jaszkiewicz, A. (1998). Pareto simulated annealing-A metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-criteria Decision Analysis*, 7, 34-47.

Deb, K., Pratap, A., Sgarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6, 182-197.

Detienne, B. (2014). A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints. *European Journal of Operational Research*, 235, 540-552.

Fowler, J. W., Mönch, L. & Ponsignon, T. (2015). Discrete-event simulation for semiconductor wafer fabrication facilities: A tutorial. *International Journal of Industrial Engineering*, 22, 661-682.

Graves, G. H., & Lee, C. Y. (1999). Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics*, 46, 845-863.

He, Y., Zhong, W., & Gu, H. (2006). Improved algorithms for two single machine scheduling problems. *Theoretical Computer Science*, 363, 257-265.

Jin, Y. L., Jiang, Z. H. & Hou, W. R. (2008). Multi-objective integrated optimization research on preventive maintenance planning and production scheduling for a single machine. *The International Journal of Advanced Manufacturing Technology*, 39, 954-964.

Kacem, I., & Chu C. (2008). Efficient branch-and-bound algorithm for minimizing the weighted sum of completion times on a single machine with one availability constraint. *International Journal of Production Economics*, 112 (1), 138-150.

Kacem, I, Chu C, & Souissi, A. (2008). Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers and Operations Research,* 35, 827-844.

Kanet, J. J. (1981). Minimizing variation of flow time in single machine systems. *Management Science,* 27, 1453-1459.

Kashan, A. H., Karimi, B. & Jolai, F. (2010). An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with non-identical job sizes. *Engineering Applications of Artificial Intelligence*, 23, 911-922.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science,* 22, 671-680.

Knowles, J. D., & Corne, D. W. (2000). Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation,* 8, 149-172.

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

Lee, C. Y. (1991). Parallel machines scheduling with non-simultaneous machine available time. *Discrete Applied Mathematics*, 30, 53-61.

Lee, C.Y. (1996). Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9 (3-4), 395-416.

Lee, C. Y., & Chen, Z. L. (2000). Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics,* 47, 145-165.

Lee, C. Y., & Liman S. D. (1992). Single machine flow-time scheduling with scheduled maintenance. *ACTA Informatica*, 29, 375-382.

Lei, D. (2013). Multi-objective artificial bee colony for interval job shop scheduling with flexible maintenance. *The International Journal of Advanced Manufacturing Technology,* 66, 1835-1843.

Low, C., Hsu, C. J. & Su, C. T. (2010). A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Systems with Application,* 37, 6429-6434.

Luo W., Cheng T.C.E. & Ji M. (2015) Single-machine scheduling with a variable maintenance activity. *Computers and Industrial Engineering*, 79, 168-174.

Ma, Y., Chu, C., & Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering,* 58, 199-211.

Moncel J., Thiery J., & Waserhole A. (2014). Computational performances of a simple interchange heuristic for a scheduling problem with an availability constraint. *Computers and Industrial Engineering*, 67, 216-222.

Mor B., & Mosheiov G. (2012). Heuristics for scheduling problems with an unavailability constraint and position-dependent processing times. *Computers and Industrial Engineering*, 62, 908-916.

Mosheiov, G., & Sidney, J. B. (2010). Scheduling a deteriorating maintenance activity on a single machine. *Journal of the Operational Research Society,* 61, 882-887.

Pinedo, M. (1995). Scheduling: theory, algorithms and systems, Prentice-Hall, Englewood Cliffs, NJ.

Pinedo, M. (2008). Scheduling: theory, algorithm, and systems. (3nd ed), New York: Springer, (Chapter 3).

Qi, X., Chen, T. & Tu, F. (1999). Scheduling the maintenance on a single machine. *Journal of the Operational Research Society,* 50, 1071-1078.

Sbihi, M. & Varnier C. (2008). Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness. *Computers and Industrial Engineering*, 55, 830-840.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121, 1-15.

Su, L. H., & Wang, H. M. (2017). Minimizing total absolute deviation of job completion times on a single machine with cleaning activities. *Computers and Industrial Engineering,* 103, 242-249.

Suppapitnarm, A., Seffen, K. A., Parks, G. T. & Clarkson, P. J. (2000). Simulated annealing: An alternative approach to true multi-objective optimization. *Engineering Optimization*, 33, 59-85.

Tamaki, H., Kita, H. & Kobayashi, S. (1996). Multi-objective optimization by genetic algorithms: A review. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 517-522.

Tan, K. C., Lee, T. H. & Khor, E. F. (2002). Evolutionary algorithms for multi-objective optimization: performance assessments and comparisons. *Artificial Intelligence Review*, 17, 253-290.

Tekinalp, O., & Karsli, G. (2007). A new multiobjective simulated annealing algorithm. *Journal of Global Optimization*, 39, 49-77.

Ulungu, L. E., Teghem, J. & Ost, C. (1998). Efficiency of interactive multi-objective simulated annealing through a case study. *Journal of Operational Research Society*, 49, 1044-1050.

Varadharajan, T. K., & Rajendran, C. (2005). A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research*, 167, 772-795.

Veldhuizen, D. A., & Van and Lamont, G. B. (1999). Multiobjective evolutionary algorithm test suites. *In Proc. ACM Symposium on Applied Computing*, 351-357.

Wang, C. S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers and Operations Research*, 29, 1621-1640.

Wang, S. & Liu M. (2013). A branch and bound algorithm for single-machine production scheduling integrated with preventive maintenance planning. *International Journal of Production Research*, 51, 847-868.

Wang, S., & Liu, M. (2015). Multi-objective optimization of parallel machine scheduling integrated with multi-resources preventive maintenance planning. *Journal of Manufacturing Systems*, 37, 182-192.

Xu, D., & Yin, Y. (2011). On single-machine scheduling with flexible maintenance activities. *International Journal of Advanced Manufacturing Technology*, 56, 1139-1145.

Xu, D., Yin, Y., & Li, H. (2009). A note on Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. *European Journal of Operational Research*, 197, 825-827.

Yang, D. L., Hung, C. L., Hsu, C. J., & Chern, M. S. (2002). Minimizing the makespan in a single machine scheduling problem with a flexible maintenance. *Journal of the Chinese Institute of Industrial Engineers*, 19, 63-66.

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. & Yang, D.-L. (2010). Minimizing the makespan on single-machine scheduling with aging effect and variable maintenance activities. *Omega*, 38, 528-533.

Yang, S.L., Ma, Y., Xu, D.L., Yang, J.B. (2011). Minimizing total completion time on a single machine with a flexible maintenance activity. *Computers and Operations Research*, 38, 755-770.

Ying, K.-C., Lu, C.-C., & Chen J.-C. (2016). Exact algorithms for single-machine scheduling problems with a variable maintenance. *Computers and Industrial Engineering*, 98, 427-433.

Table 1. An instance with nine jobs where *TV*=15 and *MT*=10

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
r_j	11	15	18	12	6	10	0	11	19
p_j	5	6	9	7	10	10	8	8	10
d_j	40	39	43	38	42	40	28	44	48
t_j	3	3	6	3	8	2	9	7	5
w_j	3	4	5	1	3	4	4	2	1

Table 2. The actions for case 1 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 1
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) < f^1(\pi^c)$ and $f^2(\pi^n) < f^2(\pi^c)$
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	
Action 1	$\pi^c = \pi^n$		
Action 2	Compare π^c with all solutions in the archive, and one of the following three results is satisfied		
	Result 1	Delete all solutions inferior to π^c in the archive and add π^c to the archive	
	Result 2	Add π^c to the archive if π^c is a new non-dominated solution	
	Result 3	The archive is not changeable if π^c is dominated	

Table 3. The actions for case 2 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 2
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) \leq f^1(\pi^c)$ and $f^2(\pi^n) > f^2(\pi^c)$ or
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	$f^1(\pi^n) > f^1(\pi^c)$ and $f^2(\pi^n) \leq f^2(\pi^c)$
Action 1	$Z^n = \alpha f^1(\pi^n) + \beta f^2(\pi^n)$, $Z^c = \alpha f^1(\pi^c) + \beta f^2(\pi^c)$		
Action 2	if ($Z^n < Z^c$) implement Action 2 in the Case 1		
	if ($Z^n \geq Z^c$) implement Action 1 in the Case 3		

Table 4. The actions for case 3 after comparing the current solution (π^c) with the neighbor solution (π^n)

	Objective 1	Objective 2	Case 3
π^c	$f^1(\pi^c)$	$f^2(\pi^c)$	$f^1(\pi^n) > f^1(\pi^c)$ and $f^2(\pi^n) > f^2(\pi^c)$
π^n	$f^1(\pi^n)$	$f^2(\pi^n)$	
Action 1	Use acceptance probability to decide whether $\pi^c = \pi^n$ or not		
Action 2	None		

Table 5. Average results obtained by the four algorithms compared with those obtained by the MIP model

Jobs	Ave. *RNI*					Ave. *HV*				
	EAT	EAD	SMOSA	NSGA-II	SSA	EAT	EAD	SMOSA	NSGA-II	SSA
5	0.911	0.978	0.978	0.978	0.978	0.974	0.999	0.999	0.999	0.999
7	0.927	0.990	0.990	0.990	0.990	0.989	1.000	1.000	1.000	1.000
9	0.914	0.998	0.998	0.997	0.998	0.997	1.000	1.000	1.000	1.000
11	0.910	0.998	0.935	0.998	0.998	0.996	1.000	0.999	1.000	1.000
13	0.947	1.000	0.746	0.995	1.000	1.000	1.000	0.995	1.000	1.000
Global Ave.	0.922	0.993	0.929	0.992	0.993	0.991	1.000	0.999	1.000	1.000

Table 5. Average results obtained by the four algorithms compared with those obtained by the MIP model (Cont.)

Jobs	Ave. *GD*				
	EAT	EAD	SMOSA	NSGA-II	SSA
5	1.794	0.360	0.360	0.360	0.360
7	1.062	1.440	1.440	1.440	1.440
9	16.120	0.082	0.082	0.082	0.082
11	7.239	0.564	14.920	0.564	0.564
13	8.252	0.000	55.306	1.118	0.000
Global Ave.	6.893	0.489	14.422	0.713	0.489

Table 6. Comparison of results of SMOSA and SSA for large-size problems

n	Ave. *RNI*			Ave. *HV*			Ave. *GD*		
	P_{SMOSA}	NSGA-II	P_{SSA}	P_{SMOSA}	NSGA-II	P_{SSA}	P_{SMOSA}	NSGA-II	P_{SSA}
20	0.457	0.8432	0.997	0.979	0.999	1.000	336.259	163.308	0.615
30	0.320	0.6039	0.951	0.971	0.998	1.000	2179.440	1163.776	57.912
40	0.501	0.645	0.928	0.982	0.998	1.000	3500.320	2410.338	175.517
50	0.690	0.7565	0.921	0.982	0.997	1.000	6110.820	5116.24	684.456
60	0.824	0.864	0.951	0.983	0.998	1.000	4819.559	4922.443	668.196
70	0.872	0.904	0.940	0.984	0.998	1.000	4888.314	6433.976	2522.108
80	0.907	0.924	0.972	0.985	0.998	1.000	4678.073	10350.480	1571.080
90	0.906	0.917	0.963	0.987	0.997	1.000	4151.309	11440.360	3691.800
100	0.928	0.938	0.971	0.990	0.997	1.000	6734.772	14853.960	3694.720
Global Ave.	0.711	0.822	0.955	0.983	0.998	1.000	4155.430	6317.209	1451.823

Figure 1

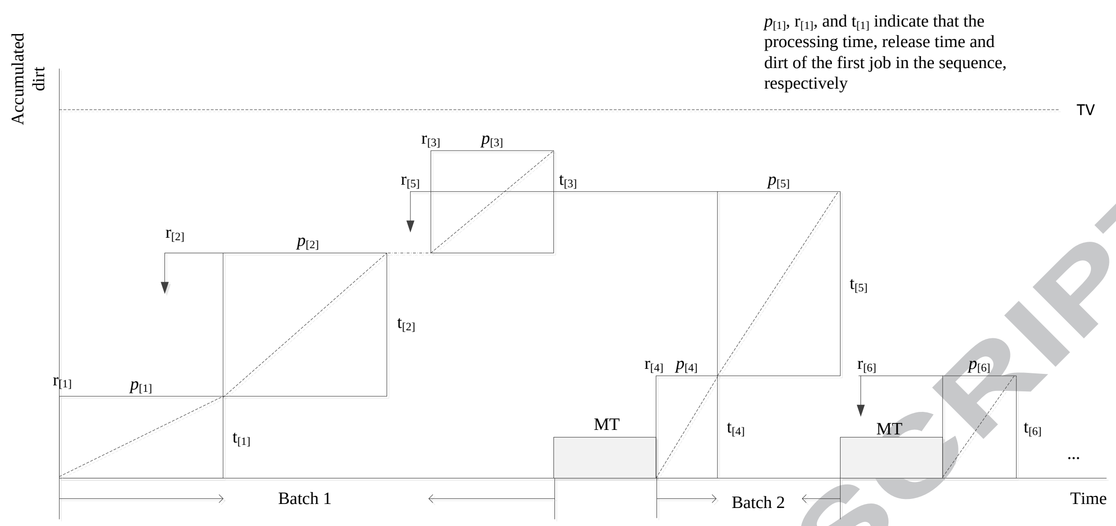
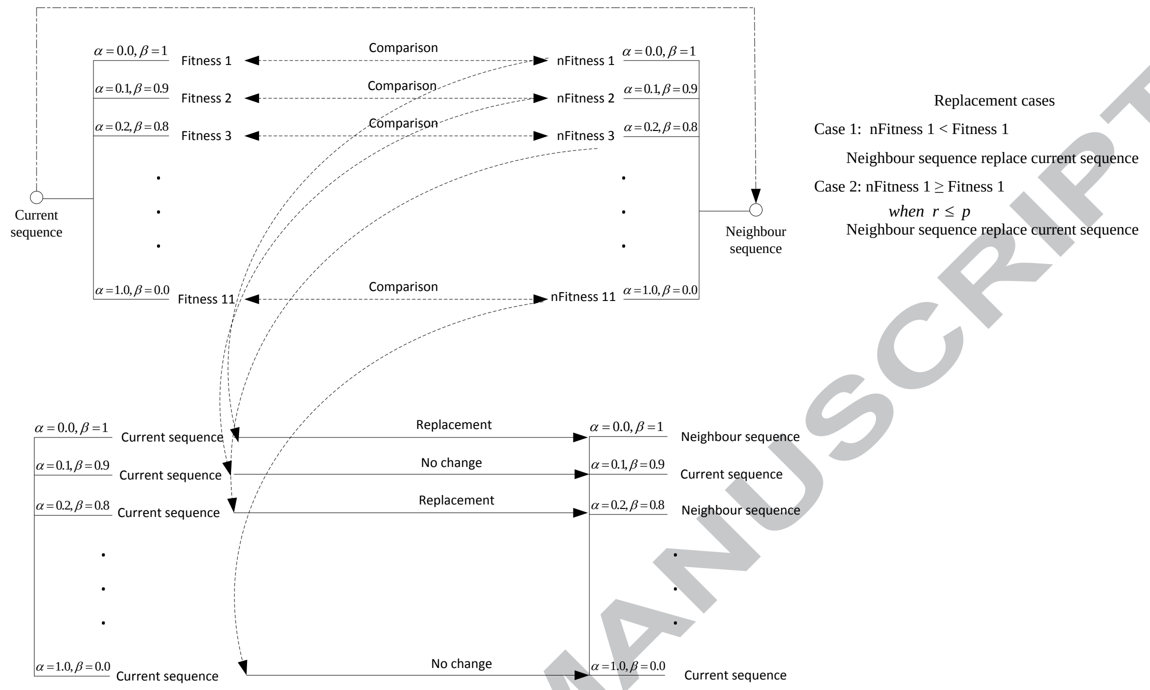


Figure 1. Schematic representation of the wet station production schedule

Figure 2



$\alpha = 0.0, \beta = 1$ Fitness 1
$\alpha = 0.1, \beta = 0.9$ Fitness 2
$\alpha = 0.2, \beta = 0.8$ Fitness 3
$\alpha = 1.0, \beta = 0.0$ Fitness 11
Current sequence
Comparison
nFitness 1 $\alpha = 0.0, \beta = 1$
nFitness 2 $\alpha = 0.1, \beta = 0.9$
nFitness 3 $\alpha = 0.2, \beta = 0.8$
nFitness 11 $\alpha = 1.0, \beta = 0.0$
Neighbour sequence
Replacement cases
Case 1: nFitness 1 < Fitness 1
Neighbour sequence replace current sequence
Case 2: nFitness 1 ≥ Fitness 1
when $r \leq p$
Neighbour sequence replace current sequence
$\alpha = 0.0, \beta = 1$ Current sequence
Replacement
$\alpha = 0.0, \beta = 1$ Neighbour sequence
$\alpha = 0.1, \beta = 0.9$ Current sequence
No change
$\alpha = 0.1, \beta = 0.9$ Current sequence
$\alpha = 0.2, \beta = 0.8$ Current sequence
Replacement
$\alpha = 0.2, \beta = 0.8$ Neighbour sequence
$\alpha = 1.0, \beta = 0.0$ Current sequence
No change
$\alpha = 1.0, \beta = 0.0$ Current sequence

Figure 2. SSA framework

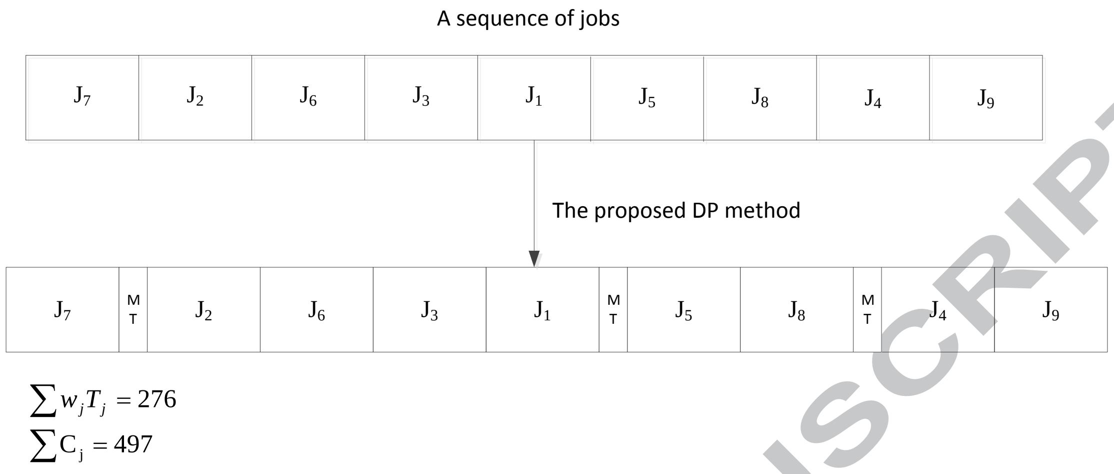


Figure 3. The schedule and objective value obtained by the DP method for nine-job instance of Table 1

Figure 4

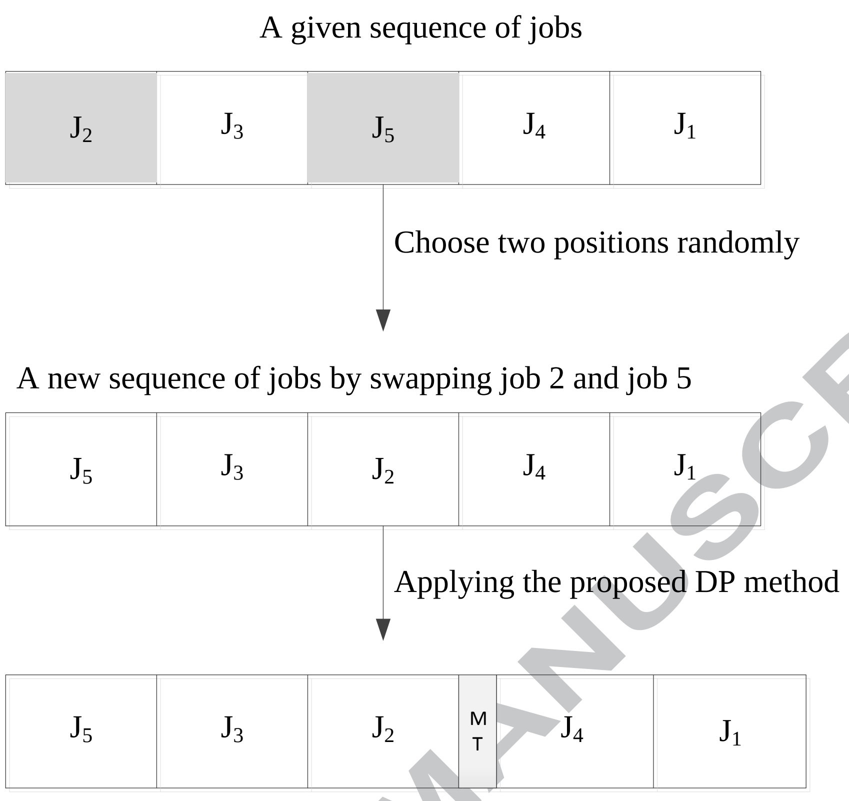
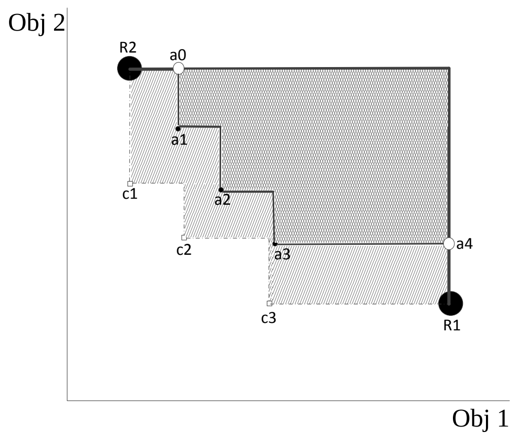

A given sequence of jobs
J_2 J_3 J_5 J_4 J_1
Choose two positions randomly
A new sequence of jobs by swapping job 2 and job 5
J_5 J_3 J_2 J_4 J_1
Applying the proposed DP method
J_5 J_3 J_2 MT J_4 J_1

Figure 4. Neighbor solutions generation

Figure 5

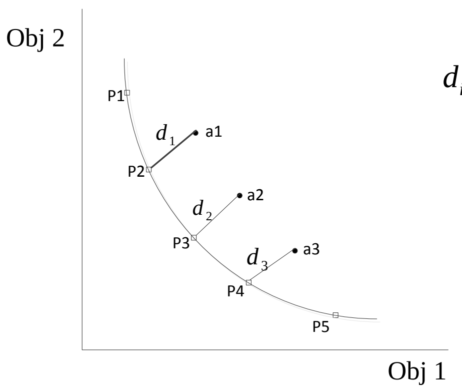


HV($P_{A(B)}$): the area surrounded by a0, a1, a2, a3 and a4

HV(P_c): the area surrounded by R2, c1, c2, c3 and R1

$$Index = \mathrm{HV}(P_{A(B)}) / HV(P_C)$$

Figure 5. Concept of *HV*



d_i : is the Euclidean distance between solutions obtained by algorithm *A* and the nearest member of *P**

$$index\ 3 = GD = \frac{1}{|X|}\left(\sum_{i=1}^{|X|} d_i^2\right)^{1/2}$$

where $|X|$ is the number of solutions obtained by algorithm *A*

Figure 6. Average distance of the algorithm A solutions from the set of Pareto-solutions

Figure 7

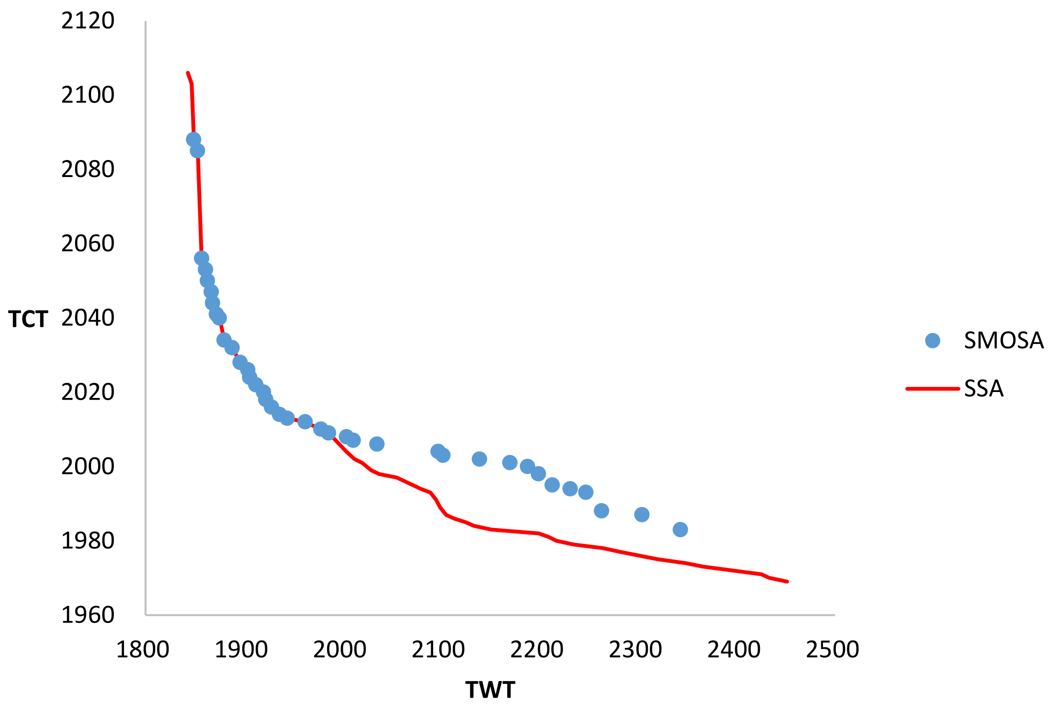


Figure 7. The sets of approximate non-dominated solutions obtained by SMOSA and SSA for the number of jobs equals to 20

Highlights

- A mixed integer programming model obtains Pareto-solutions for small problems.
- A dynamic programming (DP) method is proposed to schedule jobs and PMs.
- A scatter simulated annealing (SSA) with the DP method is proposed to find an approximate Pareto front.
- SSA was verified to be outstanding by comparing it with SMOSA algorithm.