Soft Computing (2021) 25:1501–1518
https://doi.org/10.1007/s00500-020-05234-7

METHODOLOGIES AND APPLICATION

Population-based Tabu search with evolutionary strategies for permutation flow shop scheduling problems under effects of position-dependent learning and linear deterioration

Oğuzhan Ahmet Arık[1,2]

Published online: 7 August 2020

Abstract

This paper investigates permutation flow shop scheduling (PFSS) problems under the effects of position-dependent learning and linear deterioration. In a PFSS problem, there are n jobs and m machines in series. Jobs are separated into operations on m different machines in series, and jobs have to follow the same machine order with the same sequence. The PFSS problem under the effects of learning and deterioration is introduced with a mixed-integer nonlinear programming model. The time requirement for solving large-scale problems type of PFSS problem is exceedingly high. Therefore, well-known metaheuristic methods for the PFSS problem without learning and deterioration effects such as iterated greedy algorithms and discrete differential evolution algorithm are adapted for the problem with learning and deterioration effects in order to find a faster and near-optimal or optimal solution for the problem. Furthermore, this paper proposes a hybrid solution algorithm that is called population-based Tabu search algorithm (TS_{POP}) with evolutionary strategies such as crossover and mutation. The search algorithm is built on the basic structure of Tabu search and it searches for the best candidate from a solution population instead of improving the current best candidate at each iteration. Furthermore, the performances of these methods in view of solution quality are discussed in this paper by using test problems for 20, 50, and 100 jobs with 5, 10, 20 machines. Experimental results show that the proposed TS_{POP} algorithm outperforms the other existing algorithms in view of solution quality.

Keywords Permutation flow shop scheduling · Learning effect · Deterioration effect · Iterated greedy · Discrete differential equation · Makespan · Tabu search · Evolutionary strategy

1 Introduction

In a PFSS problem, there are n jobs having m different operations on m serial machines. These jobs have to follow the same machine order ($1 \rightarrow 2 \rightarrow 3 \ldots. \rightarrow m$) with the same sequence. There are $n!$ possible job sequences in a PFSS problem. Figure 1 illustrates a solution to a PFSS problem instance consisting of 4 jobs and 4 machines. In this study, the PFSS problem is under the effects of learning and deterioration, and the performance criterion is to minimize makespan. The time requirement for solving large-scale PFSS problems is exceedingly high. Therefore, three well-known metaheuristic methods and a hybrid method that is called population-based Tabu search algorithm (TS_{POP}) with evolutionary strategies are proposed. Taillard's (1993) problem sets of 20, 50, and 100 jobs with 5, 10, and 20 machines are chosen to test performances of proposed methods.

The phenomenon of learning effect denotes a decrease in initially determined processing times because of the experience and expertise obtained via continuous repetition of similar tasks on machines or the system. On the contrary, the phenomenon of deterioration effect denotes an increase in initially determined processing times while jobs are waiting in the queue or are being processed on machines. Both of these effects have been widely studied for more than 15 years in scheduling problems. In most of

Communicated by V. Loia.

✉ Oğuzhan Ahmet Arık
oaarik@nny.edu.tr; o.a.arik@utwente.nl

[1] Industrial Engineering Department, Engineering Faculty, Nuh Naci Yazgan University, 38180 Kayseri, Turkey

[2] Industrial Engineering & Business Information Systems, Faculty of Behavioural, Management and Social Sciences, University of Twente, 7522 NB Enschede, The Netherlands

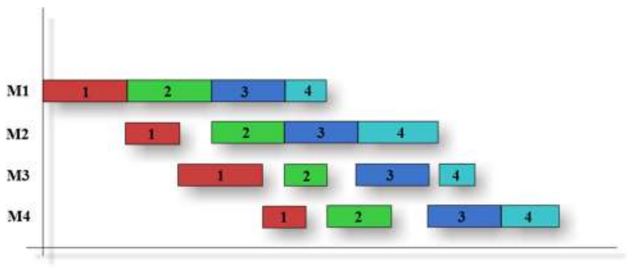


Fig. 1 Permutation flow shop scheduling problem consisting of 4 jobs and 4 machines

the scheduling problems, processing times are considered constant and researchers assume that the processing time of a job is not dependent on internal factors of the workplace such as learning or deterioration. Getting experience and the ability to learn from the current task can increase a worker's performance for similar tasks by applying new methods to new same or similar tasks. On the contrary, predetermined and assumed constant task duration can take longer because of deterioration. Gupta and Gupta (1988) presented a well-understood example of deterioration effect. In this example, the temperature of ingots that are to be processed in a rolling machine must be higher at a certain level and if the temperature of any ingot drops below to that certain level, then this ingot must be drawn back in order to be reheated up to that certain temperature level. This reheating process is an example of the deterioration effect.

Biskup (1999) introduced how position-dependent learning effect can be considered in scheduling problems. Let P_r be basic processing time of the job assigned at position r in the sequence and its actual processing time $P_{[r]}$ can be calculated as follows:

$$P_{[r]} = P_r r^a, \quad (1)$$

where a is learning effect coefficient for scheduling environment $(-1 < a < 0)$. Mosheiov (1991) showed the actual processing time $P_{[r]}$ of a job depends on its starting time and $P_{[r]}$ increases when starting time of that job increases under linear job deterioration effect. Let $S_{[r]}$ be starting time of the job at position r, $P_{[r]}$ can be calculated as follows:

$$P_{[r]} = P_r + BS_{[r]}, \quad (2)$$

where B is the linear deterioration effect coefficient for scheduling problems $(0 < B < 1)$. Both of these effects can be used in scheduling problems simultaneously as follows:

$$P_{[r]} = \left(P_r + BS_{[r]}\right) r^a. \quad (3)$$

For some of single machine scheduling problems under effects of learning and deterioration, the existing of polynomial algorithms such as the shortest processing time and the earliest due-date dispatching rules are proven by researchers (Wang and Wang 2011; Wang 2007; Wang et al. 2008b; Cheng et al. 2008; Gordon et al. 2008; Yang and Kuo 2010). Even for some flow shop scheduling problems with some special cases, the existing of polynomial algorithms are proven by researchers (Wang et al. 2008a, b; Wang 2006). These special cases in flow shop scheduling problem are increasing series of dominating machines, 2-machine environment, equal job processing times and a fixed job in the first position of the first machine. Without these special cases, the complexity of the PFSS problem under the effects of learning and deterioration is still NP-Hard.

In this paper, we integrate two strong metaheuristics for combinatorial optimization problems and apply our proposed solution approach to the PFSS problem where jobs are under the effects of learning and deterioration. The proposed algorithm uses the basic structure of Tabu search, and it searches for the best candidate from a solution population instead of improving the current best candidate at each iteration. It also uses some evolutionary strategies such as crossover and mutation operators to escape and renew the solution population. Most of the hybrid algorithms including TS and evolutionary strategies use the genetic algorithm (GA) as the main framework and use TS as a solution improvement tool. On the contrary to papers in the literature, we use evolutionary strategies to escape from local optima. Furthermore, we compare our proposed algorithm with some existing algorithms for PFSS problems.

2 Literature review

The PFSS problems with makespan minimization have been interested among researchers for more than 40 years. There are some review papers in the literature. Some of these review papers are Fernandez-Viagas et al. (2017), Yenisey and Yagmahan (2014), Reza Hejazi and Saghafian (2005) and Framinan et al. (2002, 2004). Due to the complexity of the problem, the PFSS problem is one of the most studied problems in the operations research literature. The PFSS problem under learning and deterioration effects is expressed as $F_m|prmu, LE, DE|C_{\max}$ with the notation of Graham et al. (1979). As far as we known, the best effective algorithms for PFSS without learning and deterioration effects have been variants of iterated greedy (IG) algorithm.

Ruiz and Stützle (2007) proposed an iterated greedy algorithm (IG_RS) that applies two phases iteratively. In their algorithm, the first phase named destruction eliminates some jobs from the incumbent solution, and the second phase named construction reinserts the eliminated jobs into the sequence by using the NEH construction heuristic. They also proposed using a local search technique in their IG_RS. Experimental results in their study show that their proposed IG with local search (IG_RS$_{LS}$) outperforms the state-of-art algorithms published for the PFSS problem until then. They also presented some new optimum and best solutions for Taillard benchmark instances. Ruiz and Stützle (2008) proposed two iterated greedy algorithms for PFSS problems with sequence-dependent setup times for minimizing the makespan and total weighted tardiness. Another variant of the IG algorithm named IG$_{RIS}$ for the problem was proposed by Pan et al. (2008). This variant of IG uses a new local search named reference insertion schema (RIS) instead of LS proposed by Ruiz and Stützle (2007) and Taillard (1990). The RIS uses the reference permutation obtained from the NEH algorithm, and it removes/reinserts jobs from that referenced list one by one to find better solutions. The RIS and LS use Taillard's speed-up schema to calculate the makespan or flowtime of the solution. The proposed IG$_{RIS}$ of Pan et al. (2008) outperformed so far existing metaheuristics in the literature. Pan et al. (2008) also proposed a discrete differential evolution (DDE$_{RLS}$) algorithm with RIS for PFSS problems with the makespan criterion. While finding a position for a removed job in the local search phase, there can be lots of partial solutions (ties) having the same objective function value. These ties may lead the algorithm in a cycle. Therefore, these ties must be broken with a tiebreaking mechanism to increase solution quality. Kalczynski and Kamburowski (2008), Dong et al. (2008), Fernandez-Viagas and Framinan (2014), and Vasiljevic and Danilovic (2015) proposed new tiebreaking mechanisms for the problem. Fernandez-Viagas and Framinan (2014) presented a tiebreaking mechanism (TB$_{FF}$) for NEH, IG$_{RIS}$, and IG_RS$_{LS}$ algorithms. Their experimental study revealed that these algorithms with TB$_{FF}$ outperform their original versions. Rossi et al. (2016) developed a new heuristic named as Gx by combining their new heuristic with different tiebreaking and initial orders procedures found in the literature. Dubois-Lacoste et al. (2017) suggested optimizing the partial solution after the destruction phase of the classical IG algorithm. Their new variant of the IG algorithm outperformed the existing IG algorithms. Fernandez-Viagas et al. (2017) used their proposed TB$_{FF}$ within lots of different heuristics and different metaheuristics. They compared those algorithms with each other for the same performance criterion. Their experimental study revealed that IG$_{RIS}$ and IG_RS$_{LS}$ with TB$_{FF}$ outperform other existing and promising metaheuristics. They also proved that their proposed TB$_{FF}$ and Taillard's speed-up schema increase the solution quality of the algorithms. Fernandez-Viagas and Framinan (2019) proposed a best-of-breed (IG$_{BOB}$) combination of recent variants of IG algorithms and their components. In their proposed IG variant, they inspired by algorithms of Benavides and Ritt (2018), Dubois-Lacoste et al. (2017), and Fernandez-Viagas and Framinan (2014). Their experimental study revealed that their proposed IG$_{BOB}$ is the best-so-far algorithm for the problem. Their IG$_{BOB}$ combines initial solution of Benavides and Ritt (2018), the local search procedure of Benavides and Ritt (2018) and local search for partial solution proposed by Dubois-Lacoste et al. (2017) with their existing TB$_{FF}$.

Janiak and Portmann (1998) presented a genetic algorithm for PFSS problems with resource allocation for constrained resources such as energy, catalyzer, and raw materials in order to find a schedule that minimizing the makespan. Rajkumar and Shahabudeen (2009) proposed an improved GA including multi-crossover, multi-mutation, and hypermutation operators in order to solve PFSS problems with the makespan performance criterion. Nagano et al. (2008) proposed a constructive GA of which parameters are calibrated design of experiment and their proposed GA uses Nawaz–Enscore–Ham (NEH) and local search heuristic to define fitness values of solutions. Pasupathy et al. (2006) studied multi-objective PFSS problems with their proposed GA in order to find a pareto-optimal solution for makespan and total flowtime performance criteria. Their proposed algorithm makes use of the principle of non-dominated sorting, coupled with the use of a metric for crowding distance being used as a secondary criterion. This approach is intended to alleviate the problem of genetic drift in GA methodology. Chen et al. (2012) presented self-guided GA with a novel strategy that combines global statistical information collected previous solutions and location information about individual solutions. One of the most prominent papers using simulated annealing (SA) in PFSS problems in order to minimize the makespan belongs to Osman and Potts (1989). Xiao et al. (2012) studied SA in PFSS problems with order acceptance and weighted tardiness when the objective is to maximize the total net profit with weighted tardiness penalties. Suresh and Mohanasundaram (2004) proposed an SA with a perturbation mechanism called segment random insertion that is used to generate the neighborhood of a given sequence in PFSS problems with makespan and total flowtime performance criteria. Hybrid algorithms which are designed by using the best parts of well-known metaheuristics or heuristics have been also studied in PFSS problems. Sun et al. (2015) proposed a GA based on SA in order to escape local optima and increase searching efficiency. Lin et al. (2015) used a hybrid algorithm depending on an evolutionary algorithm named as backtracking search algorithm (BSA) in order to solve PFSS problems with makespan minimization. Their hybrid BSA includes crossover/mutation strategies and SA mechanism. Laha and Chakraborty (2009) investigated PFSS problems with the makespan criterion and presented a new hybrid heuristic algorithm that is designed by combining elements from SA, NEH, and their previously published composed heuristic. Li et al. (2008) considered a multi-objective PFSS problem by proposing a hybrid algorithm based on particle swarm optimization (PSO), NEH, and SA algorithms. In their proposed hybrid algorithm, different well-known heuristics are used to create better evolutionary search results and to evaluate these search results' fitness. Haq et al. (2010) compared two heuristics that are dependent on the artificial neural network (ANN) and GA for a PFSS problem where the objective is to minimize makespan. One of their algorithms is ANN–GA starting with random population. The second algorithm is also ANN–GA, but this algorithm uses the random insertion perturbation scheme (RIPS) and they named this algorithm as ANN–GA–RIPS. They showed that ANN–GA–RIPS outperforms ANN–GA. Zobolas et al. (2009) proposed a hybrid metaheuristic for a PFSS problem with makespan minimization. Their proposed algorithm consisted of three heuristics. These are a greedy randomized constructive heuristic for initial population generation, a GA for solution evaluation and a variable neighborhood search (VNS) to improve the population. Tseng and Lin (2010) considered a PFSS problem where the objective is to minimize total flow time of the schedule, and they proposed a hybrid metaheuristic including GA for global search and a Tabu search (TS) for local search.

The learning effect has been a hot topic among scheduling researchers for more than 15 years. However, there has been a smaller number of papers focusing on PFSS problems with learning effect consideration. He (2016) considered a PFSS problem with a general exponential learning effect when the objective is to minimize maximum lateness by proposing several heuristic methods. Lee and Chung (2013) proposed a branch-and-bound algorithm and two heuristic methods to find an optimum or near-optimum solution when the objective is to minimize total tardiness of a PFSS problem under learning effect. Chung and Tong (2012) considered a machine-based learning effect in the PFSS problem when the objective is to minimize the weighted sum of total completion time and makespan. For an optimum solution, they proposed a branch-and-bound algorithm and for a near-optimum solution, they proposed two heuristic methods. In another study of Chung and Tong (2011), they considered learning effect in the PFSS problem with makespan minimization by proposing a dominance theorem and a lower bound to accelerate the branch-and-bound algorithm seeking an optimal solution. Another study using a branch-and-bound algorithm to solve the PFSS problem with learning consideration was conducted by Wang and Zhang (2015). Qin et al. (2016) studied position-dependent learning effect in the PFSS problem for different performance criterions such as makespan, total completion time, total weighted completion time, and maximum lateness by proposing GA and quantum differential evolutionary algorithm. Toksarı and Arık (2017) addressed some performance criteria such as makespan, the sum of completion times, and the sum of weighted completion times on single machine under fuzzy learning effect with fuzzy processing times. They proposed a credibility-based chance-constrained programming approach for their proposed MINLP and they proved that these problems can be solvable in polynomial time. Shiau et al. (2015) proposed a branch-and-bound algorithm and several GA algorithms in order to obtain feasible solutions for a two-agent scheduling problem in a two-machine permutation flow shop with learning effects. Xu et al. (2016) investigated re-entrant permutation flow shop scheduling with a position-based learning effect to minimize the total completion time. They developed some heuristics and a GA to search for approximate solutions. Mustu and Eren (2018) proposed GA, the kangaroo, and the variable neighborhood search algorithms for PFSS under position-dependent learning effect. Shi and Wang (2019) investigated two-machine no-wait PFSS with common due window assignment, learning effect, and resource allocation. Geng et al. (2019) addressed the no-wait flow shop scheduling problem with simultaneous consideration of common due-date assignment, convex resource allocation, and learning effect in a two-machine setting. Wang et al. (2019a, b) investigated PFSS problems with a truncated exponential sum of logarithm processing time-based and position-based learning effects. Wang et al. (2019a) investigated position-weighted learning effect and job release dates on single machine environment, and they proposed a branch-and-bound algorithm and heuristics for the problem.

The deterioration effect has been also studied by researchers in scheduling literature. Yin and Kang (2015) studied the makespan performance criterion in the PFSS problem with proportional deterioration. Furthermore, they showed the problem can be polynomially solvable for some special cases of the problem. Lee et al. (2014) investigated total tardiness minimization in PFSS problem with deterioration consideration. They proposed a branch-and-bound algorithm and two metaheuristic methods that are particle swarm optimization and SA. Wang and Wang (2013) considered three-machine PFSS problem with deteriorating jobs in order to minimize makespan, and they solved their problem by using a branch-and-bound algorithm of which efficiency is increased with two heuristic methods. Bank et al. (2012) investigated a PFSS problem with deteriorating jobs and they solved their problem with two different methods. These are particle swarm optimization with local search and SA. They showed that particle swarm optimization with local search outperforms SA in terms of solution quality but SA takes less time to find a solution. Lee et al. (2009) addressed total completion time minimization in the PFSS problem, and they tested several well-known heuristics for their problem with several deterioration patterns by proposing a dominance rule and efficient lover bound to increase search efficiency. Sun et al. (2019) investigated PFSS problems with simple linear deterioration where the objectives are to minimize the logarithm of the makespan, total logarithm of the completion time, the total weighted logarithm of the completion time, and the sum of the quadratic job logarithms of the completion times. They proposed branch-and-bound algorithms for the problems. Wang and Liang (2019) considered a single machine group scheduling problem with deteriorating jobs and resource allocation.

There are some papers investigating learning and deterioration effects simultaneously. As far as we know, the first paper that investigated these effects simultaneously was proposed by Wang (2006). Wang (2007, 2009) investigated some performance criteria for single machine scheduling problems under both effects, and they showed the existence of polynomial algorithms for these problems with/without some special cases. Toksarı and Güner (2008) proposed a MINLP for parallel machine scheduling problem under the effects of deterioration and learning where the objective is to minimize earliness/tardiness costs. Toksarı and Güner (2010) investigated a parallel machine scheduling problem under learning and deterioration effects with sequence-dependent setup times and a common due date. They proved that the optimal solution is V-shaped. Arık and Toksarı (2018) investigated a multi-objective fuzzy parallel machine scheduling problem where the objectives are to minimize earliness cost, to minimize tardiness cost and to minimize the cost of setting due dates. In their study, all parameters such as processing times, coefficients of learning and deterioration, and decision variables except binary decision variables are in form of fuzzy numbers. They proposed a local search algorithm to solve their problem, and they compared their method with fuzzy mathematical programming methods in the literature. Arık and Toksarı (2019) proposed a MINLP model for a fuzzy parallel machine scheduling problem under fuzzy job deterioration and learning effects with fuzzy processing times in order to minimize fuzzy makespan by using possibilistic distributions of fuzzy parameters and possibilistic linear programming approaches. Lu (2016) considered no-idle permutation flow shop scheduling problems with time-dependent learning effect and deteriorating jobs where the objectives are to minimize the makespan and the total completion time.

For combinatorial optimization problems, the hybridization of two or more metaheuristics is a common approach to use specific advantages of those algorithms. For instance, while GA presents a population-based stochastic search to except from local optima and TS uses a deterministic search with restricting the feasible neighborhood by neighbors that are excluded. There are some valuable hybrid approaches including GA and TA at the same time. Glover et al. (1995) used TS as a strategic oscillation in GA to allow effective transitions between feasible and infeasible regions. Abdinnour-Helm (1998) integrated TS into GA for uncapacitated hub location problem. Liaw (2000) integrated TS into GA for the open shop scheduling problem where the objective is to minimize the makespan. Li et al. (2003) used TS in a classical GA for assembly process planning problem. Jat and Yang (2011) proposed a two-phase hybrid algorithm for post-enrollment course timetabling. In their proposed method, GA is used in the first phase to improve the solution population, and TS is used in the second phase to improve the solution quality of the best solution found by GA. Meeran and Morshed (2012) proposed a hybrid algorithm including GA and TS for job shop scheduling problems. Zhang et al. (2013) proposed a hybrid algorithm including GA and TS for a multi-objective dynamic job shop scheduling problem with random job arrivals and machine breakdowns. Palacios et al. (2015) proposed a genetic Tabu search algorithm for fuzzy flexible job shop scheduling problem where the objective is to minimize the makespan. In their algorithm, the TS algorithm is applied to all solutions in the population after GS operations. Li and Gao (2016) proposed a hybrid solution approach including GA and TS for flexible job shop scheduling problem. In their algorithm, the TS algorithm is applied to all solutions in the population after GS operations.

The PFSS problems need a single job sequence from $n!$ possible alternative sequences for all machines. Exact solution algorithms may not always solve these problems in polynomial time because number of input does not increase polynomially. In this study, IG_RS$_{\mathrm{LS}}$, IG$_{\mathrm{RIS}}$, DDE$_{\mathrm{RLS}}$, and TS$_{\mathrm{POP}}$ methods are proposed in order to find approximate and faster solutions. Each of the investigated algorithms has advantages for solving combinatorial optimization problems. Each of the proposed solution techniques is executed for Taillard's (1993) test problems consisting of 20, 50, and 100 jobs with 5, 10, and 20 machines. For most of Taillard's (1993) test problems without learning and/or deterioration effects, the best makespans or upper bounds of makespans are known. Since there are no published upper bounds for the PFSS problem under the effects of learning and deterioration, we solved some of the test problems of Taillard's (1993) with a commercial solver. The results of the proposed algorithms are compared with upper bounds found by us in the section of numerical examples.

3 Mathematical model

In this section, a MINLP model is introduced for permutation flow shop scheduling problems under the effects of learning and deterioration when the objective function is to minimize the makespan.

Indices

i: job index, $i = 1\ldots n$
j: machine index, $j = 1\ldots m$
r: common position index in all machines $r = 1\ldots n$

Parameters

$P_{i,j}$: basic processing time of job i on machine j
a: learning effect coefficent
B: deterioration effect coefficent

Decision variables

$X_{i,r}$: if job i is assigned on position r of all machines, then it's 1, otherwise 0
$P_{[r],j}$: actual processing time of the job assigned on position r in machine j
$C_{[r],j}$: completion time of the job assigned on position r in machine j
$S_{[r],j}$: starting time of the job assigned on position r in machine j
$C_{\max}$: makespan of the schedule

Model

$$\text{Min } z = C_{\max} \tag{4}$$

s.t.:

$$C_{\max} \geq C_{[n],m} \quad \forall r, j \tag{5}$$

$$\sum_{i}^{n} X_{i,r} = 1 \ \forall r \tag{6}$$

$$\sum_{r}^{n} X_{i,r} = 1 \ \forall i \tag{7}$$

$$C_{[r],j} \geq S_{[r],j} + P_{[r],j} \ \forall r, j \tag{8}$$

$$S_{[r],j} \geq C_{[r],j-1} \ \forall r, \ j = 2, \ldots, m \tag{9}$$

$$S_{[r],j} \geq C_{[r-1],j} \ \forall j, \ r = 2, \ldots, m \tag{10}$$

$$P_{[r],j} = \left(\left(\sum_{i}^{n} X_{i,r} * P_{i,j} \right) + B * S_{[r],j} \right) * r^{a} \tag{11}$$

$$C_{[0],1} = 0 \tag{12}$$

$$C_{\max} \geq 0 \tag{13}$$

$$C_{[r],j}, P_{[r],j}, S_{[r],j} \geq 0 \tag{14}$$

$$X_{i,r} \in \{0, 1\}. \tag{15}$$

The objective function (4) is to minimize the makespan value of the schedule. Constraint (5) assures that the makespan is the maximum completion time of all jobs. Constraint (6) assures that position number r for all machines is used for only one job. Constraint (7) assures that a job is assigned on only a position number r of all machines. Constraint (8) shows that the completion time of the job assigned on a common position r in all machines is equal to or greater than the sum of its starting time and actual processing time. Constraint (9) shows that the starting time of the job on position r in j machine is greater than or equal to the completion time of the job in the same position of the previous machine. Constraint (10) shows that the starting time of the job on position r in j machine is greater than or equal to the completion time of the job of the previous position in the same machine. Constraint (11) shows the calculation of the actual processing time of the job assigned on position r in machine j. It is required to determine which job is assigned to which position of which machine. Therefore, transitions among job positions and jobs are required. Since transition among the processing time of a job in position r ($P_{[r],j}$) and jobs' processing times ($P_{i,j}$) makes the problem nonlinear, the proposed mathematical model is a mixed-integer nonlinear mathematical model. These transitions are made by Constraint (11). Constraint (12) shows that all jobs are ready to be processed at the beginning. Constraints (13–14) show that starting times, actual processing times, and completion times are greater than or equal to zero. Constraint (15) shows that the decision variable $X_{i,r}$ is binary.

4 Population-based Tabu search with evolutionary strategies

The Tabu search algorithm was introduced by Glover (1989, 1990) to present a search strategy for solving combinatorial optimization problems whose applications range from graph theory and matroid settings to general pure and mixed-integer programming problems. Tabu search is a deterministic search algorithm to prevent cyclical solutions by transforming only one solution into another. In order to avoid cycling, the TS stays away from certain moves that create undesired neighborhoods. These moves or undesired solutions are listed in a short-term memory named as Tabu list. Although Tabu search was originally designed for a single current solution to create better solutions by avoiding cycling, this paper proposes a Tabu search with a population-based search and evolutionary strategies. There are so many possible and feasible solutions in the solution space, and most of them can be reached by simple moves among solutions. This proposed search method uses a solution population and searches for best candidates by locally searching the population's individuals. Then, the proposed algorithm holds and forbids the current solution with the help of a Tabu list to create better solutions. If the solution is trapped in a local area and the solution population starts to be ineffective for improving the solution, then some evolutionary strategies such as crossover and mutation take place to create a new solution population that helps to improve the current solution. The hybrid algorithms (Zhang et al. 2013; Palacios et al. 2015; Li and Gao 2016) including GA and TS in the literature use generally the main framework of GA such as evaluation, selection, crossover, and mutation operators; then, they use TS algorithm to improve the best solution obtained from GA operators. In this paper, we use the main framework of the TS algorithm to improve the solution quality of individuals in the population and use evolutionary strategies such as crossover and mutation to escape the local optima. Algorithm 1 shows the general schema for the proposed population-based Tabu search with evolutionary strategies (TS_{POP}).

Algorithm 1: TS_{POP}

```
Start
Population ≔ Initial_Population;
π ≔ Find_Best_Solution(Population);
$\pi^* ≔ \pi$;
$C_{Max}(\pi)$ ≔ Calculate_Makespan(π);
$C_{Max}(\pi^*) ≔ C_{Max}(\pi)$;
while (elapsed time in ms $\leq m.n.t/2$) do
    $C_{Max}^{old} ≔ C_{Max}(\pi)$;
    Population ≔ Local_Search_Population(Population);
    π ≔ Find_Best_Solution(Population);
    $C_{Max}(\pi)$ ≔ Calculate_Makespan(π);
    if $C_{Max}^{old} \geq C_{Max}(\pi)$ then
        $k ≔ k + 1$;
    else
        $k ≔ 0$;
        $C_{Max}(\pi^*) ≔ C_{Max}(\pi)$;
        $\pi^* ≔ \pi$;
        Call Check_Tabu_List(π);
    endif
    if $k > K$ then
        $k ≔ 0$;
        Population ≔ Generate_Population(Population, $\pi^*$, π);
        Population ≔ Crossover(Population);
        Population ≔ Mutation(Population);
    endif
endwhile
return $\pi^*$;
Stop.
```

The Initial_Population procedure in Algorithm 1 is designed to produce a solution population that may be expandable to a global optimal schedule. Algorithm 2 produces n solutions. Then, the number of solutions in the population is decreased or increased to 60 solutions. The first position of the job orders in these solutions start with each possible solution. That means the first job of the first solution in the population has job#1, and the first job of the second solution in the population has job#2. Thus, each job is assigned to first positions of solutions. Then, find and assign the best job to the second position of the job orders that minimize the total idle times of machines for second position. For instance, if there are 5 jobs $j = \{1, 2, 3, 4, 5\}$ and 3 machines $k = \{1, 2, 3\}$, so we can produce 5 solutions for the population. These solutions are $\pi_1 = \{1, ?, ?, ?, ?\}$, $\pi_2 = \{2, ?, ?, ?, ?\}$, $\pi_3 = \{3, ?, ?, ?, ?\}$, $\pi_4 = \{4, ?, ?, ?, ?\}$ and $\pi_5 = \{5, ?, ?, ?, ?\}$. The first positions of solutions are fulfilled, and now the second positions of job orders of solutions are selected from unassigned jobs to minimize the total idle time of machines. For solution#1 $\pi_1 = \{1, ?, ?, ?, ?\}$, the unassigned jobs are {2, 3, 4, 5}, and we can select a job that minimizes the total idle time of machines. In that situation, if job#3 assures the minimum total idle time, then $\pi_1 = \{1, 3, ?, ?, ?\}$. This goes on until there are no unassigned job remains for each solution in the population. This procedure depends on the profile fitting procedure proposed by McCormick et al. (1989). The profile fitting heuristic was originally proposed for minimizing the cycle time of serial workstations in an assembly line with the blocking constraint. We used that heuristic for creating an initial solution population. Each job is assigned to the first position of each solution, then a search is made for determining the job for the second position by considering the total idle times of machines. This goes on until there is no unassigned job remaining. After creating initial population, the solutions ordered in an increasing order of their makespan values. Then, the number of solution in the initial population is increased or decreased to 60 by selecting best 60 solutions from the initial solution. If the population size is 20, then these 20 solutions are directly placed in 60 solutions. For remaining 40 solutions, randomly generated new solutions are placed in the population. On the contrary, if the population size is 100, then the first best 60 solutions are directly placed in the population.

Algorithm 2: Initial_Population (n, m, p_{ij})

```
Start
Step 0: Read all parameters $(n, m, p_{ij})$;
Step 1: Create an empty population with $n$ solution $Population := \{\pi_1, \ldots, \pi_n\}$;
Step 2: Assign each job $j = (1, \ldots, n)$ to first positions of solutions $\{\pi_1, \ldots, \pi_n\}$;
        and set $r := 1$;
Step 3: Set $r := r + 1$ and Find the jobs for solutions $\{\pi_1, \ldots, \pi_n\}$ that minimize total idle time of all
        machines and assign them to $r^{th}$ positions of solutions $\{\pi_1, \ldots, \pi_n\}$;
Step 4: If there are any unassigned jobs for all solutions, Go To Step 3;
Step 5: Order solutions in the population in an increasing order of the makespan values;
Step 6: If the number of solutions in $Population := \{\pi_1, \ldots, \pi_n\}$ is less than 60;
        Then increase number of solutions by generating new random solutions;
        Else reduce the number of solutions by selecting the best 60 solutions;
Step 7: Return Population $:= \{\pi_1, \ldots, \pi_{60}\}$
Stop.
```

After creating the first population, the same Local_Search_Population procedure in Algorithm 1 is designed to improve solution quality for the first *B* solutions in the population. Then, these solutions are individually sent to Local_Search operator of the proposed algorithm. The basic idea of the proposed Local_Search_Population is to produce better neighborhoods that have a chance to be the best current candidate. The Local_Search_Population and Local_Search procedures are given in Algorithms 3 and 4.

Algorithm 3: Local_Search_Population (Population)

```
Start
For i := 1 to B do
    $\pi' := \pi_i$
    if $\pi'$ is not in Tabu_list then
        $Cmax(\pi') := Calculate\_Makespan(\pi')$
        $\pi' := Local\_Search(\pi', Cmax(\pi'))$
    endif
    $\pi_i := \pi'$
endfor
return Population $:= \{\pi_1, \ldots, \pi_{60}\}$
Stop.
```

After creating the first population, the same Local_Search procedure in Algorithm 3 is designed to improve the incumbent solution. The basic idea of the proposed local search is to produce new neighborhoods that have a chance to be the best current candidate. To produce new neighborhoods of a solution, three different search operations are used *C* (predetermined number of local search iterations) times by selecting a random job from the current solution. Insertions, swapping, and double-swapping operations are applied, respectively, to the current solution. If the candidate solution is not in the Tabu list and if the makespan value is less than or equal to the incumbent solution's makespan, then the incumbent solution is replaced with this new candidate solution. Insertion local search is one of the most used search operators for PFSS problems. In this study, the proposed Local_Search procedure uses the insertion search by selecting a random job from the incumbent, and it tries to find a better solution by inserting that job to possible all positions. The swapping operation in this study uses a randomly selected job from the incumbent solution, and it swaps that job's position with all possible jobs in the solution. The double-swapping operator selects a random position r. Then, the operator removes the jobs in positions r and $r + 1$ from the solution and tries to find a better solution by inserting them to all possible positions again. After inserting these two jobs in the solution, these jobs are also swapped to find a better solution. The length of the Tabu list is 100. If a new best makespan is found, then this makespan and its schedule are added to the Tabu list. In Local_Search procedure, the solutions are replaced with their neighborhoods, so to avoid turn back to previous solutions, the new better solution value is added in the Tabu list by using Check_Tabu_List operator that is given in Algorithm 5. If the number of solutions in the Tabu list exceeds 100, then the oldest solution in the Tabu list is removed. The Local_Search algorithm is given in Algorithm 4.

Algorithm 4: Local_Search (π, C_{max})

```
Start
For i := 1 to C do
    r: = Generate_Random_Position(1, x);
    For j := 1 to n do
        if r ≠ j then
            π' := Remove the job from r^th position and insert it in j^th position;
            if π' is not in Tabu_List then
                $C_{Max}(\pi')$ := Calculate_Makespan(π');
                if $C_{max} \le C_{Max}(\pi')$ then
                    π := π';
                    $C_{max} := C_{Max}(\pi')$;
                    Call Check_Tabu_List(π);
                endif
            endif
        endif
    endfor
endfor
For i := 1 to C do
    r: = Generate_Random_Position(1, x);
    For j := 1 to n do
        if r ≠ j then
            π' := Swap the jobs from r^th and j^th positions
            if π' is not in Tabu_List then
                $C_{Max}(\pi')$ := Calculate_Makespan(π');
                if $C_{max} \le C_{Max}(\pi')$ then
                    π := π';
                    $C_{max} := C_{Max}(\pi')$;
                    Call Check_Tabu_List(π);
                endif
            endif
        endif
    endfor
endfor
For i := 1 to C do
    r: = Generate_Random_Position(1, x − 1);
    For j := 1 to n do
        if r ≠ j and r + 1 ≠ j then
            int d1 = the job in r^th position
            int d2 = the job in r + 1^th position
            int d3 = the job in j^th position
            if j > r then
                π' := remove d1, d2 and d3 jobs from the solution and insert d3 to r^th position and then insert d1 and d2 to j^th and
                j − 1^th positions;
                π'' := π';
                π'' := Swap the jobs from j^th and j − 1^th positions;
            else
                π' := remove d1, d2 and d3 jobs from the solution and insert d3 to r^th position and then insert d1 and d2 to j^th and
                j + 1^th positions;
                π'' := π';
                π'' := Swap the jobs from j^th and j + 1^th positions;
            endif
            if π' is not in Tabu_List then
                $C_{Max}(\pi')$ := Calculate_Makespan(π');
                if $C_{max} \le C_{Max}(\pi')$ then
                    π := π';
                    $C_{max} := C_{Max}(\pi')$;
                    Call Check_Tabu_List(π);
                endif
            endif
            if π'' is not in Tabu_List then
                $C_{Max}(\pi'')$ := Calculate_Makespan(π');
                if $C_{max} \le C_{Max}(\pi'')$ then
                    π := π'';
                    $C_{max} := C_{Max}(\pi'')$;
                    Call Check_Tabu_List(π);
                endif
            endif

        endif
    endfor
endfor
return π;
Stop.
```

Algorithm 5: $Check\_Tabu\_List(\pi)$

```
Start
if $Tabu\_List.Count = 0$ then
    Add $\pi$ to $Tabu\_List$
elseif $Tabu\_List.Count < 100$ then
    if $Tabu\_List.Contains(\pi)$=false then
        Add $\pi$ to $Tabu\_List$
    endif
else
    if $Tabu\_List.Contains(\pi)$=false then
        Remove oldest solution from $Tabu\_List$
        Add $\pi$ to $Tabu\_List$
    endif
endif
Stop
```

Evolutionary strategies take place when the number of forbidden solutions and the number of iterations with no improvement exceeds a certain number K as seen in Algorithm 1. This step is to escape local optima and produce a new solution population that may include new candidate solutions. In order to produce a new population, the previous population, the best solution found so far and the incumbent solution (the best solution in the current population) are used as seen Algorithm 6. The crossover operator in this study is a two-point crossover and the mutation operator is a swap-mutation. The encoding of the solution is permutation encoding, and each substring is defined with job indices. The other evolutionary operations such as evaluation and selection are not necessary because they increase the time requirement for obtaining a solution. Therefore, the crossover operation takes place for pairs of solutions in the order of $\{(\pi_1, \pi_2), (\pi_3, \pi_4), \ldots, (\pi_{59}, \pi_{60}\}$ with a probability p_c by selecting randomly two crossover points. Figure 2 shows a two-point crossover and repair operation for a solution pair. The mutation operator selects randomly two jobs in the solution and inverses the substring between these two randomly selected jobs with a probability p_m as seen in Fig. 3. The counter for local searches with non-improvement k is set as zero, and the new solution population is obtained by using the best solution in the memory and the incumbent solution with crossover and mutation operator. TS_{POP} algorithm runs until a predetermined stopping condition exists. In this study, the stopping condition for the TS_{POP} is the total elapsed time in milliseconds.

Algorithm 6: Generate_Population(Population, π^*, π)

```
Start
For j := 1 to 60 do
    Generate a random number $RND$ between 0 and 1;
    if $RND < 0.05$ then
        if $(j \bmod 2) = 0$ then
            replace solution $\pi_j$ with $\pi^*$;
        else
            replace solution $\pi_j$ with $\pi$;
        endif
    endif
endfor
return Population ;
Stop
```

5 Numerical examples

Taillard's (1993) test problems consisting of 20, 50, and 100 jobs with 5, 10, and 20 machines are used in order to show the performance of metaheuristic methods. For all problems, the learning effect coefficient and deterioration

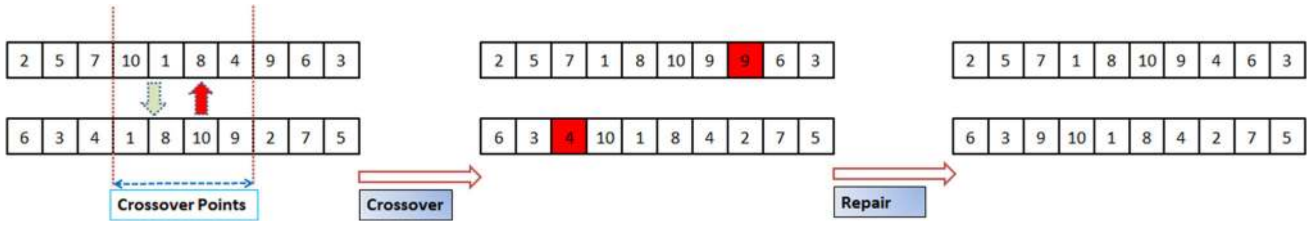


Fig. 2 Two-point crossover and repair operations

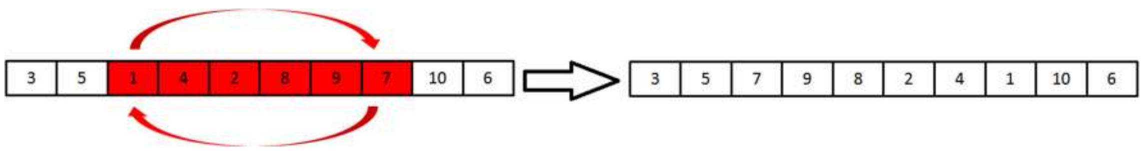


Fig. 3 The mutation operation

effect coefficient are − 0.8 and 0.1, respectively. While trying to find an optimum solution for a combinatorial problem; if time requirement for finding an optimum is exceedingly high, if the solver does not improve the solution and if the optimality gap is being reduced very slowly, then limiting execution time and optimality requirement by using metaheuristic methods can be reasonable. Metaheuristic methods try to find optimal solutions, but mostly they yield near-optimum ones. Therefore, parameter design in any metaheuristic is so significant. There are seven parameters for the proposed TS_{POP} algorithm, these are B (the predetermined number of solutions that will be used in the Local_Search_Population procedure), C (the predetermined number of local search iterations in the Local_Search procedure), the length of the Tabu list, K (a predetermined number of the maximum allowable iterations with no improvement in Algorithm 1), the crossover probability p_c and the mutation probability p_m. In our first experiments, we used lots of combinations of these parameters. We determined these parameters as $B = 5$, $C = 5$, the length of Tabu size is 100, $K = 10$, $p_c = 0.85$ and $p_c = 0.15$.

As rivals of the proposed TS_{POP}, we used two IG algorithms and a DDE algorithm for PFSS problems under the effects of learning and deterioration. IG algorithm for PFSS problems was firstly proposed by Ruiz and Stützle (2007). The IG algorithm is a single-solution metaheuristic method. In the IG algorithm for PFSS, the initial solution is obtained by using the well NEH heuristic. The IG algorithm (IG_RS) for PFSS problems applies two phases iteratively. These phases are names as destruction and construction. In the destruction phase of the algorithm, some jobs are removed from the incumbent solution. After the destruction phase of the algorithm, the removed jobs are reinserted into the partial solution to construct a complete solution again (the incumbent solution). Every time a removed job is inserted into the partial solution, a greedy selection among all possible positions that jobs can be inserted in the partial solution. In each iteration, a constant number (d) jobs are removed and reinserted. When a candidate solution has been completed, an acceptance criterion decides whether the new solution will replace the incumbent solution. IG_RS uses a simulated annealing like acceptance criterion with a constant temperature. This constant temperature is calculated as follows:

$$\text{Tempreature} = T \cdot \frac{\sum_i^n \sum_j^m p_{ij}}{n \cdot m \cdot 10} \tag{16}$$

where T is the second parameter of IG_RS to be adjusted for the temperature of simulated annealing like acceptance criterion. After the destruction and construction phase of the algorithm, an optional insertion-based local search (LS) can be adapted to increase the efficiency of the IG_RS algorithm. The LS operator randomly removes a job from the complete solution and reinserts it to all possible positions of the partial solution. If the LS operator finds better objective function value while inserting the removed job in different positions, the job is inserted into that position. This is repeated for another job. The process terminates when all jobs have been placed in all possible positions without improvements. The complexity of calculating makespan or flow time of a solution is $O(nm)$, and if there are k possible positions after removing a job, this complexity increases to $O(n^2m)$. Taillard (1990) proposed a mechanism named Taillard's acceleration in the following, so the evaluation of the k subsequences can be done in $O(nm)$ thus reducing the overall complexity of the heuristic to $O(n^2m)$. Taillard's acceleration can be used in any phase of IG algorithms such as NEH, destruction/construction, and local search. Of course, this acceleration schema works when the performance criterion is the minimization of the makespan or flowtime of a schedule when there is no effect such as learning and/or deterioration. This variant of the IG algorithm was named as IG_RS_{LS}. There are two parameters (d and T) of IG_RS_{LS}. Ruiz and Stützle (2007) suggested these parameters as $d = 4$ and $T = 0.4$ according to their parameter tuning. Another variant of the IG algorithm was proposed by Pan et al. (2008). This variant named as IG_{RIS} uses a referenced insertion schema (RIS) instead of LS proposed by Ruiz and Stützle (2007). This version of the local search operator uses a referenced solution obtained from a heuristic like NEH and to determine which jobs will be selected and removed from the complete solution. In RIS operator, jobs are not extracted randomly but in the order given by a referenced permutation. Pan et al. (2008) also suggested the same parameter setting ($d = 4$ and $T = 0.4$) for IG_{RIS} as Ruiz and Stützle (2007). The essential difference between IG_RS_{LS} and IG_{RIS} is using different local search procedure for the solution. If the IG uses the LS operator for the PFSS problem under learning and deterioration effects, it is IG_RS_{LS}. When it uses the RIS operator for the problem, it is IG_{RIS}.

Differential equation algorithm is a population-based solution method for the continuous optimization problem. Due to the discrete structure of the PFSS problem, Pan et al. (2008) proposed a DDE algorithm for the problem. In the DDE algorithm, the target individual is represented by a permutation of jobs. The previous generation's best solution in the target population is perturbed in order to obtain the mutant individual and achieve the differential variation. DDE algorithm uses a referenced local search (RLS) operator of the RIS for local search of individuals of the population. Pan et al. (2008) proposed the parameters of DDE_{RLS} as $d = 4$, population size is 10, $p_c = 0.80$ and $p_c = 0.20$. The IG and DDE algorithms used in this study is not different from the original algorithms proposed by Ruiz and Stützle (2007) and Pan et al. (2008). The only difference of the algorithms in this paper is that the algorithms do not use Taillard's acceleration schema for calculation of maximum completion time because of processing times under the effects of learning and deterioration.

In the literature for comparison of algorithms for PFSS problems, the time limitation (in milliseconds) of execution of algorithms is determined with the formula of $t \cdot n \cdot m/2$ where t is constant, n is the number of jobs and m is the number of machines. In this study, we used three different t values where $t \in \{30, 60, 90\}$ for comparison. Since there are no published upper bounds for the PFSS problem under the effects of learning and deterioration, we solved some of the test problems of Taillard's (1993) with a commercial solver. For the first 90 test instances (from 20 jobs with 5 machines to 100 jobs with 20 machines) of Taillard's (1993) benchmark problems, Commercial solver software, AIMMS, is used to solve test instances by using the MINLP model introduced in Sect. 3. While solving these problems in AIMMS, the execution of each problem is limited until 1000 s or reaching the solution's optimality gap to 0.0002. All metaheuristic algorithms (by using their original parameters) were coded with a standard desktop computer having an Intel i5 CPU and 8 GB RAM by using C# programming language with MS Access database. The well-known performance measure used to evaluate a solution method's performance for flow shop scheduling problems is the average relative percentage deviation (ARPD) as follows:

$$\sum_{i=1}^{R}\left(\frac{(f_i - f_{\text{best}})100}{f_{\text{best}}}\right)/R \tag{17}$$

where f_i is the objective function value of the proposed heuristic or metaheuristic method in ith independent run, f_{best} is the best-known solution (optimum or upper bound of optimum) for the problem, and R is the number of independent runs of the solution approach. R value was set as 5 for all test problems. In this study, we used the solutions obtained by using AIMMS solver as f_{best} values for test instances. These solutions of the first 90 problems of Taillard's (1993) benchmark problems and all results obtained from compared metaheuristics are available upon request for the readers. Table 1 shows ARPD values of compared algorithms when t value set as 30 for time limitation. Tables 2 and 3 show ARPD values of compared

Table 1 ARPD values of compared algorithms where t = 30

#of jobs	#of machines	IG_RS_{LS}	IG_{RIS}	DDE_{RLS}	TS_{POP}
20	5	0.00043	0.00038	0.00087	**0.00037**
20	10	0.00009	0.00007	0.00017	**0.00003**
20	20	**0.00000**	**0.00000**	0.01403	**0.00000**
50	5	0.00927	0.00666	0.01221	**0.00473**
50	10	0.00187	0.00173	0.00304	**0.00114**
50	20	**0.00000**	**0.00000**	0.00003	**0.00000**
100	5	0.02217	0.01687	0.03641	**0.01165**
100	10	0.01157	0.00796	0.01959	**0.00233**
100	20	**0.00000**	**0.00000**	0.00052	**0.00000**
Average		0.00504	0.00374	0.00965	**0.00225**

Table 2 ARPD values of compared algorithms where t = 60

#of jobs	#of machines	IG_RS_{LS}	IG_{RIS}	DDE_{RLS}	TS_{POP}
20	5	0.00046	0.00028	0.00077	**0.00027**
20	10	0.00003	**0.00002**	0.00041	**0.00002**
20	20	**0.00000**	**0.00000**	0.01403	**0.00000**
50	5	0.00779	0.00554	0.00891	**0.00332**
50	10	0.00193	0.00117	0.00184	**0.00087**
50	20	**0.00000**	**0.00000**	**0.00000**	**0.00000**
100	5	0.01745	0.01121	0.02933	**0.00838**
100	10	0.00823	0.00630	0.01348	**0.00210**
100	20	**0.00000**	**0.00000**	0.00004	**0.00000**
Average		0.00399	0.00273	0.00765	**0.00166**

Table 3 ARPD values of compared algorithms where t = 90

#of jobs	#of machines	IG_RS_{LS}	IG_{RIS}	DDE_{RLS}	TS_{POP}
20	5	0.00035	**0.00016**	0.00081	0.00017
20	10	0.00003	**0.00002**	0.00063	**0.00002**
20	20	**0.00000**	**0.00000**	0.01403	**0.00000**
50	5	0.00689	0.00509	0.00882	**0.00311**
50	10	0.00161	0.00072	0.00180	**0.00052**
50	20	**0.00000**	0.00002	0.00004	**0.00000**
100	5	0.01639	**0.00856**	0.02717	0.00927
100	10	0.00822	0.00584	0.01019	**0.00158**
100	20	**0.00000**	**0.00000**	0.00001	**0.00000**
Average		0.00372	0.00227	0.00706	**0.00163**

Table 4 Anova results of for comparison of solution methods

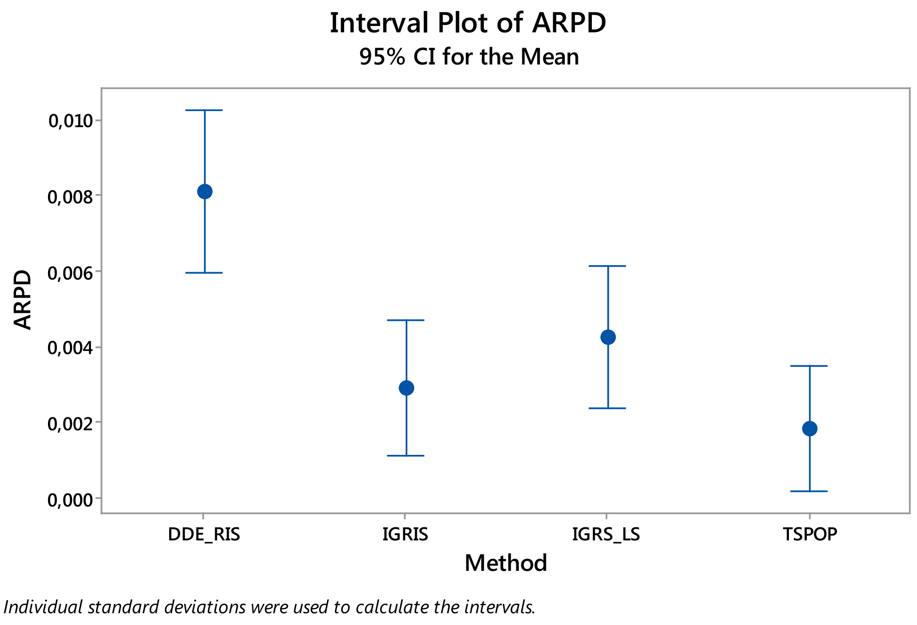
Source	*df*	Adj SS	Adj MS	*F* value	*p* value
n	2	0.010861	0.005431	24.06	0.000
m	2	0.009938	0.004969	22.02	0.000
$t{\cdot}n{\cdot}m/2$	2	0.000448	0.000224	0.99	0.371
Method	3	0.006081	0.002027	8.98	0.000
Error	1070	0.241590	0.000226		
Total	1079	0.268818			

algorithms where $t = 60$ and $t = 90$ for time limitation, respectively.

The best ARPD values are marked with bold font in Tables 1, 2 and 3 for each combination of #of jobs and #of machines. As seen from Tables 1, 2 and 3, the TS_{POP} algorithm has almost all of the best ARPD values for test instances. Since TS_{POP} algorithm has a mechanism to avoid cycling solutions in each execution of the problem, we checked how many times the proposed algorithm disables a cycling solution for all test instances within the experiment with $t = 30$. The average ratio for avoiding cycling solutions per problem is 9.73%. Thus, TS_{POP} does not use these solutions that were already found and improved in previous iterations. Furthermore, TS_{POP} generates new solutions that have chances to be new better solutions by escaping from cycling solutions. For better comparison, an ANOVA test was made with 95% confidence level for performance comparison. We tested the following factors: (1) the number of jobs (n), tested at three values: 20, 50, and 100. (2) The number of machines (m), tested at three values: 5, 10, and 20. (3) Type of methods, tested at four variants: IG_RS_{LS}, IG_{RIS}, DDE_{RLS}, and TS_{POP}. (4) Predetermined stopping criteria, tested at three variants: $t = 30$, $t = 60$ and $t = 90$. The detail of ANOVA test is given in Table 4. As seen from Table 4, all factors except predetermined stopping criteria ($t \cdot n \cdot m/2$) have a significant difference with 95% confidence level because these factors' p values are less than 0.05.

The ANOVA results in Table 4 show that there is a significant difference between solution methods. For a more detailed comparison, the interval plot of ARPD values in Fig. 4 shows that the TS_{POP} algorithm presents less ARPD values comparing other algorithms. If we consider ARPD values for each t value where $t \in \{30, 60, 90\}$, the interval plot in Fig. 5. For ARPD values of each algorithm for each t value show that the TS_{POP} algorithm outperforms other algorithms for each t value.

Fig. 4 Interval plot of ARPD values obtained by solution approaches



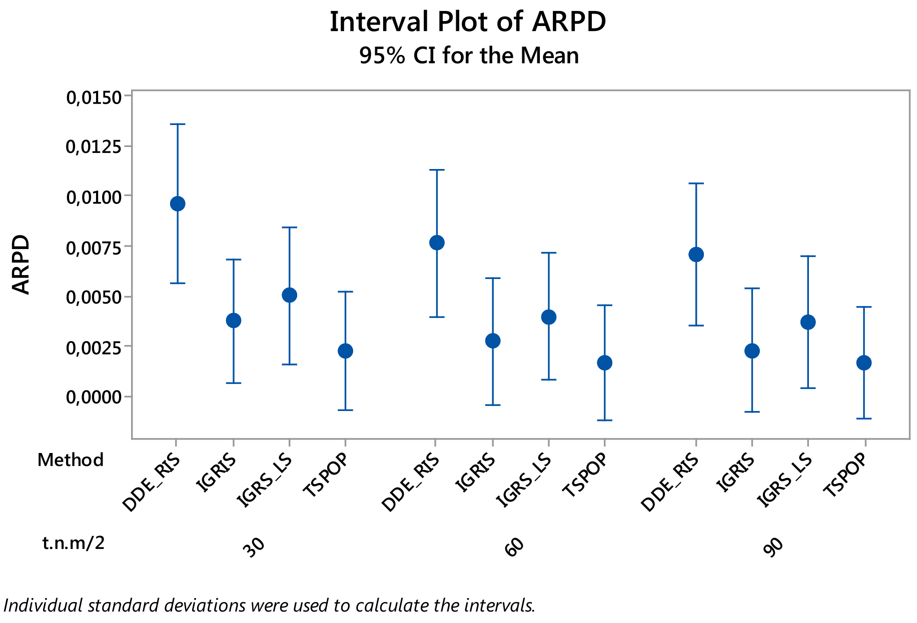


Fig. 5 Interval plot of ARPD values obtained by solution approaches for each t value

Table 5 Results of Wilcoxon signed-rank tests

Comparison	t = 30	t = 60	t = 90
$\text{IG\_RS}_{\text{LS}} - \text{TS}_{\text{POP}}$	0.000	0.000	0.000
$\text{IG}_{\text{RIS}} - \text{TS}_{\text{POP}}$	0.000	0.000	0.000
$\text{DDE}_{\text{RLS}} - \text{TS}_{\text{POP}}$	0.000	0.000	0.000

For a more detailed comparison, Wilcoxon signed-rank tests with a 95% confidence were done between the proposed TS_{POP} algorithm and other algorithms considering all t values ($t \in \{30, 60, 90\}$). The results of Wilcoxon signed-rank tests are given in Table 5. As seen in Table 5, all p values are less than 0.05. There are significant differences between TSPOP and any of the other algorithms for each t value. Therefore, we say that the proposed TSPOP algorithm outperforms extremely IG_RS_{LS}, IG_{RIS}, and DDE_{RLS} algorithms in all predetermined stopping criteria for PFSS problems under the effects of learning and deterioration.

6 Conclusion

In this study, PFSS problems under the effects of position-dependent learning and linear deterioration are studied when the objective function is to minimize the makespan. A hybrid solution algorithm called population-based Tabu search algorithm (TS_{POP}) and well-known heuristic methods (IG_RS_{LS}, IG_{RIS}, and DDE_{RLS}) are used to solve PFSS problems under the effects of dependent learning and linear deterioration. For comparison of solution approaches, some of Taillard's (1993) benchmark problems under the effects of learning and deterioration are solved with a commercial solver. These solutions are used in the comparison of the algorithms as upper bounds of the problems. The experimental results show that the proposed TS_{POP} outperforms other existing algorithms, then the problem's objective is to minimize the makespan with jobs under learning and deterioration effects. For future research, the results in this study can be used for benchmarks of other metaheuristic methods for PFSS problems under the effects of position-dependent learning and linear deterioration. Furthermore, the proposed TS$_{POP}$ algorithm can be used for sequence-dependent or flexible flow shop scheduling problems.

Compliance with ethical standards

Conflict of interest The author declares that there is no conflict of interest.

Appendix

See Table 6.

Table 6 Results and solution times of problems obtained from the commercial solver

n	m	Problem	Upper bound	Solution time (s)	*n*	*m*	Problem	Upper bound	Solution time (s)	*n*	*m*	Problem	Upper bound	Solution time (s)
20	5	TL001	400.162	0.26	50	5	TL031	475.827	904.06	100	5	TL061	534.778	1000.17
		TL002	446.704	0.28			TL032	447.998	358.26			TL062	567.550	1000.06
		TL003	413.841	4.21			TL033	424.271	1000.73			TL063	545.323	1005.08
		TL004	501.343	1.36			TL034	497.741	5.27			TL064	536.274	1000.29
		TL005	373.668	0.58			TL035	521.461	51.70			TL065	496.074	1000.23
		TL006	364.707	0.59			TL036	468.034	247.81			TL066	558.454	1000.12
		TL007	386.540	0.50			TL037	468.171	600.76			TL067	558.454	1000.98
		TL008	388.484	2.62			TL038	454.817	994.76			TL068	510.884	1000.20
		TL009	404.511	0.56			TL039	451.537	380.72			TL069	528.090	1001.82
		TL010	422.124	1.53			TL040	517.320	253.91			TL070	498.522	1001.04
20	10	TL011	961.889	0.28	50	10	TL041	1066.433	0.58	100	10	TL071	1207.515	3.14
		TL012	1097.729	0.53			TL042	1099.947	1.03			TL072	1040.739	853.03
		TL013	762.317	0.94			TL043	944.896	2.39			TL073	1123.721	15.99
		TL014	928.712	0.55			TL044	933.516	2.37			TL074	1083.033	15.38
		TL015	706.059	0.30			TL045	1012.559	1.86			TL075	1078.742	12.96
		TL016	764.093	0.11			TL046	1043.195	1.19			TL076	1152.094	13.09
		TL017	682.976	1.19			TL047	1102.802	6.80			TL077	987.159	1000.15
		TL018	886.098	1.72			TL048	940.027	99.17			TL078	1246.743	5.14
		TL019	889.479	0.50			TL049	680.305	1000.06			TL079	1143.643	28.89
		TL020	797.371	0.33			TL050	1117.525	1.15			TL080	1165.318	1000.48
20	20	TL021	3419.424	1.51	50	20	TL051	3815.445	1.68	100	20	TL081	3484.915	5.29
		TL022	3205.702	0.36			TL052	2944.699	1.75			TL082	4325.104	5.18
		TL023	3239.994	0.45			TL053	3071.987	1.58			TL083	4145.950	4.98
		TL024	3028.056	1.15			TL054	3432.194	1.11			TL084	4114.610	6.24
		TL025	3085.065	0.53			TL055	3411.574	1.51			TL085	3454.885	5.47
		TL026	3332.718	0.86			TL056	3292.718	1.05			TL086	3852.578	6.16
		TL027	3464.890	0.42			TL057	3378.442	2.25			TL087	3888.146	4.63
		TL028	3258.712	0.86			TL058	3661.708	1.58			TL088	4217.352	4.73
		TL029	3474.546	1.67			TL059	3708.572	1.68			TL089	3972.387	5.57
		TL030	3080.797	1.06			TL060	3708.914	1.61			TL090	3895.830	5.80

References

Abdinnour-Helm S (1998) A hybrid heuristic for the uncapacitated hub location problem. Eur J Oper Res 106:489–499. https://doi.org/10.1016/S0377-2217(97)00286-5

Arık OA, Toksarı MD (2018) Multi-objective fuzzy parallel machine scheduling problems under fuzzy job deterioration and learning effects. Int J Prod Res 56:2488–2505. https://doi.org/10.1080/00207543.2017.1388932

Arık OA, Toksarı MD (2019) Fuzzy parallel machine scheduling problem under fuzzy job deterioration and learning effects with fuzzy processing times. In: Ram M (ed) Advanced fuzzy logic approaches in engineering science. IGI Global, Pennsylvania, pp 49–67

Bank M, Fatemi Ghomi SMT, Jolai F, Behnamian J (2012) Application of particle swarm optimization and simulated annealing algorithms in flow shop scheduling problem under linear deterioration. Adv Eng Softw 47:1–6. https://doi.org/10.1016/j.advengsoft.2011.12.001

Benavides AJ, Ritt M (2018) Fast heuristics for minimizing the makespan in non-permutation flow shops. Comput Oper Res 100:230–243. https://doi.org/10.1016/j.cor.2018.07.017

Biskup D (1999) Single-machine scheduling with learning considerations. Eur J Oper Res 115:173–178. https://doi.org/10.1016/S0377-2217(98)00246-X

Chen S-H, Chang P-C, Cheng TCE, Zhang Q (2012) A self-guided genetic algorithm for permutation flowshop scheduling problems. Comput Oper Res 39:1450–1457. https://doi.org/10.1016/j.cor.2011.08.016

Cheng TCE, Wu CC, Lee WC (2008) Some scheduling problems with deteriorating jobs and learning effects. Comput Ind Eng 54:972–982. https://doi.org/10.1016/j.cie.2009.06.016

Chung Y-H, Tong L-I (2011) Makespan minimization for *m*-machine permutation flowshop scheduling problem with learning considerations. Int J Adv Manuf Technol 56:355–367. https://doi.org/10.1007/s00170-011-3172-2

Chung Y-H, Tong L-I (2012) Bi-criteria minimization for the permutation flowshop scheduling problem with machine-based learning effects. Comput Ind Eng 63:302–312. https://doi.org/10.1016/j.cie.2012.03.009

Dong X, Huang H, Chen P (2008) An improved NEH-based heuristic for the permutation flowshop problem. Comput Oper Res 35:3962–3968. https://doi.org/10.1016/j.cor.2007.05.005

Dubois-Lacoste J, Pagnozzi F, Stützle T (2017) An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. Comput Oper Res 81:160–166. https://doi.org/10.1016/j.cor.2016.12.021

Fernandez-Viagas V, Framinan JM (2014) On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. Comput Oper Res 45:60–67. https://doi.org/10.1016/j.cor.2013.12.012

Fernandez-Viagas V, Framinan JM (2019) A best-of-breed iterated greedy for the permutation flowshop scheduling problem with makespan objective. Comput Oper Res 112:104767. https://doi.org/10.1016/j.cor.2019.104767

Fernandez-Viagas V, Ruiz R, Framinan JM (2017) A new vision of approximate methods for the permutation flowshop to minimise makespan: state-of-the-art and computational evaluation. Eur J Oper Res 257:707–721. https://doi.org/10.1016/j.ejor.2016.09.055

Framinan JM, Leisten R, Ruiz-Usano R (2002) Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. Eur J Oper Res 141:559–569. https://doi.org/10.1016/S0377-2217(01)00278-8

Framinan JM, Gupta JND, Leisten R (2004) A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. J Oper Res Soc 55:1243–1255. https://doi.org/10.1057/palgrave.jors.2601784

Geng X-N, Wang J-B, Bai D (2019) Common due date assignment scheduling for a no-wait flowshop with convex resource allocation and learning effect. Eng Optim 51:1301–1323. https://doi.org/10.1080/0305215X.2018.1521397

Glover F (1989) Tabu search—part I. ORSA J Comput 1:190–206

Glover F (1990) Tabu search—part II. ORSA J Comput 2:4–32

Glover F, Kelly JP, Laguna M (1995) Genetic algorithms and Tabu search: hybrids for optimization. Comput Oper Res 22:111–134. https://doi.org/10.1016/0305-0548(93)E0023-M

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. J Sched 11:357–370. https://doi.org/10.1007/s10951-008-0064-x

Graham RL, Lawler EL, Lenstra JK, Kan R (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

Gupta JND, Gupta SK (1988) Single facility scheduling with nonlinear processing times. Comput Ind Eng 14:387–393

Haq AN, Ramanan TR, Shashikant KS, Sridharan R (2010) A hybrid neural network–genetic algorithm approach for permutation flow shop scheduling. Int J Prod Res 48:4217–4231. https://doi.org/10.1080/00207540802404364

He H (2016) Minimization of maximum lateness in an *m*-machine permutation flow shop with a general exponential learning effect. Comput Ind Eng 97:73–83. https://doi.org/10.1016/j.cie.2016.04.010

Janiak A, Portmann M-C (1998) Genetic algorithm for the permutation flow-shop scheduling problem with linear models of operations. Ann Oper Res 83:95–114

Jat SN, Yang S (2011) A hybrid genetic algorithm and Tabu search approach for post enrolment course timetabling. J Sched 14:617–637. https://doi.org/10.1007/s10951-010-0202-0

Kalczynski PJ, Kamburowski J (2008) An improved NEH heuristic to minimize makespan in permutation flow shops. Comput Oper Res 35:3001–3008. https://doi.org/10.1016/j.cor.2007.01.020

Laha D, Chakraborty UK (2009) An efficient hybrid heuristic for makespan minimization in permutation flow shop scheduling. Int J Adv Manuf Technol 44:559–569. https://doi.org/10.1007/s00170-008-1845-2

Lee W-C, Chung Y-H (2013) Permutation flowshop scheduling to minimize the total tardiness with learning effects. Int J Prod Econ 141:327–334. https://doi.org/10.1016/j.ijpe.2012.08.014

Lee W-C, Wu C-C, Chung Y-H, Liu H-C (2009) Minimizing the total completion time in permutation flow shop with machine-dependent job deterioration rates. Comput Oper Res 36:2111–2121. https://doi.org/10.1016/j.cor.2008.07.008

Lee W-C, Yeh W-C, Chung Y-H (2014) Total tardiness minimization in permutation flowshop with deterioration consideration. Appl Math Model 38:3081–3092. https://doi.org/10.1016/j.apm.2013.11.031

Li X, Gao L (2016) An effective hybrid genetic algorithm and Tabu search for flexible job shop scheduling problem. Int J Prod Econ 174:93–110. https://doi.org/10.1016/j.ijpe.2016.01.016

Li JR, Khoo LP, Tor SB (2003) A Tabu-enhanced genetic algorithm approach for assembly process planning. J Intell Manuf 14:197–208. https://doi.org/10.1023/A:1022903514179

Li B-B, Wang L, Liu B (2008) An effective PSO-based hybrid algorithm for multiobjective permutation flow shop scheduling. IEEE Trans Syst Man Cybern Part A Syst Hum 38:818–831. https://doi.org/10.1109/TSMCA.2008.923086

Liaw C-F (2000) A hybrid genetic algorithm for the open shop scheduling problem. Eur J Oper Res 124:28–42

Lin Q, Gao L, Li X, Zhang C (2015) A hybrid backtracking search algorithm for permutation flow-shop scheduling problem. Comput Ind Eng 85:437–446. https://doi.org/10.1016/j.cie.2015.04.009

Lu Y-Y (2016) Research on no-idle permutation flowshop scheduling with time-dependent learning effect and deteriorating jobs. Appl Math Model 40:3447–3450. https://doi.org/10.1016/j.apm.2015.09.081

McCormick ST, Pinedo ML, Shenker S, Wolf B (1989) Sequencing in an assembly line with blocking to minimize cycle time. Oper Res 37:925–935. https://doi.org/10.1287/opre.37.6.925

Meeran S, Morshed MS (2012) A hybrid genetic Tabu search algorithm for solving job shop scheduling problems: a case study. J Intell Manuf 23:1063–1078. https://doi.org/10.1007/s10845-011-0520-x

Mosheiov G (1991) V-shaped policies for scheduling deteriorating jobs. Oper Res 39:979–991. https://doi.org/10.1287/opre.39.6.979

Muştu S, Eren T (2018) Maximum completion time under a learning effect in the permutation flowshop scheduling problem. Int J Ind Eng Theory Appl Pract 25:156–174

Nagano MS, Ruiz R, Lorena LAN (2008) A constructive genetic algorithm for permutation flowshop scheduling. Comput Ind Eng 55:195–207. https://doi.org/10.1016/j.cie.2007.11.018

Osman I, Potts C (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17:551–557. https://doi.org/10.1016/0305-0483(89)90059-5

Palacios JJ, González MA, Vela CR et al (2015) Genetic Tabu search for the fuzzy flexible job shop problem. Comput Oper Res 54:74–89. https://doi.org/10.1016/j.cor.2014.08.023

Pan Q-K, Tasgetiren MF, Liang Y-C (2008) A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput Ind Eng 55:795–816. https://doi.org/10.1016/j.cie.2008.03.003

Pasupathy T, Rajendran C, Suresh RK (2006) A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. Int J Adv Manuf Technol 27:804–815. https://doi.org/10.1007/s00170-004-2249-6

Qin H, Zhang Z-H, Bai D (2016) Permutation flowshop group scheduling with position-based learning effect. Comput Ind Eng 92:1–15. https://doi.org/10.1016/j.cie.2015.12.001

Rajkumar R, Shahabudeen P (2009) An improved genetic algorithm for the flowshop scheduling problem. Int J Prod Res 47:233–249. https://doi.org/10.1080/00207540701523041

Reza Hejazi S, Saghafian S (2005) Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res 43:2895–2929. https://doi.org/10.1080/0020754050056417

Rossi FL, Nagano MS, Neto RFT (2016) Evaluation of high performance constructive heuristics for the flow shop with makespan minimization. Int J Adv Manuf Technol 87:125–136

Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177:2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

Ruiz R, Stützle T (2008) An iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 187:1143–1159. https://doi.org/10.1016/j.ejor.2006.07.029

Shi H-B, Wang J-B (2019) Research on common due window assignment flowshop scheduling with learning effect and resource allocation. Eng Optim. https://doi.org/10.1080/0305215X.2019.1604698

Shiau Y-R, Tsai M-S, Lee W-C, Cheng TCE (2015) Two-agent two-machine flowshop scheduling with learning effects to minimize the total completion time. Comput Ind Eng 87:580–589. https://doi.org/10.1016/j.cie.2015.05.032

Sun H, Yu J, Wang H (2015) Multi-population and self-adaptive genetic algorithm based on simulated annealing for permutation flow shop scheduling problem. In: Deng Z, Li H (eds) Proceedings of the 2015 Chinese intelligent automation conference. Springer, Berlin, Heidelberg, pp 11–19. https://doi.org/10.1007/978-3-662-46466-3_2

Sun L-H, Ge C-C, Zhang W et al (2019) Permutation flowshop scheduling with simple linear deterioration. Eng Optim 51:1281–1300. https://doi.org/10.1080/0305215X.2018.1519558

Suresh RK, Mohanasundaram KM (2004) Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives. In: 2004 IEEE Conference on cybernetics and intelligent systems

Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47:65–74. https://doi.org/10.1016/0377-2217(90)90090-X

Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285. https://doi.org/10.1016/0377-2217(93)90182-M

Toksarı MD, Arık OA (2017) Single machine scheduling problems under position-dependent fuzzy learning effect with fuzzy processing times. J Manuf Syst 45:159–179. https://doi.org/10.1016/j.jmsy.2017.08.006

Toksarı MD, Güner E (2008) Minimizing the earliness/tardiness costs on parallel machine with learning effects and deteriorating jobs: a mixed nonlinear integer programming approach. Int J Adv Manuf Technol 38:801–808. https://doi.org/10.1007/s00170-007-1128-3

Toksarı MD, Güner E (2010) Parallel machine scheduling problem to minimize the earliness/tardiness costs with learning effect and deteriorating jobs. J Intell Manuf 21:843–851. https://doi.org/10.1007/s10845-009-0260-3

Tseng L-Y, Lin Y-T (2010) A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. Int J Prod Econ 127:121–128. https://doi.org/10.1016/j.ijpe.2010.05.003

Vasiljevic D, Danilovic M (2015) Handling ties in heuristics for the permutation flow shop scheduling problem. J Manuf Syst 35:1–9. https://doi.org/10.1016/j.jmsy.2014.11.011

Wang J-B (2006) A note on scheduling problems with learning effect and deteriorating jobs. Int J Syst Sci 37:827–833. https://doi.org/10.1080/00207720600879260

Wang J-B (2007) Single-machine scheduling problems with the effects of learning and deterioration. Omega 35:397–402. https://doi.org/10.1016/j.omega.2005.07.008

Wang J-B (2009) Single-machine scheduling with learning effect and deteriorating jobs. Comput Ind Eng 57:1452–1456. https://doi.org/10.1016/j.cie.2009.07.015

Wang J-B, Liang X-X (2019) Group scheduling with deteriorating jobs and allotted resource under limited resource availability constraint. Eng Optim 51:231–246. https://doi.org/10.1080/0305215X.2018.1454442

Wang J-B, Wang C (2011) Single-machine due-window assignment problem with learning effect and deteriorating jobs. Appl Math Model 35:4017–4022. https://doi.org/10.1016/j.apm.2011.02.023

Wang J-B, Wang M-Z (2013) Minimizing makespan in three-machine flow shops with deteriorating jobs. Comput Oper Res 40:547–557. https://doi.org/10.1016/j.cor.2012.08.006

Wang J-J, Zhang B-H (2015) Permutation flowshop problems with bi-criterion makespan and total completion time objective and position-weighted learning effects. Comput Oper Res 58:24–31. https://doi.org/10.1016/j.cor.2014.12.006

Wang JB, Lin L, Shan F (2008a) Flow shop scheduling with effects of learning and deterioration. J Appl Math Comput 26:367–379. https://doi.org/10.1007/s12190-007-0033-0

Wang JB, Ng CT, Cheng TCE, Liu LL (2008b) Single-machine scheduling with a time-dependent learning effect. Int J Prod Econ 111:802–811. https://doi.org/10.1016/j.ijpe.2007.03.013

Wang J-B, Gao M, Wang J-J et al (2019a) Scheduling with a position-weighted learning effect and job release dates. Eng Optim. https://doi.org/10.1080/0305215X.2019.1664498

Wang J-B, Liu F, Wang J-J (2019b) Research on *m*-machine flow shop scheduling with truncated learning effects. Int Trans Oper Res 26:1135–1151. https://doi.org/10.1111/itor.12323

Xiao Y-Y, Zhang R-Q, Zhao Q-H, Kaku I (2012) Permutation flow shop scheduling with order acceptance and weighted tardiness. Appl Math Comput 218:7911–7926. https://doi.org/10.1016/j.amc.2012.01.073

Xu J, Lin W-C, Wu J et al (2016) Heuristic based genetic algorithms for the re-entrant total completion time flowshop scheduling with learning consideration. Int J Comput Intell Syst 9:1082–1100. https://doi.org/10.1080/18756891.2016.1256572

Yang DL, Kuo WH (2010) Some scheduling problems with deteriorating jobs and learning effects. Comput Ind Eng 58:25–28. https://doi.org/10.1016/j.cie.2009.06.016

Yenisey MM, Yagmahan B (2014) Multi-objective permutation flow shop scheduling problem: literature review, classification and current trends. Omega (United Kingdom) 45:119–135. https://doi.org/10.1016/j.omega.2013.07.004

Yin N, Kang L (2015) Minimizing makespan in permutation flow shop scheduling with proportional deterioration. Asia Pac J Oper Res 32:15500505. https://doi.org/10.1142/S0217595915500505

Zhang L, Gao L, Li X (2013) A hybrid genetic algorithm and Tabu search for a multi-objective dynamic job shop scheduling problem. Int J Prod Res 51:3516–3531. https://doi.org/10.1080/00207543.2012.751509

Zobolas GI, Tarantilis CD, Ioannou G (2009) Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Comput Oper Res 36:1249–1267. https://doi.org/10.1016/j.cor.2008.01.007