Computers & Operations Research 138 (2022) 105604

Contents lists available at ScienceDirect

Computers and Operations Research

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

Minimizing the total tardiness and the total carbon emissions in the permutation flow shop scheduling problem

Reza Ghorbani Saber, Mohammad Ranjbar *

Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ARTICLE INFO

Keywords:
Flow shop scheduling
Total tardiness
Total carbon emissions
Heuristic algorithm
Multi-objective optimization
VNS algorithm

ABSTRACT

In this paper, we consider the permutation flow shop scheduling problem and aim to minimize the total tardiness as well as the total carbon emissions. We present a formulation of the problem through a mixed-integer programming model. To solve the problem, we develop a multi-objective decomposition-based heuristic (MODBH) algorithm, working based on job insertion, as well as a multi-objective VNS algorithm. Furthermore, a multi-objective iterated greedy algorithm is utilized to validate the efficiency of the developed methods. Using extensive computational experiments, we indicate that the MODBH algorithm has a significant superiority to the other developed solution approaches. Furthermore, the multi-objective VNS algorithm shows better performance than the multi-objective iterated greedy algorithm.

1. Introduction

In recent decades, global warming and climate change have been vital issues and are being exacerbated by the increasing amount of greenhouse gas emissions across the world. One of the best-known greenhouse gases is carbon dioxide (CO2) which is a major byproduct of manufacturing activities. To alleviate the problem, a lot of research has been conducted to reduce the amount of carbon emissions by changing the schedule of machines in manufacturing facilities. In this context, the flow shop scheduling problem (FSSP) is a kind of machine scheduling problem which has gained significant attention.

In a flow shop environment, there is a set of jobs that have to be processed on a set of machines where all jobs visit machines with an identical sequence. The permutation flow shop scheduling problem (PFSSP) is one of the common simplified versions of the FSSP in which jobs have an identical sequence on all machines.

The PFSSP with typical manufacturing goals such as minimizing the makespan, the total flow times, the total tardiness and so forth, has been vastly considered in the literature e.g. Framinan et al. (2004), Ruiz and Maroto (2005) and Pan and Ruiz (2013). The total tardiness is the criterion that evaluates customer satisfaction and is chiefly important in the make-to-order systems because if the given due date of each job is violated, the corresponding customer may well incline to change his/her supplier. Several heuristics and metaheuristics approaches have been developed for the PFSSP with the total tardiness goal such as Rajendran and Ziegler (2003), Framinan and Leisten (2008) and Vallada and Ruiz (2010). A more detailed review was provided by Vallada et al. (2008). One of the best-known heuristics for the PFSSP was developed by Kim (1993), dubbed the NEHedd heuristic. The NEHedd works based on the NEH insertion heuristic, developed by Nawaz et al. (1983). Afterwards, some acceleration methods were developed to improve the performance of the NEHedd heuristic by Framinan and Leisten (2008), Fernandez-Viagas and Framinan (2015) and Fernandez-Viagas et al. (2020).

The green permutation flow shop scheduling problem (GPFSSP) is a generalization of the PFSSP in which environmental criteria such as carbon emissions are captured. In the literature on green scheduling, two strategies have been employed so far. The first strategy switches off idle machines to reduce energy consumption (Mouzon et al. (2007) and Mouzon and Yildirim (2008)). The second strategy benefits from controllable processing speeds of machines, implying machines have different processing speeds leading to different energy consumption and consequently different magnitude of carbon emissions. In other words, the more processing speed, the more carbon emissions. This strategy has recently gained a great deal of consideration; Yüksel et al. (2020) listed several speed-scaling approaches.

The PFSSP with environmental concerns has been investigated with several objective functions in the literature. Fang et al. (2013) considered the PFSSP with a restriction on peak power consumption in conjunction with some time-based objective functions. They developed a mathematical programming model and combinatorial approaches to

* Corresponding author.
E-mail address: m_ranjbar@um.ac.ir (M. Ranjbar).

https://doi.org/10.1016/j.cor.2021.105604
Received 9 September 2020; Received in revised form 17 August 2021; Accepted 12 October 2021
Available online 24 October 2021

solve the problem. Moreover, Zhang et al. (2014) studied a flexible flow shop scheduling problem with no tardy jobs and aimed to minimize energy costs under time-of-use tariffs. They proposed a hybrid genetic algorithm to solve large instances in reasonable CPU runtimes. Mansouri et al. (2016) investigated the GPFSSP with two machines and analyzed the trade-off between minimizing makespan and total energy consumption. They developed a mixed-integer linear multi-objective optimization model to obtain the Pareto frontier encompassing the makespan and the total energy consumption. Likewise, Ding et al. (2016) captured the GPFSSP with the goals of minimizing the total energy consumption (TEC) and the makespan. They extended the NEH-insertion procedure to consider the energy criterion and developed two multi-objective optimization algorithms, named MONEH and MMOIG. Using computational results, they showed that MMOIG was the best-developed algorithm and both MONEH and MMOIG algorithms outperform other metaheuristics e.g. NSGA-II, AMGA and IGSA. Furthermore, Öztop et al. (2020) conducted research on the energy-efficient PFSSP with two objective functions: the total flow time and the total energy consumption. They developed a bi-objective mathematical model, a novel heuristic for the initial solution generation, and three bi-objective metaheuristics. They compared the solutions of the metaheuristics with the solutions of the mathematical model. Moreover, Yüksel et al. (2020) studied the no-wait permutation flow shop scheduling problem with the total tardiness and the total energy consumption goals. They developed a mixed-integer programming (MIP) model and a constraint programming model as well as a set of metaheuristics including a novel multi-objective discrete artificial bee colony algorithm, a traditional multi-objective genetic algorithm, and a variant of multi-objective genetic algorithm with a local search. Interested readers are referred to Yenisey and Yagmahan (2014) for a review on the multi-objective flow shop scheduling problem.

There is also valuable research in the literature in the field of stochastic multi-objective permutation flow shop scheduling problems. In the research conducted by Fu et al. (2019), a stochastic multi-objective energy-conscious distributed PFSSP was studied. They targeted to minimize the makespan and the total energy consumption under a constraint on the total tardiness. To deal with the problem, they designed a multi-objective brain storm optimization algorithm, incorporated with stochastic simulation, and compared it with NSGA-II based on a set of benchmark test instances. Amiri and Behnamian (2020) considered the PFSSP with stochastic processing times and aimed to optimize the makespan and the total energy consumption. They presented a mathematical model with a finite number of scenarios and they implemented it on small-size instances. For large-size instances, they developed a scenario-based estimation of distribution algorithm, and they showed its superiority over a proposed genetic algorithm.

In this paper, we address the following research gaps. First, we introduce the PFSSP with the goals of minimizing the total tardiness and the total carbon emissions, making a balance between customer satisfaction and sustainability. Next, we develop two efficient bi-objective solution approaches dominating previously developed ones in the literature. Since the PFSSP with the total tardiness objective function is NP-hard for two or a higher number of machines (Garey et al. (1976)), the problem at hand, which is a generalization of that, is NP-hard as well.

The contribution of this paper is twofold. Firstly, we develop an efficient multi-objective decomposition-based heuristic (MODBH) algorithm, showing better performance than the state-of-the-art algorithms for solving the GPFSSP. Secondly, we develop a multi-objective variable neighborhood search (MOVNS) algorithm to generate high-quality Pareto-solutions in reasonable times.

The remainder of this paper is organized as follows. We describe and formulate the problem in Section 2. Our developed MODBH algorithm is presented in Section 3. In Section 4, a multi-objective variable neighborhood search (MOVNS) algorithm is developed. Computational results are discussed in Section 5. Finally, a summary and outlook on future research opportunities are provided in Section 6.

2. Problem description and modeling

In this section, we present a MIP formulation for GPFSP, used in our MODBH algorithm. This formulation has a great deal in common with formulations developed by Yüksel et al. (2020), Öztop et al. (2020), Foumani and Smith-Miles (2019) and Birgin et al. (2020). What sets this modeling apart from other formulations is considering the total tardiness in conjunction with the total carbon emissions in the model.

We consider a set of jobs, denoted by $N = \{1, \cdots, n\}$, that must be processed on a set of machines, shown by $M = \{1, \cdots, m\}$. Each job visits all machines based on the ascending order of machines number. Each machine can work at a finite set of different processing speeds $S = \{v_1, \cdots, v_s\}$. We assume that each job $j \in N$ has a standard processing time on machine $i \in M$, denoted by p_{ij}. Regarding processing speeds, we define the processing time of job j on machine i at processing speed $v \in S$ as $p_{ijv} = \frac{p_{ij}}{v}$. This gives rise to a particular amount of carbon emissions shown by $q_{ijv} = \gamma \pi_{iv} p_{ijv}$ where π_{iv} indicates the power of machine i when it works at processing speed v, and γ shows the coefficient of carbon emissions per unit of energy consumption. It is worth mentioning that we must keep the machines on until all jobs are processed; π'_i indicates the power consumption of machine i when it runs in the idle mode. For each machine the higher the speed of the machine is, the more carbon emissions would be. In other words, for each pair of arbitrary speeds v, $v' \in S$, if $v' > v$, then we have $p_{ijv'} < p_{ijv}$ and $q_{ijv'} > q_{ijv}$ (Ding et al. (2016)). In the following, we develop a MIP formulation for the problem. Let us first introduce the decision variables of the MIP model as follows.

$$Z_{jk} = \begin{cases} 1; \textit{if job } j \textit{ precedes job } k \textit{ in the sequence} \\ 0; \textit{otherwise} \end{cases}$$

$$X_{ijv} = \begin{cases} 1; \textit{if machine } i \textit{ processes job } j \textit{ at processing speed } v \\ 0; \textit{otherwise} \end{cases}$$

The optimal sequence of jobs processing is determined using variables Z_{jk}, and the optimal speed of each job on each machine is fixed using variables X_{ijv}. The total carbon emissions denoted by TCE can be calculated as follows:

$$TCE = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{v=v_1}^{v_s} q_{ijv} X_{ijv} + \sum_{i=1}^{m} \gamma \pi'_i (CC_i - \sum_{j=1}^{n} \sum_{v=v_1}^{v_s} p_{ijv} X_{ijv}) \quad (1)$$

Let us introduce C_{ij} as the completion time of job j on machine i. In equation (1), $CC_i = C_{i(n)}$ shows the completion time of the last job on machine i in the sequence, determined by the MIP model. Moreover, the completion time of the k^{th} job on machine i in the sequence, denoted as $C_{i(k)}$, is calculated as below.

$$C_{i(k)} = \max(C_{i-1(k)} + \sum_{v=v_1}^{v_s} p_{ikv} X_{ikv}, C_{i(k-1)} + \sum_{v=v_1}^{v_s} p_{ikv} X_{ikv}) \quad (2)$$

Let us introduce T_j and d_j as the tardiness and the due date of job j, respectively. The total tardiness objective function is expressed as $Min \sum_{j=1}^{n} T_j$ where $T_j = \max(C_{mj} - d_j, 0)$. The MIP model reads as follows.

MIP:

$$Min \sum_{j=1}^{n} T_j \quad (3)$$

$$Min(TCE) \quad (4)$$

$$T_j \geq C_{mj} - d_j, \qquad \forall j \in N \quad (5)$$

$$\sum_{v=1}^{s} X_{ijv} = 1, \qquad \forall j \in N; \forall i \in M \quad (6)$$

$$C_{1j} \geq \sum_{v=1}^{s} p_{1jv} X_{1jv} - R\left(n - 1 - \sum_{k=1}^{n} Z_{jk}\right), \qquad \forall j \in N \tag{7}$$

$$C_{ij} \geq C_{i-1j} + \sum_{v=1}^{s} p_{ijv} X_{ijv}, \qquad \forall j \in N; \forall i > 1 \in M \tag{8}$$

$$C_{ij} \geq C_{ik} + \sum_{v=1}^{s} p_{ijv} X_{ijv} - RZ_{jk}, \qquad \forall j,k \in N; \forall i \in M \tag{9}$$

$$C_{ik} \geq C_{ij} + \sum_{v=1}^{s} p_{ijv} X_{ijv} - R(1 - Z_{jk}), \qquad \forall j,k \in N; \forall i \in M \tag{10}$$

$$CC_i \geq C_{ij}, \qquad \forall j \in N; \forall i \in M \tag{11}$$

$$Z_{jk} + Z_{kj} = 1, \qquad \forall j,k \in N : j \neq k \tag{12}$$

$$Z_{hj} + Z_{jk} - Z_{hk} \leq 1, \qquad \forall j,k,h \in N : j \neq k \neq h \tag{13}$$

$$Z_{jj} = 0, \qquad \forall j \in N \tag{14}$$

$$Z_{jk} \in \{0,1\}, \qquad \forall j,k \in N \tag{15}$$

$$X_{ijv} \in \{0,1\}, \qquad \forall j \in N; \forall i \in M; \forall v \in S \tag{16}$$

$$C_{ij} \geq 0, \qquad \forall j \in N; \forall i \in M \tag{17}$$

$$CC_i \geq 0, \qquad \forall i \in M \tag{18}$$

$$T_j \geq 0, \qquad \forall j \in N \tag{19}$$

Objective functions (3) and (4) indicate two aforementioned goals: minimizing the total tardiness and the total carbon emissions. Constraints (5) and (6) calculate the tardiness of each job. Constraints (6) indicate that each machine can process each job at only one processing speed. Moreover, Constraints (7) represent the completion time of the first job on the first machine, determined by the sequence. It should be noted that for each job j, $\sum_{k=1}^{n} Z_{jk}$ indicates the position of that job in the sequence by calculating the number of jobs placed after that. It is worthwhile to mention that R shows a sufficiently big number in this model. Constraints (8), (9) and (10) determine the completion times of jobs. Constraints (11) alludes to the completion time of the last job on each machine i based on the determined sequence. Constraints (12) and (13) guarantee the ordinal relation between jobs in the sequence and avoid cycling. Furthermore, according to Constraints (14), no job precedes itself. Finally, Constraints (15) to (19) indicate the range of decision variables.

3. A multi-objective decomposition-based heuristic algorithm

In this section, a description of our MODBH algorithm is presented. This algorithm functions based on the idea of job insertion and a set of decomposed MIP models are deployed to insert jobs one by one in the sequence with proper processing speeds.

It should be noted that Kopanos et al. (2010) utilized a similar decomposition-based approach to solve a scheduling problem in multi-product multistage batch manufacturing. They developed a method based on solving decomposed MIP models in an iterative insertion procedure. They also developed an improvement rescheduling stage through solving a decomposition-based MIP. They used the idea of inserting jobs sequentially through solving MIP models. In contrast to this research, they consider a single-objective problem and they do not deal with speed scaling framework for machines.

3.1. The general sketch of the MODBH algorithm

The MODBH algorithm decomposes the solving process of the GPFSP, including n jobs, into $(n-1)$ stages. The MODBH algorithm takes an ordered list of jobs as input and solves the partial problem including the first $k+1$ jobs of the list in stage $k(k = 1, \cdots, n-1)$. We show each solution including k jobs by sol^k. In stage $k(k = 1, .., n-1)$, a set of non-dominated partial solutions NPS^{k+1} is obtained where each solution sol^k leads to a set of new solutions in stage $k+1$ using the *insertion* algorithm, shown as *InsA* and described in Section 3.4, and the well-known ε-constraint method. We indicate how NPS^2 is initially created through the *initialization* algorithm in Section 3.2. Furthermore, in order to control the size of the solution population, we consider a maximum value for it, shown by pop_{max}, and we choose an elite set of solutions having maximum distance with one another in each stage using the *Elite-Pop* algorithm, described in Section 3.5.

3.2. The initialization algorithm

As a pre-processing step, we construct an ordered list of jobs based on the non-descending order of o_j where $o_j = d_j - \sum_{i=1}^{m} p_{ij}$, using increasing job number as tie breaking rule. For each job $j \in N$, o_j considers both the total tardiness and the total carbon emissions. The jobs having lower d_j are given more priority because they might contribute to more increase in the total tardiness. Moreover, the jobs having a greater magnitude of the sum of processing times should be considered earlier for the determination of processing speed because they might wreak a greater deal of carbon emissions. The ordered jobs are placed in list $N^{'}$ where $N^{'}(k)$ indicates the k^{th} element of list $N^{'}$.

The first set of non-dominated partial solutions where each includes the first two jobs of list $N^{'}$ is created using the initialization algorithm, shown by Algorithm 1. Note that in this section, the expression $\text{MIP}^{(k)}(\Omega)$ alludes to a model originated from the MIP model, described in Section 2, in which jobs $N^{'}(1), \cdots, N^{'}(k)$ constitute a partial problem of the GPFSSP and a set of additional constraints Ω is imposed. By $X-relaxed$, we mean that variables $X_{ijv}; \forall j \in N; \forall i \in M; \forall v \in S$ are relaxed to accelerate the implementation of the model. Therefore, $\text{MIP}^{(k)}\left(X-relaxed, \sum T_j\right)$ indicates the $\text{MIP}^{(k)}$ model in which all variables X_{ijv} are relaxed and the total tardiness is considered as the single objective function of the model. It is worth mentioning that the relaxation of X-variables might give rise to infeasible solutions because they may get real values. In this case, we make the solution feasible by means of a repair procedure. For each machine i and job j, the repair procedure simply sets non-integer variable X_{ijv} to one if $X_{ijv} = \max\left(X_{ijv^{'}}\right); \forall v^{'} \in S$, otherwise it gets zero.

Algorithm 1: The initialization algorithm

1. Take $N^{'}(1)$ and $N^{'}(2)$ as input and let $NPS^2 = \varnothing$
2. $sol^2 \leftarrow \text{MIP}^{(2)}\left(X-relaxed, \sum T_j\right)$, repair sol^2 (if required), calculate $TCE(sol^2)$ and let $UB \leftarrow TCE(sol^2)$.
3. $NPS^2 \leftarrow NPS^2 \cup sol^2$.
4. $sol^2 \leftarrow \text{MIP}(X-relaxed, TCE)$, repair sol^2 and let $LB \leftarrow TCE(sol^2)$.
5. $NPS^2 \leftarrow NPS^2 \cup sol^2$.
6. **For** $i = 1$ to λ **do**
7. $\varepsilon = UB - i \times \dfrac{UB - TLB}{\lambda + 1}$
8. $sol^2 \leftarrow \text{MIP}\left(X-relaxed, \sum T_j, TCE \leq \varepsilon\right)$ and repair sol^2, if needed.
9. If sol^2 is not dominated by solutions of NPS^2, then let $NPS^2 \leftarrow NPS^2 \cup sol^2$.
10. **End**

In Line 2 of the initialization algorithm, the partial solution sol^2 including jobs $N^{'}(1)$ and $N^{'}(2)$ is obtained by solving $\text{MIP}^{(2)}\left(X-relaxed, \sum T_j\right)$ and repairing the obtained solution, if required. we gain an upper bound for the TCE of the current partial

problem, shown as *UB*. Moreover, $\text{MIP}^{(2)}(X-relaxed, TCE)$ alludes to the $\text{MIP}^{(2)}$ in which variables X_{ijv} are relaxed, and minimization of the *TCE* is the single objective function of the model. The obtained solution by $\text{MIP}^{(2)}(X-relaxed, TCE)$ results in a lower bound (*LB*) for the *TCE* of the current partial problem after repair, if required. In Lines 6 to 10 of the initialization algorithm, a set of new non-dominated partial solutions, named NPS^2, are generated using the $\varepsilon-$constraint method.

It is worth mentioning that the $\varepsilon-$constraint method was first introduced by Haime in (1971). The idea behind this method is that solving a multi-objective mathematical model can be converted to a single objective model by making one of the objective functions as the main one and transform the others to constraints by setting a threshold for each one.

3.3. The main body of the MODBH algorithm

The MODBH algorithm takes NPS^2 as input and sequentially finds a set of non-dominated partial solutions in each stage $k = 3, \cdots, n$. The pseudo-code of this algorithm is shown by Algorithm 2. In each iteration of Line 3 of the MODBH algorithm, the insertion algorithm $InsA(sol^k, N^{'}(k+1))$ is applied, taking partial sol^k, including jobs $N^{'}(1)$ to $N^{'}(k)$, and job $N^{'}(k+1)$ as input and returns a set of new non-dominated partial solutions (NPS^{k+1}). If the size of set NPS^{k+1} becomes greater than pop_{max}, we apply the Elite-Pop procedure, described by Algorithm 4, to select a set of elite solutions having the maximum distance in terms of both the total tardiness and the total carbon emissions.

Algorithm 2: The MODBH algorithm

1. Take NPS^2 as input, let $k = 2$ and let $NPS^k = \emptyset$ for $k = 3, \cdots, n$.
2. **For** each $sol^k \in NPS^k$ **do**
3. Apply $InsA(sol^k, N^{'}(k+1))$.
4. **End**
5. **If** $|NPS^{k+1}| > pop_{max}$ **then** $NPS^{k+1} \leftarrow ElitePop(NPS^{k+1})$
6. $k \leftarrow k+1$
7. **If** $k < n$ **then** go to Line 2.
8. **Stop**.

3.4. The insertion algorithm

The pseudo-code of the insertion algorithm, named *InsA*, is presented in Algorithm 3. It takes a partial solution sol^k and job $N^{'}(k+1)$ as input and updates the set of non-dominated partial solutions NPS^{k+1}. Similar to the initialization algorithm, *InsA* first creates two extreme solutions originated from sol^k, and then constructs lower and upper bounds to the *TCE*. Subsequently, in Lines 10 to 16, new solutions are created based on sol^k and the ε-constraint method, and they are repaired if needed. Next, these solutions are added to set NPS^{k+1} if they are not dominated by solutions of NPS^{k+1}.

Algorithm 3: The insertion algorithm (InsA(sol^k, job $\text{N}^{'}(k+1)$))

1. Take sol^k and job $N^{'}(k+1)$ as input.
2. $sol^{k+1} \leftarrow \text{MIP}^{(k+1)}\left(Z^{sol^k} - fixed, X - relaxed, \sum T_j\right)$.
3. $sol^{k+1} \leftarrow$ Repair (sol^{k+1})
4. Let $UB = TCE(sol^{k+1})$.
5. **If** sol^{k+1} is not dominated by solutions of NPS^{k+1}, **then** let $NPS^{k+1} \leftarrow NPS^{k+1} \cup sol^{k+1}$.
6. $sol^{k+1} \leftarrow \text{MIP}^{(k+1)}\left(Z^{sol^k} - fixed, X - relaxed, TCE\right)$.
7. $sol^{k+1} \leftarrow$ Repair (sol^{k+1})
8. Let $LB = TCE(sol^{k+1})$.
9. **If** sol^{k+1} is not dominated by solutions of NPS^{k+1}, **then** let $NPS^{k+1} \leftarrow NPS^{k+1} \cup sol^{k+1}$
10. **For** $i = 1$ **to** λ **do**
11.
12. $sol^{k+1} \leftarrow \text{MIP}^{(k+1)}\left(Z^{sol^k} - fixed, X - relaxed, \sum T_j, TCE \le \varepsilon\right)$

(continued on next column)

(continued)

13. **If** sol^{k+1} is not dominated by solutions of NPS^{k+1}, **then** let $NPS^{k+1} \leftarrow NPS^{k+1} \cup sol^{k+1}$.
14. $sol^{k+1} \leftarrow$ Repair (sol^{k+1})
15. $NPS^{k+1} \leftarrow NPS^{k+1} \cup sol^{k+1}$
16. **End**

3.5. The Elite-Pop algorithm

In some iterations of the MODBH algorithm, it is highly likely that the size of the non-dominated partial solution set violates the threshold value pop_{max}. To prevent this, we deploy the Elite-Pop Algorithm, shown as Algorithm 4, working based upon the crowding distance assignment procedure, developed by Deb et al. (2002). In this procedure, we first build an ordered set of objective function value pairs for non-dominated partial solutions, denoted by *G*. This set is ordered based on non-decreasing values of the total tardiness where $\left(\left(\sum T_j\right)_{(1)}, TCE_{(1)}\right)$ indicates the objective function values of the solution with the minimum total tardiness. To select a uniformly dispersed set of non-dominated partial solutions, we define two distance parameters $dist'_h = |\left(\left(\sum T_j\right)_{(h+1)} - \left(\sum T_j\right)_{(h-1)}\right)|/\left(\left(\sum T_j\right)^{max}\right.$ and $dist''_h = |(TCE_{(h+1)} - TCE_{(h-1)})|/\left(TCE^{max} - TCE^{min}\right)$ for $h = 2$ to $(|NPS| - 1)$. Let the united distance $udist_h = dist'_h + dist''_h; h = 2, \cdots, |NPS| - 1$ and $udist_1 = udist_{|NPS|} = +\infty$. Now, we keep pop_{max} solutions of NPS^{k+1} having the largest values of the united distance and discard others.

Algorithm 4: The Elite-Pop algorithm

1. Take NPS^{k+1} as input.
2. Let $G = \left\{\left(\left(\sum T_j\right)_{(1)}, TCE_{(1)}\right), \cdots, \left(\left(\sum T_j\right)_{(|NPS|)}, TCE_{(|NPS|)}\right)\right\}$
3. $udist_1 \leftarrow +\infty$, $udist_{|NPS|} \leftarrow +\infty$
4. **For** $h = 2$ to $(|NPS| - 1)$ **do**
5. $dist'_h \leftarrow |\left(\left(\sum T_j\right)_{(h+1)} - \left(\sum T_j\right)_{(h-1)}\right)|/\left(\left(\sum T_j\right)^{max} - \left(\sum T_j\right)^{min}\right)$
6. $dist''_h \leftarrow |(TCE_{(h+1)} - TCE_{(h-1)})|/\left(TCE^{max} - TCE^{min}\right)$
7. $udist_h \leftarrow dist'_h + dist''_h$
8. **End**
9. Keep pop_{max} solutions with the largest values of $udist_h$ and remove others from NPS^{k+1}.

4. A multi-objective variable neighborhood search algorithm

The variable neighborhood search (VNS) algorithm is a meta-heuristic that deploys a systematic approach for searching a variable set of neighborhoods to find the best solution in them. By changing the size and structure of neighborhoods, the VNS algorithm tries to escape from being trapped in local optima.

The VNS algorithm was first proposed by Mladenović and Hansen, (1997), and it has three main elements including shaking procedure, improvement procedure and neighborhood change step (Hansen et al. (2017)). There are many variants of the VNS algorithm based on different types of these main elements. In this paper, we propose a multi-objective framework of the VNS algorithm developed by Duarte et al. (2015).

4.1. Solution representation

The solution representation of the GPFSSP contains two parts. The first part is a sequence of jobs and the second one is the processing speeds of each job on each machine. We represent a solution based on a multi-chromosome design used by Yüksel et al. (2020). Let us consider an example of the GPFSSP including 4 jobs, 3 machines and 3 relative processing speeds. Table 1 shows the solution representation of the example in which the first row shows the sequence of jobs and the subsequent rows represent the level of processing speeds of each job on each machine.

Table 1
The example of solution representation.

Sequence	4	1	3	2
Machine 1	1	3	3	2
Machine 2	3	2	1	1
Machine 3	1	1	3	2

4.2. The general sketch of the MOVNS algorithm

Algorithm 5 shows the general sketch of our proposed MOVNS algorithm. In addition to pop_{max}, described in the MODBH algorithm, the MOVNS algorithm takes two more parameters α_{max} and T_{max} as input where α_{max} indicates the maximum number of moves in the shaking procedure and T_{max} is a given time limit. Furthermore, *NCS* indicates a set of non-dominated complete solutions.

Algorithm 5: The MOVNS algorithm

1. Take $\alpha_{max} = \frac{n}{2}$, T_{max} and pop_{max} as input.
2. *NCS*←generate pop_{max} random solutions
3. **While** current–time < T_{max} **do**
4. α←1
5. **While** $\alpha \leq \alpha_{max}$ **do**
6. $NCS^{'}$←MO-Shake (*NCS*,α)
7. $NCS^{''}$←MO-VND ($NCS^{'}$)
8. *NCS*←MO-Neighborhood change ($NCS, NCS^{''}, \alpha$)
9. **If** $|NCS| > maxpop$, **then***NCS*←*Elite* −*Pop*(*NCS*)
10. **End**
11. **End**

At the beginning of this algorithm, pop_{max} random solutions are generated which constitute set *NCS*. Next, a *while-loop* is repeated until the given time limit T_{max} is met. At the outset of this loop, the domain of the neighborhood search is set to $\alpha = 1$. Following, a nested *while-loop* starts and continues until the maximum domain of neighborhood search (α_{max}) is achieved. In each iteration of the nested *while-loop*, solutions of set *NCS* are transformed into new solutions, shown by *NCS*', through the *shaking* procedure, called MO-Shake, and are improved using an *improvement* procedure, called MO-VND, described in Section 4.4. Finally, the neighborhood structure will change using the MO-Neighborhood change procedure. Similar to the MODBH algorithm, we manage the size of generated solutions using *ElitePop* algorithm, described in Section 3.5.

4.3. The shaking procedure

The MOVNS algorithm starts to explore variants of neighborhoods iteratively to improve the solution set. The MO-Shake procedure is used to avoid local optima and create solution diversification. Our developed MO-Shake procedure is shown in Algorithm 6.

Algorithm 6: The MO-Shake procedure

1. Take *NCS* and α as input
2. $NCS^{'}$←∅
3. **For** each $sol \in NS$ **do**
4. $sol^{'}$←α −*reverse* (sol)
5. $sol^{''}$←α −*block* ($sol^{'}$)
6. $NCS^{'}$←$NCS^{'} \cup \{sol^{''}\}$
7. **End**
8. **Return** $NCS^{'}$

Two neighborhood structures, named α −*reverse* and α −*block*, are introduced for the shaking procedure. α −*reverse* extracts randomly α pairs of jobs from the solution representation table, and exchanges the position of the selected jobs in the table without exchanging the corresponding processing speeds. For instance, consider the example of Table 1 and let $\alpha = 1$. A random pair of jobs are jobs 3 and 4 where the resultant solution of 1 −*reverse* is shown in Table 2 in which the affected cells are bold.

Table 2
Illustration of α −*reverse* move.

Sequence	**3**	**1**	**4**	**2**
Machine 1	1	3	3	2
Machine 2	3	2	1	1
Machine 3	1	1	3	2

Moreover, α −*block* extracts α consecutive jobs of the solution representation table and inserts them in a new random position. Given $\alpha = 2$, the selected jobs are jobs 1 and 3 and the new positions are at the end of the table; the resultant solution is shown in Table 3.

Note that in α −*reverse* and α −*block* we just change the position of jobs and the columns of processing speeds are not changed; hence this leads to different processing speeds for the shifted jobs.

4.4. The improvement procedure

We chose to use a multi-objective variable neighborhood descent (MO-VND) procedure, described in Algorithm 7, as the improvement procedure. In the MO-VND procedure, we try to improve the values of the two objective functions one by one in such a way that the total tardiness is considered first. Thus, the MO-VND includes two similar parts. In this section, we use f_1 to refer to the value of the total tardiness and f_2 to refer to the value of the total carbon emissions.

Three moves, called *increase* , *decrease* and *swap*, are deployed in this procedure. All of these moves are based on the well-known *first improvement* strategy. An *increase*(A*decrease*) move includes the selection of an operation and raising (decreasing) the speed of processing this operation if it is not at the maximum (minimum) speed level. The *increase*(*decrease*) operator, used in Line 7 (23) of the MO-VND procedure, applies *increase*(*decrease*) moves to all operations. The *swap* move considers a pair of columns of the solution representation table and exchanges their positions in the table. This move only changes the sequence and has no effect on processing speeds.

The MO-VND procedure takes a solution set $NCS^{'}$ and parameter $\beta_{max} = 2$ as input where $\beta \leq \beta_{max}$ and different values of β corresponds to different moves including *increase*, *decrease* and *swap*. Lines 2 to 18 constitute the first part of the MO-VND procedure in which a set of non-dominated solution NCS_1 is generated. In this part, the focus is on the improvement of solutions in terms of the total tardiness using the *increase* and *swap* operators, functioning as a solution intensification strategy. It should be mentioned that the *Update* operator, used in Line 14, selects non-dominated solutions from the union of set NCS_1 and solution *sol*.

Algorithm 7: The MO-VND Procedure

1. Takes $NCS^{'}$ and $\beta_{max} = 2$ as input.
2. β←1, NCS_1←∅
3. **For** each $sol \in NCS^{'}$ **do**
4. NCS_1←{sol}
5. **While** $\beta \leq \beta_{max}$ **do**
6. **If** $\beta = 1$ **then**
7. $sol^{'}$←*increase*(sol)
8. **Else**
9. $sol^{'}$←*swap*(sol)
10. **End**
11. **If** $f_1(sol^{'}) < f_1(sol)$ **then**
12. sol←$sol^{'}$
13. β←1

(*continued on next page*)

Table 3
Illustration of α −*opt* move.

Sequence	**4**	**2**	**1**	**3**
Machine 1	1	3	3	2
Machine 2	3	2	1	1
Machine 3	1	1	3	2

(*continued*)

```
                    Update(NCS_1, sol)
                  Else β←β + 1
                  End
              End
          End
          For each sol ∈ NCS_1 do
          NCS_2←{sol}, β←1
              While β ≤ β_max do
                  If β = 1 then
                      sol'←decrease(sol)
                  Else
                      sol'←swap(sol)
                  End
                  If f_2(sol') < f_2(sol) then
                      sol←sol'
                      β←1
                      Update(NCS_2, sol)
                  End
              End
          End
          Return NCS_2
```

The second part of the MO-VND procedure starts by taking solution set NCS_1 as input and functions like the first part, but it tries to improve them in terms of the total carbon emissions. Finally, the MO-VND procedure returns the set of non-dominated solutions NCS_2 as output.

4.5. The neighborhood change step

As the third main element of the MOVNS algorithm, we describe the procedure of changing the neighborhood in this section. Algorithm 8 presents the MO-Neighborhood change procedure to determine the domain of neighborhood search. Considering *NCS* and *NCS*' as the initial and improved solution sets in this procedure respectively, we use the basic MO- Improvement procedure to determine whether any improvement has been achieved or not. If it returns *True*, implying an improvement has been realized, the algorithm would start from the first neighborhood structure and set *NCS* will be updated i.e. a non-dominated solution set is selected from the union of sets *NCS* and *NCS*'. Otherwise, we would broaden the search domain to the next neighborhood structure.

Algorithm 8: The MO-Neighborhood change procedure

```
Take NS, NS' and α as input.
If MO-Improvement (NCS, NCS') = True then
        α←1
        NCS←Update(NCS, NCS')
Else
        α←α + 1
End
```

5. Computational experiments

In this section, we discuss computational experiments to evaluate the performance of the developed solution approaches. To validate the efficiency of our methods developed for solving the GPFSSP in Sections 3 and 4, we need to compare them with a validated solution approach. To do so, we choose the modified multi-objective iterated greedy (MMOIG) algorithm developed by Ding et al. (2016), which is mentioned as the algorithm with the best performance among the proposed algorithms in that research. All the algorithms were implemented on a PC with 32 GB RAM, Intel® Core™ i7 4.00 GHz processor and were coded in C++ using Visual Studio 2019. For the MIP models used in the implementation of the MODBH, we made use of the ILOG CPLEX 12.6.3 to solve the problems.

5.1. Experimental setup

To evaluate and compare the performance of the developed solution approaches, we generate a random test set. To do so, we consider $n = 20, 40, 60, 80$ and 100, $m = 4, 8$ and 16 and, we generate 10 random instances for each combination of n and m, resulting in 150 test instances. Furthermore, five levels for machine speeds are considered which are identical in all instances. All parameters of the test set and their references are summarized in Table 4. It should be noted that all test instances are available on *http://m_ranjbar.profcms.um.ac.ir/index.php/publications/submitted-articles*.

5.2. The performance criteria

To evaluate the performance of the proposed multi-objective algorithms we introduce the following criteria.

a. The ratio of the Pareto-optimal solutions (R_A) (Yüksel, Taşgetiren, Kandiller, & Gao, 2020): this metric, shown by equation (20), calculates the ratio of the number of in common solutions of a solution set A with a Pareto-optimal solution set P divided by the number of solutions of P. As we do not have the Pareto-optimal solution set for the problem at hand, we consider all non-dominated solutions found by the three algorithms as the Pareto-optimal solution set. Note that $R_A \in [0, 1]$ and the higher amount of this metric indicates the higher quality of the solution set A.

$$R_A = \frac{|A \cap P|}{|P|} \tag{20}$$

b. Convergence metric (C_{AB}) (Zitzler, 1999): convergence metric, shown by equation (21), indicates the proportion of the solutions of set B which are dominated by the solutions of set A. We use symbol $sol' \succcurlyeq sol$ to show that solution *sol* is weakly dominated by solution *sol*'. Note that $C_{AB} \in [0, 1]$ and the closer value to 1 implies that the more proportion of set B is dominated by set A. Furthermore, notice that C_{AB} may not be equal to $1 - C_{BA}$, because it is possible that some solutions in the two sets are not dominated by one another.

$$C_{AB} = \frac{\{sol \in B | \exists sol' \in A : sol' \succcurlyeq sol\}}{|B|} \tag{21}$$

c. Inverted generational distance (IGD_A) (Coello and Cortes (2005)): the IGD_A metric, shown by equation (22), reflects two measures simultaneously. It shows the approximation of solution set A to the Pareto-optimal solution set P. Moreover, it indicates the distribution of set A through calculating the average distance of solutions in P to their closest solution in A. Similar to R_A, we use the combined non-dominated solution set found by the three algorithms as the Pareto-optimal solution set. Note that D_i in equation (22) indicates the Euclidian distance between solution i in P from its nearest solution in A. The lower value of IGD_A shows better quality and implies set A is more approximate and distributed.

Table 4
Summary of the parameters of the test set.

Parameter	Parameter Level	Reference
number of jobs (n)	20,40,60,80,100	Ding et al. (2016)
number of machines (m)	4,8,16	Ding et al. (2016)
relative machine processing speeds (S)	(1, 1.3, 1.55, 1.75, 2.10)	Ding et al. (2016)
processing times (p_{ij})	Uniform distribution [5,50] minutes	Ding et al. (2016)
power of machines (π_{iv})	In interval [4,16] KW	Ding et al. (2016)
power of machines in idle mode (π'_i)	1 KW	Ding et al. (2016)
conversion coefficient to CO_2 (γ)	0.785	Foumani and Smith-Miles (2019)
due dates (d_j)	$(1 + 3u)\sum_{i=1}^{m} p_{ij}$:$u \in [0, 1]$	Hasija and Rajendran, (2004)

Table 5
Adjusting the parameter δ in the MMOIG algorithm.

n	m	δ		
		3	6	9
20	4	0.02	0.06	0.03
20	8	0.02	0.03	0.09
20	16	0.02	0.06	0.04
40	4	0.03	0.1	0.05
40	8	0.04	0.02	0.04
40	16	0.02	0.03	0.02
60	4	0.02	0.04	0.07
60	8	0.04	0.02	0.02
60	16	0.03	0.08	0.02
80	4	0.04	0.06	0.04
80	8	0.05	0.04	0.01
80	16	0.03	0.02	0.02
100	4	0.02	0.03	0.04
100	8	0.04	0.03	0.02
100	16	0.04	0.02	0.02
Average		0.03	0.04	0.04

Table 6
CPU runtimes (seconds) of the MODBH algorithm.

n	m	Average CPU runtimes	Standard deviation of CPU runtimes
20	4	12.15	3.29
20	8	11.92	3.94
20	16	22.87	6.83
40	4	346.2	38.2
40	8	426.10	56.00
40	16	674.55	61.91
60	4	1921.49	293.31
60	8	2857.68	480.19
60	16	4948.05	801.26
80	4	2403.81	444.90
80	8	4571.22	499.09
80	16	7143.30	927.62
100	4	2863.82	539.51
100	8	3234.32	627.94
100	16	13759.14	1441.47

$$IGD_A = \frac{\sqrt{\sum_{i=1}^{|P|} D_i^2}}{|P|} \tag{22}$$

d. Distribution spacing (DS_A) (Tan et al. (2006)): this metrics, shown by equation (23), measures the variation of the distance among the solutions of set A and indicates how solutions of set A are evenly distributed. Note that $D^{'}_i$ in equation (23) is the Euclidian distance between solution i of set A and its nearest solution and $\overline{D} = \sum_{i=1}^{|A|} D^{'}_i / |A|$.

$$DS_A = \sqrt{\frac{1}{|A|} \frac{\sum_{i=1}^{|A|} (D^{'}_i - \overline{D})^2}{\overline{D}}} \tag{23}$$

More information about performance measures in multi-objective optimization is provided by Audet et al. (2020).

5.3. Parameters setting

To improve the performance of the developed solution approaches we need to adjust the parameter used in the algorithms. In the MODBH algorithm, the only parameter which needs to be adjusted is λ, reflecting the number of generated solutions between the founded upper bound and lower bound in the algorithm. High values of λ lead to the Pareto fronts with a higher number of points, but they are very time-consuming. Using fine tuning, we set $\lambda = 3$ to make a reasonable tradeoff between the number of points and CPU runtimes.

Using a fine-tuning approach, in the MOVNS algorithm we set $\alpha_{max} = \frac{n}{2}$and $\beta_{max} = 2$, respectively. For the MMOIG algorithm there are two parameters that must be adjusted; the probability of changing speeds in the destruction phase (ρ) and the number of jobs to be removed from the complete sequence in the destruction phase, denoted as δ in this paper. Similar to the original paper, we found that $\rho = 0.4$ is the most suitable value. To determine δ, we choose three levels 3,6 and 9 and the *IGD* metric leading to Table 5 where one instance has been considered in each cell. According to Table 5, we set $\delta = 3$ resulting a smaller average value of the *IGD*.

Moreover, in the MOVNS and the MMOIG, T_{max} is adjusted as will be discussed later in Section 5.4.1 and we consider $pop_{max} = 25$ in all related algorithms.

5.4. Comparative computational experiments

The MMOIG algorithm was deployed by Ding et al. (2016) to solve the GPFSSP with the total energy consumption (*TEC*) and the makespan (C_{max}) as the objective functions of the problem. In this section, we consider two parts where in the first one comparative computational results on $\sum T_j$ and *TCE* are reported whereas in the second one the comparative computational results on C_{max} and *TCE* are presented. It should be noted that *TCE* and *TEC* are very much alike. Consequently, we only need to replace C_{max} with $\sum T_j$ or vice versa in all three algorithms for a fair comparison.

5.4.1. Computational results of $\sum T_j$ and TCE

In this section, we consider $\sum T_j$ and *TCE* as the two objective functions in all three algorithms, i.e. the MODBH, the MOVNS and the MMOIG. Table 6 presents the average and standard deviation of CPU runtimes obtained by the MODBH algorithm. As can be seen and expected, larger instances are more time-consuming. Note that for a fair comparison among algorithms, we run the MOVNS algorithm and the MMOIG algorithms by considering identical CPU runtimes, obtained from the implementation of the MODBH algorithm.

Table 7 shows the comparison of the algorithms based on average values of *R*, *IGD* and *DS* metrics where in each row the best values are bold. According to Table 7, the MODBH algorithm shows the best performance based on the *IGD* and *R* measures, indicating the solution sets obtained by the MODBH algorithm are closer to aggregate non-dominated solution sets. This also signifies the solution obtained by the MODBH algorithm are better distributed throughout the solution space, and they could offer a better tradeoff between two objective functions.

In our view, the main reason that justifies the good performance of the MODBH is in the insertion algorithm, where we decide about the position of the inserted job and the corresponding machine speed by solving a relaxed MIP model. Ding et al. (2016) used an extended NEH-Insertion, which inserts a job, say *j*, to all possible positions and calls an ad-hoc energy-saving method that slows down the operations of job *j* without changing the total tardiness (or the makespan). The merit of our approach is that the processing scaling procedure is done in a holistic way that looks for the optimal machine speed for all scheduled operations rather than those of job *j*, which could bring certain benefit.

Furthermore, it can be seen from Table 7 that the MOVNS algorithm has the best performance based on *DS* metric. Overall, it can be concluded that the MODBH and MOVNS algorithms are superior to the

Table 7
Comparison of the developed algorithms using *R*, *IGD* and *DS* metrics.

n	*m*	MODBH			MOVNS			MMOIG		
		R	*IGD*	*DS*	*R*	*IGD*	*DS*	*R*	*IGD*	*DS*
20	4	**0.58**	**0.50**	0.30	0.34	0.52	**0.22**	0.11	3.27	0.24
20	8	**0.45**	**0.93**	0.25	0.31	1.78	**0.17**	0.12	3.28	0.20
20	16	**0.37**	**1.10**	0.18	0.34	1.92	**0.14**	0.24	2.52	0.40
40	4	**0.53**	**0.41**	0.29	0.39	2.45	**0.11**	0.05	4.71	0.15
40	8	**0.46**	**0.63**	0.33	0.42	1.67	0.26	0.22	3.84	**0.16**
40	16	**0.37**	**1.39**	0.34	**0.37**	2.31	**0.18**	0.30	3.26	0.21
60	4	**0.61**	**0.31**	0.21	0.33	3.21	0.14	0.07	5.14	**0.10**
60	8	**0.61**	**0.34**	0.22	0.31	1.98	**0.18**	0.08	5.11	0.19
60	16	**0.36**	**0.90**	0.36	0.31	2.06	**0.10**	0.29	4.19	0.20
80	4	**0.69**	**0.36**	0.28	0.35	3.32	**0.13**	0.07	4.14	0.22
80	8	**0.68**	**0.31**	0.25	0.24	1.84	**0.09**	0.03	4.78	0.19
80	16	**0.42**	**1.31**	0.20	0.29	1.74	**0.12**	0.21	4.81	0.13
100	4	**0.88**	**0.18**	0.28	0.37	4.87	**0.07**	0.03	3.06	0.11
100	8	**0.72**	**0.35**	0.38	0.09	1.50	0.24	0.10	3.64	**0.10**
100	16	**0.60**	**0.85**	0.37	0.18	1.48	0.16	0.07	3.81	**0.11**

Table 8
the paired comparisons of the algorithms based on the convergence metric.

n	*m*	$C_{MODBH,MOVNS}$	$C_{MOVNS,MODBH}$	$C_{MODBH,MMOIG}$	$C_{MMOIG,MODBH}$	$C_{MOVNS,MMOIG}$	$C_{MMOIG,MOVNS}$
20	4	**0.55**	0.24	**0.74**	0.00	**0.32**	0.10
20	8	**0.42**	0.08	**0.72**	0.06	**0.17**	0.09
20	16	**0.42**	0.17	**0.48**	0.18	0.00	**0.13**
40	4	**0.87**	0.00	**0.46**	0.09	**0.02**	0.00
40	8	**0.35**	0.24	**0.63**	0.00	0.00	**0.01**
40	16	**0.47**	0.02	**0.74**	0.10	0.10	**0.13**
60	4	**0.60**	0.03	**0.88**	0.00	**0.26**	0.07
60	8	**0.61**	0.04	**0.88**	0.09	**0.05**	0.00
60	16	**0.40**	0.07	**0.34**	0.03	**0.01**	0.00
80	4	**0.73**	0.00	**0.90**	0.00	0.00	0.00
80	8	**0.64**	0.00	**0.94**	0.00	0.00	**0.07**
80	16	**0.75**	0.02	**0.56**	0.00	**0.06**	0.02
100	4	**0.90**	0.00	**1.00**	0.00	**0.07**	0.06
100	8	**0.80**	0.01	**1.00**	0.00	0.00	0.00
100	16	**0.81**	0.00	**0.99**	0.00	**0.07**	0.01

MMOIG algorithm. Moreover, the MODBH algorithm has better performance than the MOVNS algorithm in terms of the quality of obtained solutions; however, obtained solutions by the MOVNS algorithm are more evenly distributed across the obtained fronts. It should be noticed that the obtained fronts by the MOVNS are narrower than the MODBH's.

Table 8 represents the convergence metric for each pair of the algorithms. According to Table 8, it can be generally concluded that the MODBH algorithm dominates the others. Furthermore, the MOVNS and the MMOIG algorithms have almost the same dominancy over each other.

For better illustration of comparative results of the developed algorithms, we select three instances, i.e. 20 × 4, 60 × 8 and 100 × 16, and

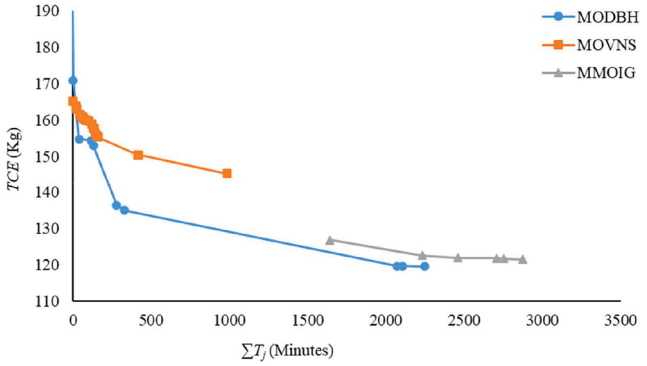

Fig. 1. The Pareto fronts of the developed algorithms for 20 × 4 instance.

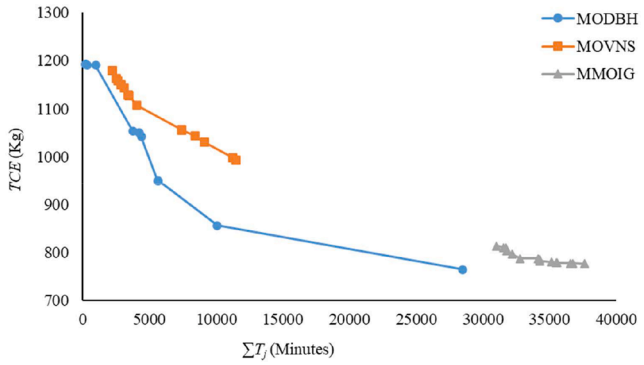

Fig. 2. The Pareto fronts of the developed algorithms for 60 × 8 instance.

we plots the obtained Pareto fronts using Figs. 1, 2 and 3. As can be seen from these figures, the MODBH algorithm generates wide Pareto fronts, and it is dominant to two other algorithms in most cases. Considering the MOVNS and the MMOIG algorithms, they are not much dominant over each other, like what was seen in Table 8. The MOVNS algorithm has better performance considering the total tardiness whereas the MMOIG algorithm generates fronts with better *TCE* objective function values.

To summarize this section, it can be concluded that the strength of the MODBH algorithm is generating wide and high-quality Pareto fronts, which is due to the ε −constraint method and proper lower and upper bounds that are deployed in the algorithm. Furthermore, the MIP contributes to high-quality solutions by solving the sub-problems in each

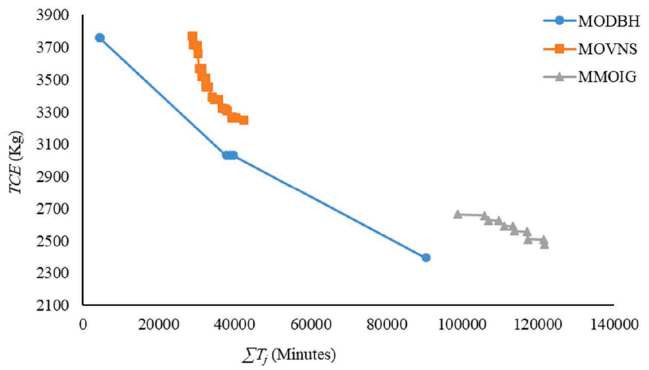

Fig. 3. The Pareto fronts of the developed algorithms for 120 × 16 instance.

iteration. Moreover, to achieve more uniformly distributed fronts by this algorithm, we can increase λ to discover more non-dominated solutions when the ε −constraint method is used in each iteration, but this will increase the CPU runtime.

5.4.2. Computational results of C_{max} and TCE

Since the MMOIG algorithm was previously developed for C_{max} by Ding et al. (2016), we need to validate the superiority of our developed algorithms by using C_{max} in lieu of $\sum T_j$. Thus, in this section, we consider C_{max} and *TCE* as the two objective functions in all three algorithms and the results are discussed below.

According to Table 9, the MODBH and MOVNS algorithms are again superior over the MMOIG algorithm considering *R* and *IGD* metrics, implying that our developed methods generate high-quality and distributed fronts. However, the MMOIG algorithm has achieved better results in terms of *DS* metric. Furthermore, Table 10 indicates that the dominancy of the MODBH algorithm to the MOVNS and MMOIG algorithm based on the convergence metric.

5.5. Sensitivity analyses

In this section, some sensitivity analyses are investigated in which we consider the total tardiness and the total carbon emissions as the objective functions of the GPFSP.

5.5.1. The impact of the initial job ordering

In this section, we investigate the comparative efficiency of job ranking measure $o_j, j \in N$ compared to some usual ranking measures, presented in Table 11. To do so, we choose randomly one instance from each combination of values of *n* and *m*, and we implement the MODBH algorithm using four ranking factors, i.e. o_j, d_j, $p_j = \sum_{i=1}^{m} p_{ij}$, and a random list. The results, presented using the *IGD* metric, indicate that using o_j leads to better solution sets. It should be noted that for calculating the *IGD* metric, we consider the aggregate non-dominated solution set obtained from these implementations as the Pareto optimal solutions set.

5.5.2. The robustness of the MOVNS algorithm

In this section, we investigate the robustness of the MOVNS algorithm which staunchly depends on the random moves, deployed in the shaking procedure. To this end, we choose three random instances and run the MOVNS algorithm 10 times to solve these instances. We calculate the *IGD* metric for them by considering the aggregate non-dominated solution set obtained from the three algorithms as the Pareto optimal solution set. According to Table 12, the low variance of the metrics shows that the results of MOVNS are robust.

5.5.3. The optimality gap of the MODBH algorithm

In this section, we investigate the optimality gap between the MODBH algorithm, showing the best performance among all examined algorithms in this article, and the Pareto front obtained by means of the developed MIP model in Section 2 and the ε −constraint approach. To make such a comparison, we generate 10 random small instances with $n = 10$, $m = 2$ and $S = \{1, 1.55, 2.1\}$. Moreover, we assume that $\lambda = 10$ and $pop_{max} = 25$, and we show comparative results based on the *IGD* metric which is based on the distance of solution of a front from the Pareto-optimal front and could be deployed to measure the gap between the fronts. It should be noted that, in this section, the union of non-dominated solutions by both the MIP model and the MODBH are considered as the Pareto-optimal front.

To solve the MIP model with ε −constraint method, we calculate the upper and lower bounds of the *TCE* using the approach mentioned in Lines 2 and 4 of Algorithm 1. We also implement Lines 6 to 10 of Algorithm 1 without relaxing *X* variables. Note that there is not any job insertion procedure for MIP and all jobs should be scheduled simultaneously.

According to Table 13, the low difference between the *IGD* metric of the MIP model and the MODBH shows that the front generated by these approaches are very close to one another. This indicates that the MODBH method has a competitive performance in comparison with the MIP model.

Considering instance 8 of Table 13, Fig. 4 illustrates the corresponding Pareto fronts obtained by the MODBH algorithm and the MIP model for this instance. This figure signifies the high-quality and wide front generated by the MODBH algorithm.

Table 9
Comparison of the developed algorithms based on *R*, *IGD* and *DS* metrics.

n	*m*	MODBH			MOVNS			MMOIG		
		R	*IGD*	*DS*	*R*	*IGD*	*DS*	*R*	*IGD*	*DS*
20	4	0.41	0.97	0.22	**0.52**	**087**	0.15	0.07	2.98	**0.13**
20	8	0.38	**0.69**	0.34	**0.47**	1.21	0.17	0.15	2.51	**0.15**
20	16	0.42	**0.55**	0.27	**0.46**	1.15	0.19	0.12	2.81	**0.17**
40	4	**0.44**	**0.83**	0.37	0.31	1.58	**0.15**	0.25	2.62	0.19
40	8	**0.61**	**0.47**	0.35	0.33	1.98	**0.14**	0.06	2.98	0.17
40	16	**0.53**	**0.45**	0.32	0.46	1.28	0.14	0.01	3.07	**0.11**
60	4	**0.44**	**0.74**	0.67	0.37	1.96	**0.11**	0.19	2.84	0.12
60	8	**0.68**	**0.85**	0.44	0.28	1.59	**0.12**	0.04	3.37	0.14
60	16	**0.40**	**0.59**	0.27	0.36	1.21	**0.09**	0.24	3.09	0.11
80	4	0.29	**0.61**	0.41	**0.47**	1.39	0.16	0.24	4.12	**0.12**
80	8	0.27	**0.86**	0.69	**0.53**	1.97	0.11	0.20	3.73	**0.09**
80	16	0.31	**1.08**	0.52	**0.57**	2.11	0.15	0.12	3.71	**0.10**
100	4	**0.43**	**1.26**	0.66	0.37	3.01	0.11	0.20	3.69	**0.08**
100	8	0.34	**0.84**	0.41	**0.42**	3.17	0.10	0.24	3.59	**0.09**
100	16	**0.51**	**0.71**	0.39	0.34	2.98	0.12	0.15	3.87	**0.11**

Table 10
The paired comparisons of the algorithms based on the convergence metric.

n	m	$C_{MODBH,MOVNS}$	$C_{MOVNS,MODBH}$	$C_{MODBH,MMOIG}$	$C_{MMOIG,MODBH}$	$C_{MOVNS,MMOIG}$	$C_{MMOIG,MONS}$
20	4	0.31	**0.39**	**0.81**	0.00	**0.84**	0.00
20	8	**0.42**	0.37	**0.89**	0.00	**0.29**	0.00
20	16	**0.37**	0.29	**0.91**	0.00	**0.12**	0.00
40	4	**0.71**	0.25	**0.68**	0.04	**0.10**	0.01
40	8	**0.65**	0.21	**0.94**	0.00	**0.05**	0.00
40	16	**0.42**	0.21	**0.98**	0.00	0.00	0.00
60	4	**0.57**	0.29	**0.51**	0.01	**0.17**	0.03
60	8	**0.64**	0.19	**0.81**	0.00	0.00	0.00
60	16	**0.43**	0.27	**0.77**	0.00	0.00	0.00
80	4	**0.51**	0.31	**0.59**	0.02	**0.19**	0.03
80	8	**0.36**	0.29	**0.49**	0.00	0.00	0.00
80	16	**0.50**	0.27	**0.77**	0.00	0.00	0.00
100	4	**0.71**	0.09	**0.61**	0.09	**0.01**	0.00
100	8	**0.97**	0.00	**0.95**	0.01	0.00	0.00
100	16	**0.94**	0.01	**0.89**	0.01	0.00	0.00

Table 11
The contribution of o_j to the performance of the MODBH algorithm.

n	m	IGD_{o_j}	IGD_{d_j}	IGD_{p_j}	IGD_{random}
20	4	**0.31**	0.38	0.40	0.45
20	8	**0.34**	0.41	0.35	0.39
20	16	**0.42**	0.49	0.41	0.68
40	4	**0.35**	0.37	0.41	0.53
40	8	**0.41**	0.58	0.52	0.78
40	16	**0.65**	0.74	0.69	0.72
60	4	**0.59**	0.67	0.69	0.71
60	8	**0.49**	0.54	0.51	0.69
60	16	**0.73**	0.65	0.83	0.91
80	4	0.45	**0.41**	0.48	0.49
80	8	**0.32**	0.45	0.37	0.53
80	16	**0.44**	0.59	0.52	0.61
100	4	**0.63**	0.70	0.79	0.96
100	8	**0.43**	0.45	0.49	0.61
100	16	**0.58**	0.61	0.59	0.82
Average		**0.48**	0.54	0.54	0.69

6. Conclusion

In this paper, we investigate a bi-objective permutation flow shop scheduling problem to minimize the total carbon emissions as well as the total tardiness. We present a MIP model and develop two multi-objective heuristic algorithms dubbed the MODBH and the MOVNS. The former works based on the idea of decomposing the MIP model into a series of job insertion procedures. The latter is a generalization of the well-known VNS algorithm and generates uniformly distributed solutions. We also utilized the MMOIG algorithm, developed by Ding et al. (2016), to validate the computational results of our developed solution approaches. We run all three algorithms for two bi-objective cases, i.e. I) the total tardiness and the total carbon emissions, and II) the makespan and the total carbon emissions.

The experimental results reveal that the MODBH algorithm is superior to the other algorithms in terms of the quality of obtained solution sets. It also offers a better tradeoff between two objective functions considering the distribution of the obtained sets. Moreover, the MOVNS algorithm has acceptable quality, and it shows the best performance based on the measure of the uniformity of obtained solution sets.

As guidelines for future studies, more assumptions like setup times and a finite storage capacity between machines can be considered to make the problem more challenging and practical. Moreover, investigating some accelerating methods for the MODBH algorithm to lessen CPU runtime, particularly for large instances, might be an interesting research topic.

Table 13
The comparative results of the MIP model and the MODBH algorithm.

Instance	IGD_{MIP}	IGD_{MODBH}
1	0.10	0.19
2	0.15	0.21
3	0.27	0.32
4	0.21	0.25
5	0.14	0.19
6	0.39	0.41
7	0.13	0.18
8	0.16	0.19
9	0.20	0.15
10	0.26	0.11
Average	**0.20**	0.22

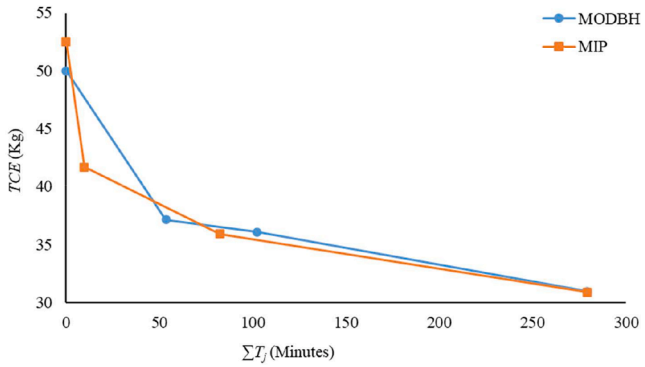

Fig. 4. The Pareto fronts of an instance (10 × 2) obtained by the MIP and the MODBH algorithm.

Table 12
Robustness analysis of the MOVNS algorithm.

n	m	IGD_1	IGD_2	IGD_3	IGD_4	IGD_5	IGD_6	IGD_7	IGD_8	IGD_9	IGD_{10}	variance
20	4	0.41	0.49	0.39	0.45	0.40	0.47	0.42	0.48	0.45	0.39	0.001
60	8	1.87	1.71	1.90	2.07	1.95	1.82	1.94	1.79	1.89	1.83	0.010
100	16	1.57	1.39	1.45	1.41	1.49	1.53	1.60	1.51	1.44	1.37	0.006

CRediT authorship contribution statement

Reza Ghorbani Saber: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft. **Mohammad Ranjbar:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Project administration, Resources, Supervision, Validation, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Amiri, M.F., Behnamian, J., 2020. Multi-objective green flowshop scheduling problem under uncertainty: Estimation of distribution algorithm. J. Cleaner Prod. 251, 119734.

Audet, C., Bigeon, J., Cartier, D., Le Digabel, S., Salomon, L., 2020. Performance indicators in multiobjective optimization. Eur. J. Oper. Res. 292 (2), 397–422.

Birgin, E.G., Ferreira, J.E., Ronconi, D.P., 2020. A filtered beam search method for the m-machine permutation flowshop scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs. Comput. Oper. Res. 114, 104824. https://doi.org/10.1016/j.cor.2019.104824.

Coello, C.A.C., Cortes, N.C., 2005. Solving multiobjective optimization problems using an artificial immune system. Genet. Program Evolvable Mach. 6 (2), 163–190.

Ding, J.-Y., Song, S., Wu, C., 2016. Carbon-efficient scheduling of flow shops by multi-objective optimization. Eur. J. Oper. Res. 248 (3), 758–771.

Duarte, A., Pantrigo, J.J., Pardo, E.G., Mladenovic, N., 2015. Multi-objective variable neighborhood search: an application to combinatorial optimization problems. J. Global Optim. 63 (3), 515–536.

Fang, K., Uhan, N.A., Zhao, F.u., Sutherland, J.W., 2013. Flow shop scheduling with peak power consumption constraints. Ann. Oper. Res. 206 (1), 115–145.

Fernandez-Viagas, V., Framinan, J.M., 2015. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. Comput. Oper. Res. 60, 27–36.

Fernandez-Viagas, V., Molina-Pariente, J.M., Framinan, J.M., 2020. Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling. Eur. J. Oper. Res. 282 (3), 858–872.

Foumani, M., Smith-Miles, K., 2019. The impact of various carbon reduction policies on green flowshop scheduling. Appl. Energy 249, 300–315.

Framinan, J.M., Leisten, R., 2008. Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm. Int. J. Prod. Res. 46 (22), 6479–6498.

Framinan, J.M., Gupta, J.N.D., Leisten, R., 2004. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. J. Oper. Res. Soc. 55 (12), 1243–1255.

Fu, Y., Tian, G., Fathollahi-Fard, A.M., Ahmadi, A., Zhang, C., 2019. Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint. J. Cleaner Prod. 226, 515–525.

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Math. Oper. Res. 1 (2), 117–129.

Haimes, Y., 1971. On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Trans. Syst. Man Cybernet. 1 (3), 296–297.

Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S., 2017. Variable neighborhood search: basics and variants. EURO J. Comput. Opt. 5 (3), 423–454.

Hasija, S., Rajendran, C., 2004. Scheduling in flowshops to minimize total tardiness of jobs. Int. J. Prod. Res. 42 (11), 2289–2301.

Kim, Y.-D., 1993. Heuristics for flowshop scheduling problems minimizing mean tardiness. J. Oper. Res. Soc. 44 (1), 19–28.

Kopanos, G.M., Méndez, C.A., Puigjaner, L., 2010. MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. Eur. J. Oper. Res. 207 (2), 644–655.

Mansouri, S.A., Aktas, E., Besikci, U., 2016. Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. Eur. J. Oper. Res. 248 (3), 772–788.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24 (11), 1097–1100.

Mouzon, G., Yildirim, M.B., 2008. A framework to minimise total energy consumption and total tardiness on a single machine. Int. J. Sustainable Eng. 1 (2), 105–116.

Mouzon, G., Yildirim, M.B., Twomey, J., 2007. Operational methods for minimization of energy consumption of manufacturing equipment. Int. J. Prod. Res. 45 (18-19), 4247–4271.

Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11 (1), 91–95.

Öztop, H., Tasgetiren, M.F., Eliiyi, D.T., Pan, Q.-K., Kandiller, L., 2020. An Energy-Efficient Permutation Flowshop Scheduling Problem. Expert Syst. Appl. 150, 113279. https://doi.org/10.1016/j.eswa.2020.113279.

Pan, Q.-K., Ruiz, R., 2013. A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Comput. Oper. Res. 40 (1), 117–128.

Rajendran, C., Ziegler, H., 2003. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. Eur. J. Oper. Res. 149 (3), 513–522.

Ruiz, R., Maroto, C., 2005. A comprehensive review and evaluation of permutation flowshop heuristics. Eur. J. Oper. Res. 165 (2), 479–494.

Tan, K.C., Goh, C.K., Yang, Y.J., Lee, T.H., 2006. Evolving better population distribution and exploration in evolutionary multi-objective optimization. Eur. J. Oper. Res. 171 (2), 463–495.

Vallada, E., Ruiz, R., 2010. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. Omega 38 (1-2), 57–67.

Vallada, E., Ruiz, R., Minella, G., 2008. Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. Comput. Oper. Res. 35 (4), 1350–1373.

Yenisey, M.M., Yagmahan, B., 2014. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. Omega 45, 119–135.

Yüksel, D., Taşgetiren, M.F., Kandiller, L., Gao, L., 2020. An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption. Comput. Ind. Eng. 145, 106431. https://doi.org/10.1016/j.cie.2020.106431.

Zhang, H., Zhao, F., Fang, K., Sutherland, J.W., 2014. Energy-conscious flow shop scheduling under time-of-use electricity tariffs. CIRP Ann. 63 (1), 37–40.

Zitzler, E., 1999. Evolutionary algorithms for multiobjective optimization: Methods and applications, (Vol. 63).. Shaker, Ithaca.