Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm

Mehdi Mahmoodjanloo, Reza Tavakkoli-Moghaddam, Armand Baboli, Ali Bozorgi-Amiri

▶ **To cite this version:**

Mehdi Mahmoodjanloo, Reza Tavakkoli-Moghaddam, Armand Baboli, Ali Bozorgi-Amiri. Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm. Applied Soft Computing, 2020, 94, pp.106416. 10.1016/j.asoc.2020.106416 . hal-02906514

HAL Id: hal-02906514
https://hal.science/hal-02906514v1

Submitted on 22 Aug 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S1568494620303562
Manuscript_c26df4b89a8216d2e89b57524300a6f1

Flexible job shop scheduling problem with reconfigurable machine tools: An improved differential evolution algorithm

Mehdi Mahmoodjanloo [a], Reza Tavakkoli-Moghaddam [a,b,*], Armand Baboli [c,*], Ali Bozorgi-Amiri [a]

[a] *School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran*
[b] *Universal Scientific Education and Research Network (USERN), Tehran, Iran*
[c] *LIRIS laboratory, UMR 5205 CNRS, INSA of Lyon, 69621 Villeurbanne cedex, France*

Abstract

Developing reconfigurable machine tools (RMTs) has attracted increasing attention recently. An RMT can be utilized as a group of machines, which can obtain different configurations to satisfy manufacturing requirements. This paper deals with a production scheduling problem in a shop-floor with RMTs as an extension of a flexible job shop scheduling problem (FJSSP). To begin with, two mixed-integer linear programming models with the position- and sequence-based decision variables are formulated to minimize the maximum completion time (i.e., makespan). The CPLEX solver is used to solve the small- and medium-sized instances. The computational experiments show that the sequence-based model significantly outperforms the other one. Since even the sequence-based model cannot optimally solve most of the medium-sized problems, a self-adaptive differential evolution (DE) algorithm is proposed to efficiently solve the given problem. Moreover, the effectiveness of the proposed algorithm is enhanced by introducing a new mutation strategy based on a searching approach hired from a Nelder-Mead method. The performance of the proposed method and three other well-known variants of the DE algorithm are first validated by comparing their results with the results of the sequence-based model. Additional experiments on another data set including large-sized problems also confirm that the proposed algorithm is extremely efficient and effective.

Keywords: Flexible job shop; Configuration-dependent setup times; Industry 4.0; Self-adaptive differential evolution; Nelder-Mead mutation strategy.

1. Introduction

In recent decades, economic globalization and market competition lead to the rapid introduction of new products, more variants, low prices and high fluctuations in demand. Therefore, manufacturing systems are adjusting to satisfy these requirements. A significant approach to cope with these issues is the ability of reconfigurability for manufacturing systems and tools. Hence, a new class of production machines, called reconfigurable machine

* Corresponding authors.
Email addresses: mehdi.janloo@ut.ac.ir (M. Mahmoodjanloo), tavakoli@ut.ac.ir (R. Tavakkoli-Moghaddam), armand.baboli@insa-lyon.fr (A. Baboli), alibozorgi@ut.ac.ir (A. Bozorgi-Amiri).

© 2020 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

tools (RMTs), have been introduced. An RMT machine usually has a modular structure, which makes it able to obtain different configurations to satisfy manufacturing requirements. One of the benefits of developing RMTs is that the use of several different machines that share many costly and common modules while being rarely used at the same time can be prevented (Gadalla and Xue, 2017). An RMT in each configuration can process one or more operations with a certain rate. By changing the configuration, the machine can either perform some new operations or perform the same operation/operations with a different production rate (Moghaddam et al., 2019). In the development of RMTs, rapid conversion of the machine – decreasing the reconfiguration time – is one of the main objectives that can improve the responsiveness of a manufacturing system to produce highly customized products. Designing efficient RMTs and trying to decrease their reconfiguration time and to develop self-reconfigurable modular machines are the challenging and yet interesting problems (Aguilar et al., 2013; Hasan et al., 2013; Azulay, 2014; Pérez et al., 2014). Considering the high level of dynamism, it seems that the studies in production scheduling in these systems face new challenges.

In this paper, it is assumed that several jobs are assigned to a shop floor including several RMTs. Each job has a set of operations, which need to be processed in a specific order, and each operation can be processed at least on one configuration of one of the existing RMTs. Hence, the problem can be an extension of a flexible job shop scheduling problem (FJSSP). The objective is to minimize makespan (i.e., the maximum completion time of the jobs). The problem is more complicated than the FJSSP because three decisions have to be taken; these decisions include allocating of the operations to the machines, sequencing of the jobs and determining of the configuration of the machines to perform the allocated operations. Moreover, each RMT needs an amount of time to reconfiguration. We name it configuration-dependent setup time since it depends on the current and next configurations of the machine. Conversion time between every two configurations can be different because it needs to remove/add different auxiliary modules from/to the machine (Moghaddam et al., 2018).

The main contributions of this paper are as follows:

- Studying a new variant of a job shop scheduling problem (JSSP) that contains reconfigurable machine tools among the first studies in this area.
- Developing two mixed-integer linear programming (MILP) models based on operation-position and operation-sequence formulations and comparing them based on their computational performances.
- Extracting a lower bound for the problem.
- Developing a self-adaptive differential evolution (DE) algorithm to solve the problem efficiently.
- Enhancing the efficiency of the proposed algorithm by introducing a new mutation strategy inspired by the Nelder-Mead search approach.

The rest of the paper is organized as follows. Section 2 reviews the related literature. Section 3 presents two mathematical models and the formulation to calculate the lower bound of the problem. Section 4 proposes the solving approach. Sections 5 and 6 present the results of computational experiments and the conclusions, respectively.

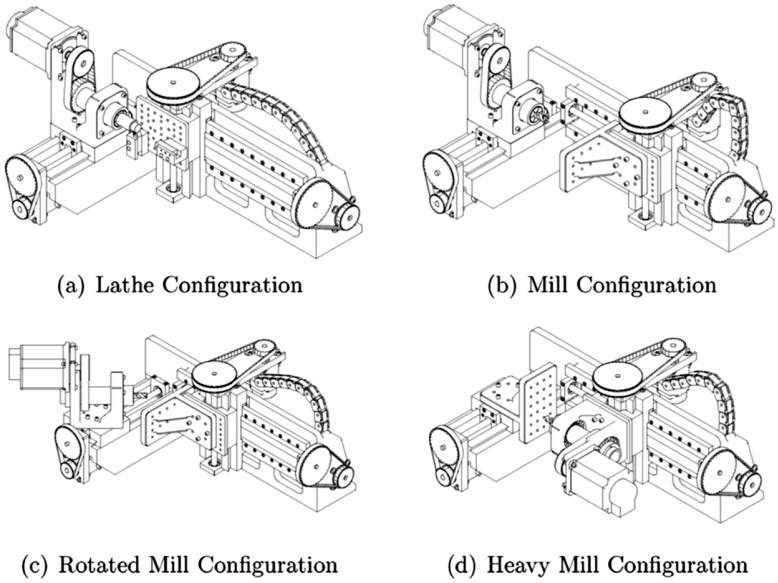
2. Literature review

2.1. Reconfigurable machine tools

For the first time, Koren and Kota (1999) developed a new generation of machines, named as RMTs, with the ability of easy and rapid change to perform different kinds of machining operations. Thereafter, the research on this kind of machines was initiated, and many researchers developed various prototypes of these tools (Gadalla and Xue, 2017). Indeed, an RMT can be used as a group of machines so that different functionality or capacity for a set of certain operations can be achieved through a change of its configurations. To compete with conventional machines, reconfiguration in RMTs must be done with the minimum loss of time. Hence, the concept of reconfigurability is defined as the ability to change the functionality or capacity of the RMT by changing/rearranging the components of the machine. Over the last two decades, a lot of research has been done to fulfill this objective, in which a few of them can be referred to as follows.

Ersal et al. (2004) presented a methodology to make an RMT able to be automatically reconfigured using a library of modular components. A modular reconfigurable machine has been developed by Padayachee and Bright (2012). They utilized a plug-and-play approach to control the scalability of the machine and developed a control system to support the modularity and reconfigurability. The developed machine could support turning and milling tasks. Pérez et al. (2014) developed a micro/mesoscale computer numerical control (CNC) machine tool with the ability of reconfiguration to do different machining operations (e.g., milling, drilling, and turning). Aguilar et al. (2013) designed a lathe-mill RMT and developed a prototype of the machine, which could be used in the jewelry industry. The RMT could achieve four different configurations including a mill configuration, a rotated mill configuration, a lathe configuration and a heavy mill configuration to perform various turning and milling tasks. Required time to change configurations among these four states was less than 15 minutes. Fig. 1 represents the four possible configurations for the lathe-mill RMT.

RMTs also play an important role in developing modern manufacturing approaches, like reconfigurable manufacturing systems (RMS). Indeed, an RMS is a system with the advantages of dedicated manufacturing systems (DMS) and flexible manufacturing systems (FMS), which is designed to adjust with rapid changes in volume or variety of market demand. To have a responsible and cost-effective system, RMSs need to have six characteristics including convertibility, scalability, modularity, customization, diagnosability, and integrability.



(a) Lathe Configuration (b) Mill Configuration

(c) Rotated Mill Configuration (d) Heavy Mill Configuration

Fig. 1. Four possible configurations in the lathe-mill RMT designed by Aguilar et al. (2013)

Development of RMSs is also a promising area for the achievement of Industry 4.0, known as the fourth industrial revolution or cyber-physical system. Qin et al. (2016) reviewed the significant characteristics of some current manufacturing systems (including single-station automated cells, automated assembly systems, computer-integrated manufacturing (CIM) systems, FMSs, and RMSs) to identify the research gaps between Industry 4.0 and these systems' requirements. They found that RMSs are the most potential production systems to achieve Industry 4.0 objectives such as the ability of self-optimization and self-configuration. One of the most important objective in Industry 4.0 is to produce individualized products at a reasonable cost or so-called mass-individualization. While traditional RMSs have been designed for high-volume manufacturing, it is emerged some new challenges to adjust them with the requirements of mass-individualization paradigm. Gu and Koren (2018) proposed a new manufacturing system architecture to satisfy these requirements. In the proposed architecture to achieve a high-mix/low-volume production at affordable costs, a material handling system based on forward/return conveyors (or gantry) is designed to provide a flexible routing of the parts among the stages. The system can also contain some high-level production machines (e.g., RMTs, CNCs, and additive manufacturing tools). Based on the suggested architecture, each job has its production route, and it can enter one stage more than once or bypass some stages by utilizing the designed transferring system. They highlighted some operational challenges of the suggested architecture, including the effective scheduling algorithms to improve the utilization of each machine. In this paper, we aim to study the scheduling problem in such systems.

2.2. FJSSP with family setup times

The considered scheduling problem has many similarities to the FJSSP with sequence-dependent family setup times. This problem is an extension of job shop scheduling problem, which was originally introduced by Brucker and Schlie (1990). In the FJSSP, each operation can be performed on a set of compatible machines. This is a common case especially in FMSs, whose machines have a high level of flexibility to perform variant operations (Rossi, 2014). The importance of such systems is that the availability of alternative machines can improve performance and reliability (Rohaninejad et al. 2015; Abdollahzadeh-Sangroudi and Ranjbar-Bourani, 2019). Moreover, considering setup times is an important issue in scheduling problems. It is an important characteristic that has a significant impact on the performance and applicability of many practical scheduling settings. In many real-world applications (e.g., automobile, pharmaceutical, and chemical manufacturing), machines need to a setup operation between jobs. Also, the setups may depend on the preceding process on the same machine (Shen et al. 2018).

In some production systems (e.g., Cellular Manufacturing Systems (CMSs)), there are several job families, which should be processed in the same production line (Gholipour-Kanani et al., 2012). Each family includes a set of jobs with similar features in terms of tooling, setups and operation sequence. To process two sequent jobs on the same machine, a setup operation is required if the jobs belong to two different families (Allahverdi, 2015).

In traditional RMSs, the scheduling problem can be described as follows. There is a product family, which can be classified into several product subfamilies, and each subfamily contains several jobs that should be done in a predesignated configuration. The objective is to determine an optimal sequence of subfamilies and optimal scheduling of the jobs inside each subfamily regarding one or more predefined performance criteria. Normally, performing the jobs inside each subfamily needs no setup time, while changing configuration to perform two different subfamilies requires a sequence-dependent setup time (Azab and Naderi, 2015). Based on the scheduling literature, such problem only needs to sequencing decisions, so it can, for example, be associated with the flow shop scheduling problems. On the other hand, scheduling in the above-mentioned modified RMS architecture can be associated to the JSSP because each job has its production route. Moreover, since each operation can be processed on more than one machine or on more than one machine-configuration, we have a FJSSP with group/family setup times.

Also, it is worth notable that the problem is different from the classic FJSSP with sequence-dependent family setup times because in this case, a job can belong simultaneously to more than one group (i.e., an operation can be performed on different configurations of a machine). Performing two consecutive operations on the same machine needs no setup time if both of them would be performed on the same configuration. Therefore, the setup times depend on the machine configurations rather than the job sequences. To the best of our knowledge, there is no study to tackle this problem. In this paper, regarding the existence of the research gaps in this area, a scheduling problem is modeled.

2.3. Corresponding solution methods

As mentioned before, the JSSP and its extensions are some of the most complicated combinatorial optimization problems, which are strongly NP-hard. So, even with a medium size, they cannot be solved exactly in a reasonable time. Therefore, some researchers have started to utilize heuristics and meta-heuristics to solve JSSPs in reasonably computational time (Jamili et al., 2011). Zhang et al. (2019) reviewed some of the most successful meta-heuristics to solve JSSPs, including Genetic Algorithm (GA), Tabu Search (TS), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Differential Evolution (DE), and Firefly Algorithm (FA).

In the current use, DE is one of the most powerful stochastic optimization algorithms, especially in the field of continuous optimization problems. The satisfying performance of DE in terms of robustness, convergence speed, and accuracy still makes it attractive for many researchers to apply DE in various real-world optimization problems (Das and Suganthan, 2011). Since more than two decades ago, DE and its variants have been placed among the best evolutionary algorithms as indicated by the IEEE Congress on Evolutionary Computation (CEC) competition series (Yuen and Zhang, 2015). Although DE was developed at first for the problems with continuous search space, DE-based approaches have been applied successfully in the field of discrete optimization, such as scheduling problems. However, DE alone could not be efficient to solve scheduling problems. The reason may be that the searching mechanisms of the classic DE (especially the mutation operator) are not able to be adapted on a permutation representation (Ponsich et al., 2009). Hence, to make the algorithm more efficient, some local search approaches have been hybridized with DE in the literature (e.g., a combination of DE with TS) to solve the JSSP (Ponsich and Coello, 2013), embedding a local search algorithm based on the critical path into DE to solve the FJSSP (Yuan and Xu, 2013), introducing a chaotic strategy to update the parameters and two new mutation operators for DE to solve the JSSP (Zhang et al., 2016), embedding a speed-up neighborhood search procedure into DE to solve the FJSSP (Zhao et al., 2016), combination of DE with Simulated Annealing (SA) to solve the distributed FJSSP (Wu and Liu, 2018; Wu et al., 2018).

The Nelder-Mead (NM) simplex search is a numerical optimization method, which is designed for unconstrained problems with multidimensional space. It is a direct search method, which can converge to non-stationary points without using gradient information. The procedure of the NM algorithm is based on geometric operators contain reflection, expansion, contraction and shrinking. By taking advantages of these operators, NM is known as an effective method in local search; however, it has some weaknesses in global search (Wang et al., 2011). Hence, to utilize its exploitation abilities, there are lots of efforts in the literature to combine NM with meta-heuristics as global exploration approaches (Chelouah and Siarry, 2005; Moravec and Rudolf, 2018). Since NM is a close relative of DE, some researchers have been motivated to design hybrid approaches based on NM and DE (for

example see: Menchaca-Mendez and Coello, 2009; Moraglio and Johnson, 2010; Gao et al., 2011; Wang et al., 2011; Fan and Yan, 2015a).

In this research, a new mutation strategy based on the NM operators has been introduced, whose aim is to speed up convergence and to strengthen the exploitation phase of the DE algorithm. Moreover, to more efficiently control the parameters and mutation strategies, a self-adaptive strategy based on the research of Fan and Yan (2015a) has been developed.

3. Problem description and MILP modelling

3.1. Problem definition

The FJSSP with machine configuration-dependent setup times (FJSSP-CDST) can be described as follows. There is a set K of RMTs on a shop floor with a predefined layout. Each RMT k has a set of c_k configurations. One or more operations can be processed in each configuration with a special rate. A set $\mathcal{J}$ of n jobs should be processed in the shop floor. Each job i has a set of n_i operations with a predefined sequence that has already been determined by process planning unit (e.g., $O_{i,1} \rightarrow O_{i,2} \rightarrow \cdots \rightarrow O_{i,n_i}$). Moreover, it is assumed that each operation j of job i (O_{ij}) can be processed at least on one configuration of one of the existing RMTs. No setup is needed to perform operations in a machine configuration, while to switch to a different configuration on the machine, the RMT needs to a setup that is dependent on two consecutive configurations. It is assumed that the configuration-dependent setup times satisfy the triangular inequality. Moreover, each RMT can only fit into one configuration at a time, and it cannot perform more than one operation simultaneously. All the jobs are available at time 0, and preemption is not allowed. The main decisions of the problem include allocating of each operation to an eligible RMT, sequencing of the jobs and determining the appropriate configurations of each machine to perform the allocated operations. The objective is to minimize makespan.

3.2. Mathematical formulations

In this section, two different MILP models of the problem are developed based on operation-position and operation-sequence formulations. The following sets, indices and parameters are used in both developed models.

Sets and indices:

$\mathcal{J}$ Set of jobs, and each job $i \in \mathcal{J}$

N_i Set of operations related to job i, where the operation j of job i is denoted by O_{ij}

K Set of machines and index $k \in K$

C_k Set of configurations of machine k and index $c \in C_k$

Parameters:

p_{ijkc} Processing time of operation O_{ij} on cth configuration of machine k

r_{ijkc}	1 if operation O_{ij} can be processed on configuration c of machine k; 0, otherwise
$ST_{c_1,c_2,k}$	Configuration-dependent setup time when the configuration is changed from c_1 to c_2 (i.e., $c_1 \neq c_2$) on machine k
M	A big positive number

3.2.1. Operation-position based (PB) model

At the first model, the scheduling problem has been considered as positioning decisions (i.e., assigning the operations to some predefined time-positions). Hence, we need to define a new set L_k for the positions of the machine k and its related index l, in which an operation can be processed or setup can be done.

l Index for job positions processed on machine k, index $l \in L_k$ (i.e., $|L_k| = \sum_{i\in J}\sum_{j\in N_i}\sum_{c\in C_k} r_{ijkc}$) and $L = \bigcup_{k\in K} L_k$.

Decision variables of the PB model:

y_{ijklc}	Binary variable. If operation O_{ij} is processed on position l of machine k (i.e., the position kl) with configuration c, then $y_{ijklc} = 1$; otherwise, $y_{ijklc} = 0$.
$Z_{c_1,c_2,k,l}$	Binary variable. If at the beginning of position kl, the machine's configuration is changed from c_1 to c_2 (i.e., $c_1 \neq c_2$), then $Z_{c_1,c_2,k,l} = 1$; otherwise, $Z_{c_1,c_2,k,l} = 0$.
F_{ij}	Completion time of operation O_{ij}
F'_{kl}	Finishing time of position kl
C_{max}	Completion time (i.e., makespan)

The PB model is as follows:

$$\text{Min } C_{max} \tag{1}$$

s.t.

$$\sum_{k\in K}\sum_{l\in L_k}\sum_{c\in C_k} y_{ijklc} = 1 \qquad \forall\, i \in \mathcal{J}, j \in N_i \tag{2}$$

$$\sum_{l\in L_k} y_{ijklc} \leq r_{ijkc} \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \tag{3}$$

$$\sum_{i\in \mathcal{J}}\sum_{j\in N_i}\sum_{c\in C_k} y_{ijklc} \leq 1 \qquad \forall\, k \in K, l \in L_k \tag{4}$$

$$\sum_{i\in \mathcal{J}}\sum_{j\in N_i}\sum_{c\in C_k} y_{ijk,l-1,c} \geq \sum_{i\in \mathcal{J}}\sum_{j\in N_i}\sum_{c\in C_k} y_{ijklc} \qquad \forall\, k \in K, l \in L_k, l \neq 1 \tag{5}$$

$$\sum_{i\in \mathcal{J}}\sum_{j\in N_i} y_{ijk,l-1,c_1} + \sum_{i\in \mathcal{J}}\sum_{j\in N_i} y_{ijk,l,c_2} \geq 2\, Z_{c_1,c_2,k,l} \qquad \forall\, k \in K, l \in L_k, c_1, c_2 \in C_k, c_1 \neq c_2 \tag{6}$$

$$\sum_{i\in \mathcal{J}}\sum_{j\in N_i} y_{ijk,l-1,c_1} + \sum_{i\in \mathcal{J}}\sum_{j\in N_i} y_{ijk,l,c_2} - 1 \leq Z_{c_1,c_2,k,l} \qquad \forall\, k \in K, l \in L_k, c_1, c_2 \in C_k, c_1 \neq c_2 \tag{7}$$

$$F_{ij} \geq F_{i,j-1} + \sum_{k\in K}\sum_{l\in L_k}\sum_{c\in C_k} p_{ijkc}\, y_{ijklc} \qquad \forall\, i \in \mathcal{J}, j \in N_i \tag{8}$$

$$F'_{kl} \geq F'_{k,l-1} + \sum_{c_1\in C_k}\sum_{\substack{c_2\in C_k \\ c_1\neq c_2}} ST_{c_1,c_2,k}\, Z_{c_1,c_2,k,l} + \sum_{i\in\mathcal{J}}\sum_{j\in N_i}\sum_{c\in C_k} p_{ijkc}\, y_{ijklc} \qquad \forall\, k \in K, l \in L_k \tag{9}$$

$$F'_{kl} \leq F_{ij} + M\left(1 - \sum_{c\in C_k} y_{ijklc}\right) \qquad \forall\, k \in K, l \in L_k, i \in \mathcal{J}, j \in N_i \tag{10}$$

$$F'_{kl} \geq F_{ij} - M\left(1 - \sum_{c\in C_k} y_{ijklc}\right) \qquad \forall\, k \in K, l \in L_k, i \in \mathcal{J}, j \in N_i \tag{11}$$

$$F_{in_i} \leq C_{max} \qquad \forall\, i \in \mathcal{J} \tag{12}$$

$$y_{ijklc}, Z_{c_1,c_2,k,l} \in \{0,1\} \qquad \forall\, k \in K, l \in L_k, c \in C_k, i \in \mathcal{J}, j \in N_i \tag{13}$$

$$F_{ij}, F'_{kl} \geq 0 \qquad \forall\, k \in K, l \in L_k, i \in \mathcal{J}, j \in N_i \tag{14}$$

Eq. (1) is the objective function. Constraints (2) ensure that each operation should be assigned to one position of an existing machine configuration. If operation O_{ij} is not allowed to be processed on configuration c of machine k ($r_{ijkc} = 0$), then Constraint (3) prevent the assignment of the operation O_{ij} to any positions of the machine configuration kc. Constraint set (4) is to show that in each machine position kl at most one operation can be processed. Constraints (5) ensure that each machine position can be assigned only when the previous position is allocated. Constraints (6) and (7) ensure that for each machine, if two consecutive positions are allocated to different configurations, then a proper setup should be done. Constraints (8) guarantee that the completion time of the operation O_{ij} should be greater than the completion time of the previous operation plus processing time of O_{ij}. Constraints (9) guarantee that the completion time of each machine position should be greater than the completion time of the previous position plus a possible setup time and the processing time of an operation, which is allocated to the position. Constraints (10) and (11) ensure that the completion time of each operation should be set with its associated position. Constraint set (12) is used to determine the completion time. F_{in_i} is the completion time of the last operation of job i. In other words, Constraints (12) are the linear form of $C_{max} = \max_{i\in\mathcal{J}}\{F_{in_i}\}$. Constraints (13) and (14) define the decision variables.

3.2.2. Operation-sequence based (SB) model

At the second model, the scheduling problem is considered as a sequencing decision.

Decision variables of the SB model:

x_{ijkc} 1 if operation O_{ij} is processed on configuration c of machine k; 0, otherwise

$U_{iji'j'}$	1 if operation O_{ij} is scheduled before $O_{i'j'}$, then $U_{iji'j'} = 1$; 0, otherwise
F_{ij}	Completion time of operation O_{ij}
C_{max}	Maximum completion time (i.e., makespan)

The SB model is as follows:

$$\text{Min } C_{max} \tag{15}$$

s.t.

$$\sum_{k\in K}\sum_{c\in C_k} x_{ijkc} = 1 \qquad \forall\, i \in \mathcal{J}, j \in N_i \tag{16}$$

$$x_{ijkc} \le r_{ijkc} \qquad \forall\, k \in K, c \in C_k, i \in \mathcal{J}, j \in N_i \tag{17}$$

$$F_{ij} \ge F_{i,j-1} + \sum_{k\in K}\sum_{c\in C_k} p_{ijkc}\, x_{ijkc} \qquad \forall\, i \in \mathcal{J}, j \in N_i \tag{18}$$

$$F_{ij} \ge F_{i'j'} + p_{ijkc_2} + ST_{c_1,c_2,k} - \left(2 - x_{ijkc_2} - x_{i'j'kc_1} + U_{iji'j'}\right)M \qquad \forall\, k \in K, c_1, c_2 \in C_k, i, i' \in \mathcal{J}, j \in N_i, j' \in N_{i'}, O_{ij} \ne O_{i'j'} \tag{19}$$

$$F_{i'j'} \ge F_{ij} + p_{i'j'kc_2} + ST_{c_1,c_2,k} - \left(3 - x_{ijkc_1} - x_{i'j'kc_2} - U_{iji'j'}\right)M \qquad \forall\, k \in K, c_1, c_2 \in C_k, i, i' \in \mathcal{J}, j \in N_i, j' \in N_{i'}, O_{ij} \ne O_{i'j'} \tag{20}$$

$$C_{max} \ge F_{in_i} \qquad \forall\, i \in \mathcal{J} \tag{21}$$

$$x_{ijkc} \in \{0,1\} \qquad \forall\, k \in K, c \in C_k, i \in \mathcal{J}, j \in N_i \tag{22}$$

$$U_{iji'j'} \in \{0,1\} \qquad \forall\, i, i' \in \mathcal{J}, j \in N_i, j' \in N_{i'}, O_{ij} \ne O_{i'j'} \tag{23}$$

$$F_{ij} \ge 0 \qquad \forall\, i \in \mathcal{J}, j \in N_i \tag{24}$$

Eq. (15) is the objective function. Constraint (16) ensures that each operation should be assigned to one machine configuration. If operation O_{ij} is not allowed to be processed on configuration c of machine k ($r_{ijkc} = 0$), then Constraint (17) prevent the assignment of operation O_{ij} to machine configuration kc. Constraint (18) guarantees that the completion time of operation O_{ij} should be greater than the completion time of the previous operation plus processing time of O_{ij}. On each machine k, Constraints (19) and (20) prevent the overlapping of the allocated operations. Constraint set (21) is used to determine the completion time. Constraints (22) – (24) define the decision variables.

3.3. Lower-bound method

The FJSSP-CDST is a generalization of the classical JSSP, and the JSSP is NP-hard (Garey et al., 1976), therefore the FJSSP-CDST is also an NP-hard problem. Moreover, by adding the machine configuration decisions, the problem is even more complicated than some other extensions of JSSP (e.g., flexible JSSP with sequence-dependent setup times). Hence, developing a lower bound (LB) can be helpful to analyze the outcomes of proposed

algorithms. The goal is to develop a new formulation, whose complex constraints relaxation provides an LB for the original problem.

The developed models contain three types of decisions including machine configuration, job assignment and sequencing. Since the complexity of the models results from the sequencing decisions, relaxation of this type of decisions can decrease the complexity of the models. Hence, the obtained objective value can be considered as an LB for makespan. The independent variables in the proposed models, including x_{ijkc} and $U_{iji'j'}$ in the SB model and y_{ijklc} in the PB model, deal with three types of decisions in the problem. Variable $U_{iji'j'}$ deal with sequencing decisions in the SB model. In the PB model, how the positioning of the operations on each machine determines the sequence of them. Hence, extracting the lower bound based on the PB model needs to decompose the variable y_{ijklc}. Because of its simpler structure, we prefer to extract the LB based on the SB model.

In shop scheduling problems, a LB for completion time can be achieved based on machine lower bound (LB_1) (i.e., the maximum of the total processing time of operations allocated to the machines) and job lower bound (LB_2) (i.e., the maximum of the total processing time of the jobs). Finally, the lower bound of the problem can be achieved as $LB = \max\{LB_1, LB_2\}$.

Proposition 1. LB_1 is an LB of the total processing time of the operations allocated to the machines.

$$LB_1 = \max_{k\in K}\left\{\sum_{i\in\mathcal{J}}\sum_{j\in N_i}\min_{\{c\in C_k | r_{ijkc}\neq 0\}}\{p_{ijkc}\}\right\} \tag{25}$$

Proof. For each machine $k \in K$, an LB for the allocated operations can be achieved if each operation is processed on an admissible configuration of the machine with the minimum processing time without considering any reconfiguration time. □

Proposition 2. LB_2 is an LB for the total processing time of each job.

$$LB_2 = \max_{i\in\mathcal{J}}\left\{\sum_{j\in N_i}\min_{\{k\in K, c\in C_k | r_{ijkc}\neq 0\}}\{p_{ijkc}\}\right\} \tag{26}$$

Proof. For each job $i \in \mathcal{J}$, a lower bound for the operations can be achieved if each operation is processed on an admissible machine configuration with the minimum processing time. □

To incorporate the machine and job lower bounds into the SB model, Constraints (28) – (30) and (31) – (33) should be considered to linearize Eq. (25) and Eq. (26), respectively.

$$\text{Min}_x\ \zeta_{LB} = C_{max} \tag{27}$$

s.t.

Constraint sets (16) and (17)

$$\Theta_{ijk} \le p_{ijkc} + M(1 - x_{ijkc}) \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \qquad (28)$$

$$\Theta_{ijk} \ge p_{ijkc} - M(1 - x_{ijkc}) \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \qquad (29)$$

$$C_{max} \ge \sum_{i \in J} \sum_{j \in N_i} \Theta_{ijk} \qquad \forall\, k \in K \qquad (30)$$

$$\Phi_{ij} \le p_{ijkc} + M(1 - x_{ijkc}) \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \qquad (31)$$

$$\Phi_{ij} \ge p_{ijkc} - M(1 - x_{ijkc}) \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \qquad (32)$$

$$C_{max} \ge \sum_{j \in N_i} \Phi_{ij} \qquad \forall\, i \in \mathcal{J} \qquad (33)$$

$$x_{ijkc} \in \{0,1\} \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K, c \in C_k \qquad (34)$$

$$C_{max}, \Phi_{ij}, \Theta_{ijk} \ge 0 \qquad \forall\, i \in \mathcal{J}, j \in N_i, k \in K \qquad (35)$$

where Θ_{ijk} is the processing time of O_{ij} on machine k if the operation is assigned to one of the admissible configurations of the machine. Constraints (16) and (17) ensure that each operation is assigned to an admissible machine configuration. Since there are not any constraints to select the admissible configurations of the machine, Constraints (28) and (29), and a minimization logic of the objective function guarantee that the minimum processing time is selected to perform each operation, hence $\Theta_{ijk} = \min_{\{c \in C_k | r_{ijkc} \neq 0\}} \{p_{ijkc}\}$. Moreover, based on Eq. (25), we have $LB_1 = \max_{k \in K} \{\sum_{i \in \mathcal{J}} \sum_{j \in N_i} \Theta_{ijk}\}$. Therefore $LB_1 \ge \sum_{i \in J} \sum_{j \in N_i} \Theta_{ijk}$. The same procedure is used for the job lower bound (i.e., $LB_2 \ge \sum_{j \in N_i} \Phi_{ij}$). Finally, since the LB of makespan is equal to the maximum of LB_1 and LB_2, Constraints (30) and (33) can be extracted.

4. Solution Algorithm

The proposed self-adaptive DE algorithm based on the Nelder-Mead mutation strategy (SADE-NMMS) has been presented as follows.

4.1. Required basic concepts

4.1.1. Differential evolution (DE)

DE is a simple, reliable, and efficient population-based stochastic optimization technique (Storn & Price, 1997). The classic DE algorithm has three parameters contain the number of population (NP), mutation scale factor (F) and crossover rate (CR). The algorithm starts by generating a population of NP random solutions, called individuals. Then the main loop of the algorithm starts working until a termination criterion is met (e.g., for a number of iterations). In each iteration, mutation and crossover operators should be done for each individual to obtain a trial vector (new solution), respectively. Eventually, a greedy approach is done to select between the trial vector and the individual. The selected one is promoted to the next iteration. The process of mutation and crossover for an individual $X \in R^n$ can be described as follows.

Mutation: For individual X_i, three distinct randomly selected individuals with indices r_1, r_2 and r_3 (i.e., $i \neq r_1 \neq r_2 \neq r_3$) are considered to generate a mutated individual V_i as follows.

DE/rand/1:

$$V_i = X_{r_1} + F.\left(X_{r_2} - X_{r_3}\right) \tag{36}$$

Crossover: Based on current individual X_i and mutated individual V_i, a trial individual T_i is formed by:

$$t_{ij} = \begin{cases} v_{ij} & \text{if } r \leq CR \text{ or } j = j_{rand} \\ x_{ij} & \text{otherwise} \end{cases} \qquad \forall\, j = 1, 2, \ldots, n \tag{37}$$

where t_{ij} is element j of T_i, the CR is the crossover rate, $r = rand[0, 1]$, and j_{rand} is an index, which is randomly selected to guarantee that at least one element of the mutated individual should be chosen.

4.1.2. Nelder-Mead simplex search

The Nelder-Mead (NM) simplex search, originally published in 1965 (Nelder and Mead, 1965), is one of the best known numerical optimization method designed for unconstrained problems with multidimensional space. In n-dimensional solution space ($X \in R^n$), NM initializes with n+1 random solutions (called individuals) and orders them based on their function values from the best to the worst (e.g., to minimize a function $f(X)$), suppose $f(X_1) \leq f(X_2) \leq \cdots \leq f(X_{n+1})$. Then a centroid ($\bar{X}$) of the n best individuals should be calculated ($\bar{X} = \sum_{i=1}^{n} X_i$). Considering four factors including reflection coefficient ($\alpha_r > 0$), expansion coefficient ($\alpha_e > 1$), contraction coefficient ($0 < \alpha_c < 1$) and shrinkage coefficient ($0 < \alpha_s < 1$), the steps of the algorithm can be described below. Based on these steps, the algorithm starts working until a termination condition is met.

Reflection: Compute the reflected point as follows.

$$X_r = \bar{X} + \alpha_r.\left(\bar{X} - X_{n+1}\right) \tag{38}$$

If the reflected point is better than the best point (i.e., $f(X_r) < f(X_1)$), then the expansion phase should be applied, else if $f(X_1) \leq f(X_r) < f(X_n)$, the reflected point should be replaced with the worst point X_{n+1}, and the iteration should be terminated. Otherwise, if $f(X_n) \leq f(X_r)$, the contraction phase should be applied.

Expansion: Compute the expansion point as follows.

$$X_e = \bar{X} + \alpha_e.\left(X_r - \bar{X}\right) \tag{39}$$

If the expanded point is better than the reflected point (i.e., $f(X_e) \leq f(X_r)$), then the expanded point should be replaced with the worst point X_{n+1}; otherwise, the reflected point should be replaced with the worst point, and the iteration should be terminated.

Contraction: Compute the contraction point between the centroid and the better of the two points X_r and X_{n+1}. If $f(X_{n+1}) \leq f(X_r)$, then the inside contraction should be performed using Eq. (40). Otherwise, the outside contraction should be performed using Eq. (41).

$$X_c = \overline{X} - \alpha_c.(X_r - \overline{X}) \tag{40}$$
$$X_c = \overline{X} + \alpha_c.(\overline{X} - X_{n+1}) \tag{41}$$

In inside contraction, if $f(X_c) \leq f(X_{n+1})$, or in outside contraction, if $f(X_c) \leq f(X_r)$, then X_c should be replaced with the worst point, and the iteration should be terminated. Otherwise, the shrinking phase should be applied.

Shrink: This operator is applied to converge the points around the best point to adjust the accuracy of the algorithm. The operator is performed as follows.

$$X_i' = X_1 + \alpha_s.(X_i - X_1) \qquad \forall\, i = 2, 3, \dots, n+1 \tag{42}$$

Eventually, the set $\{X_1, X_2', X_3', \dots, X_{n+1}'\}$ can be used for the next iteration.

4.1.3. Proposed Nelder-Mead mutation strategy

In the literature, different mutation strategies have been utilized to enhance the exploitation and exploration capabilities of the DE variants. Many pieces of evidence reveal that different optimization problems require different strategies for parameter settings and mutation (Qin et al., 2009). Most of the proposed mutation strategies in the literature have a simple structure to randomly obtain a single target. For example, the mutation strategy "rand/1", which is defined in Eq. (36), is a suitable strategy for an exploration target, while another strategy named "current-to-best/1" is usually used for exploitation. Hence, some researchers have started to develop new intelligent approaches to utilize various capabilities of the different mutation strategies (Fan and Yan, 2015b). Besides these approaches, which focus on making intelligent the whole of the algorithm, it seems that developing some new flexible mutation strategies can be advantageous to search the solution space more efficient. Inspiring the Nelder-Mead operators, we develop a new mutation strategy, which can intelligently search the solution space. The procedure of the *NM-based-rand/3* mutation strategy is presented in Fig. 2. The number "3" refers to the number of initial points as a parameter of the algorithm to start the searching procedure. It is worth mentioning that the parameter can be tuned for different utilizations.

To illustrate how the proposed mutation strategy can efficiently search the solution space, an instance based on various mutation strategies is considered. Current point x^i and three randomly selected individuals are considered, which can be ordered as x^{r_3}, x^{r_2} and x^{r_1} (i.e., $f(x^{r_3}) < f(x^{r_2}) < f(x^{r_1})$). In Fig. 3, three mutated individuals based on some of the most popular mutation strategies are represented. As can be seen from the figures, none of these strategies utilizes all of the useful information. In a similar situation, the potential mutated points, based on the proposed strategy, is represented in Fig. 4. Some other information (e.g., the position of the worst point and the weight of the rest individuals) is utilized. Moreover, coefficients α_r, α_e, α_c and α_s help to improve efficiency. For example, the higher values of the reflection and explosion coefficients, more effective exploration, and the lower values of the contraction and shrinkage coefficients, the more effective exploitation.

4.2. Details of the proposed SADE-NMMS to solve FJSSP-CDST

To efficiently solve the FJSSP-CDST as a combinatorial optimization problem with discrete solution space, an improved variant DE is proposed based on the new mutation strategy and a self-adaptive procedure. The main steps of the algorithm are presented in Fig. 5. Moreover, the details of the algorithm are presented below.

Select three indices randomly, and rank them decreasingly. Suppose that the indices r_1 and r_3 are respectively related to the worst and to the best individual ($f(x^{r_1}) \geq f(x^{r_2}) \geq f(x^{r_3})$);
Calculate the *centroid* $\bar{X}$ as the weighted arithmetic mean of x^{r_2} and x^{r_3}, i.e. $\bar{X} = \frac{f(x^{r_3})x^{r_2} + f(x^{r_2})x^{r_3}}{f(x^{r_2}) + f(x^{r_3})}$;

Calculate the *reflection* point $a_2 = \bar{X} + \alpha_r(\bar{X} - x^{r_1})$;
If $f(a_2) < f(x^{r_3})$
    Calculate the *expansion* point $a_1 = \bar{X} + \alpha_e(a_2 - \bar{X})$;
    If $f(a_1) < f(x^{r_3})$
        Set the mutation vector $v^i = a_1$;
    Else
        Set the mutation vector $v^i = a_2$;
    End
Else
    Calculate the *contraction* points a_3 and a_4 as follows:
        $a_3 = \bar{X} + \alpha_c(\bar{X} - x^{r_1})$;
        $a_4 = \bar{X} - \alpha_c(\bar{X} - x^{r_1})$;
    Get the best *contraction* point as follow:
        $a_5 = arg\ \min(f(a_3), f(a_4))$;
    If $f(a_5) < f(x^{r_3})$
        Set the mutation vector $v^i = a_5$;
    Else
        Shrink to the best individual as follow:
            $v^i = x^i + \alpha_s(x^{r_3} - x^i)$;
    End
End
Return v^i

Fig. 2. Procedure of the *NM-based-rand/3* mutation strategy

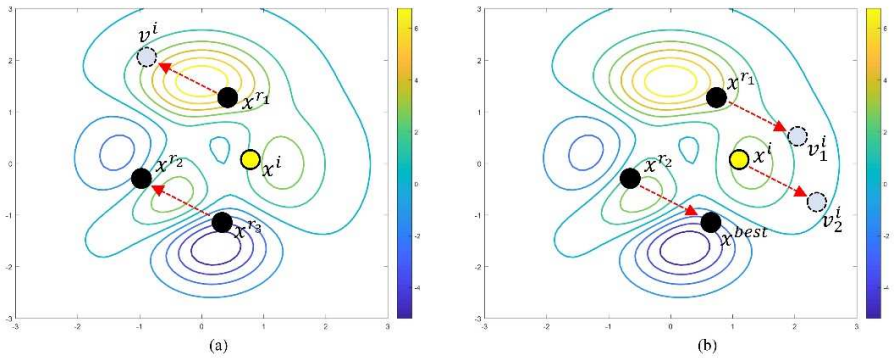


Fig. 3. Three well-known mutation strategies in the DE algorithm

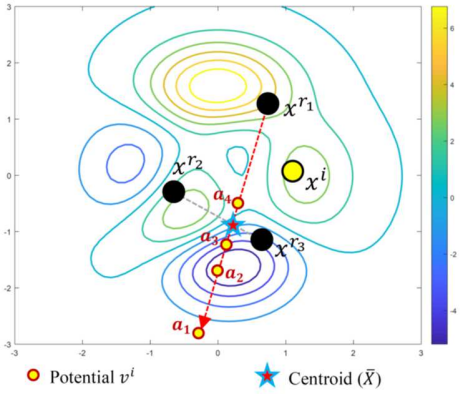


Fig. 4. *NM-based-rand/3* mutation strategy for the DE algorithm

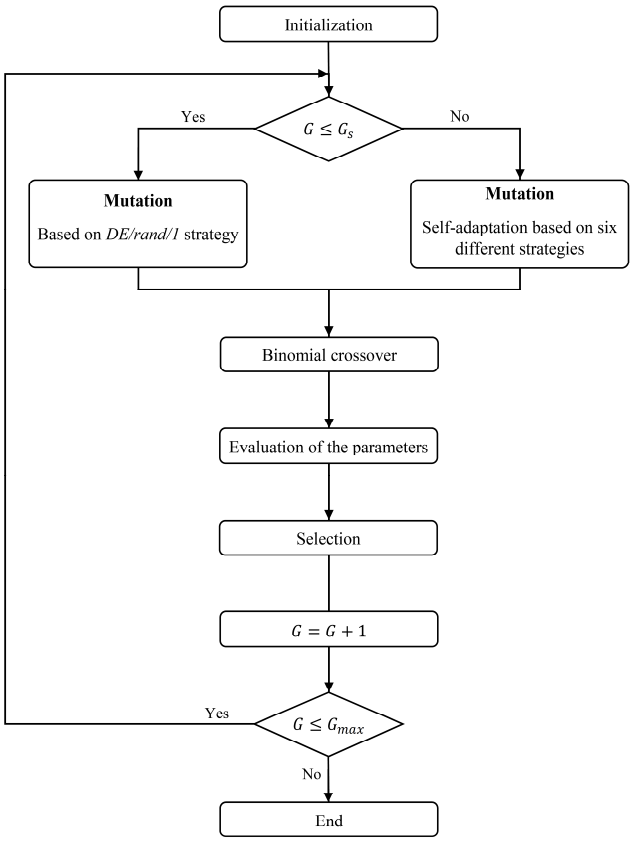


Fig. 5. Main flow chart of SADE-NMMS

4.2.1. Encoding

To apply a DE-based algorithm for solving FJSSP-CDST as a discrete optimization problem, it is needed to design an appropriate encoding approach with real values to represent a feasible solution. Each solution needs to present the operations scheduling and the machine configuration simultaneously. Hence, we design an encoding method including two parts, which can be represented as a two-dimensional matrix ($\mathbb{A}_{2\times n}$). The solution matrix has two rows. The first row is related to the machine configuration decisions, and the second row is related to the operations scheduling decisions. The number of columns is equal to the total number of operations (i.e., $n = \sum_{i\in\mathcal{J}} N_i$).

To better illustrate the encoding scheme, we design a simple instance with two jobs and two machines. The first job needs four operations, and the second job needs two operations. Moreover, each machine has two configurations. Table 1 shows the processing time of the operations on each machine configuration (p_{ijkc}). The reconfiguration times for the machines are as follows: $ST_{c1,c2,1} = 65$, $ST_{c2,c1,1} = 85$, $ST_{c1,c2,2} = 90$ and $ST_{c2,c1,2} = 120$.

Table 1. Processing times of the example

Machine	Configuration	Job 1				Job 2	
		O_{11}	O_{12}	O_{13}	O_{14}	O_{21}	O_{22}
1	1	150				140	
	2			80	140	120	
2	1	200	160				80
	2		90			90	

Fig. 6 shows a random solution (chromosome) of the example. To determine the sequencing of the operations based on the chromosome, the value of the elements associated to the first operation of the jobs should be compared (e.g., $\mathbb{a}_{2,1}$ and $\mathbb{a}_{2,5}$), where $\mathbb{a}_{i,j}$ refers to the element (i,j) of the chromosome. The operation with a smaller value is selected, and then, the next operation should be compared with remain operations of other jobs. For example, since $\mathbb{a}_{2,1}$ is smaller than $\mathbb{a}_{2,5}$ (i.e., $0.15 < 0.32$), O_{11} should be selected as the first operation. Thereafter, the next operation of job 1 (O_{12}) should be compared with O_{21}. Since $\mathbb{a}_{2,2}$ is not smaller than $\mathbb{a}_{2,5}$ ($0.51 > 0.32$), O_{21} should be selected as the second operation. The procedure continues until all of the operations be placed on the sequence. Based on this procedure, the final sequence of the example can be extracted as $O_{11} \rightarrow O_{21} \rightarrow O_{12} \rightarrow O_{13} \rightarrow O_{14} \rightarrow O_{22}$.

	O_{11}	O_{12}	O_{13}	O_{14}	O_{21}	O_{22}
Conf. selection	0.53	0.44	0.08	0.29	0.67	0.92
Sequencing	0.15	0.51	0.23	0.74	0.32	0.86

Fig. 6. Random solution (chromosome) for the example

To select a machine configuration for each operation, the first row of the chromosome and a probability matrix ($\bar{P}$) should be utilized. Each element of the probability matrix (i.e., $\bar{P} = \|\bar{p}_{ijkc}\|$) can be calculated as Eq. (43). The probability matrix of the example is presented in Fig. 7. For example, as it presented in Table 1, O_{11} can be performed on both machine-configurations $k = 1, c = 1$ and $k = 2, c = 1$ with processing times 150 and 200, respectively. Hence, we can calculate the associated elements as: $\bar{p}_{1,1,1,1} = \left(\frac{1}{150}\right)/\left(\frac{1}{150} + \frac{1}{200}\right) = 0.571$ and $\bar{p}_{1,1,2,1} = \left(\frac{1}{200}\right)/\left(\frac{1}{150} + \frac{1}{200}\right) = 0.429$. Therefore, the first configuration of the first machine, which is faster in performing O_{11}, has more chance to be selected.

$$\bar{p}_{ijkc} = \frac{\frac{1}{p_{ijkc}}}{\sum_{k \in K, c \in C_k} \frac{1}{p_{ijkc}}} \qquad \forall\, i \in \mathcal{J}, j \in N_i, p_{ijkc} \neq 0 \qquad (43)$$

	O_{11}	O_{12}	O_{13}	O_{14}	O_{21}	O_{22}
$k = 1, c = 1$	0.571				0.269	
$k = 1, c = 2$			1	1	0.313	
$k = 2, c = 1$	0.429	0.36				1
$k = 2, c = 2$		0.64			0.418	

Fig. 7. Probability matrix ($\bar{P}$) for machine-configuration selection

Based on the presented chromosome in Fig. 6 and the extracted probability matrix in Fig. 7, a machine configuration for each operation can be selected. To do this, the value of each element in the first row of the chromosome should be compared with the cumulative summation of the related column in the probability matrix. For example, O_{11} should be processed on the first configuration of the machine 1 because of $\mathbb{a}_{1,1} \in [0, 0.571]$, and O_{12} should be processed on the second configuration of the machine 2 because of $\mathbb{a}_{1,2} \in (0.36, 1]$. The resultant schedule of the chromosome is presented in Fig. 8. Makespan of the schedule is equal to 460.

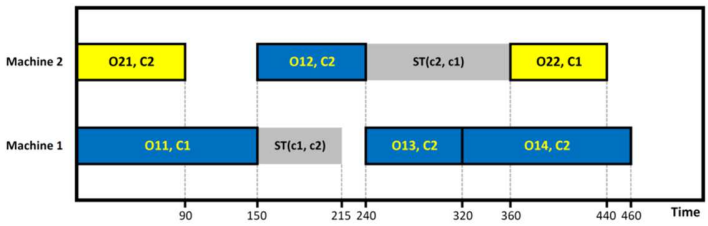


Fig. 8. Resultant schedule of the chromosome

4.2.2. Initialization

The initialization phase starts by random generation of *NP* chromosomes (i.e., individuals). Each individual should be decoded to evaluate the resultant schedule and extract the makespan. These individuals can be considered as the initial generation (i.e., $G = 0$). Thereafter, the mutation and crossover operators should be utilized iteratively to search the solution space.

4.2.3. Mutation and crossover

To balance between exploitation and exploration abilities of the algorithm, the generations of DE are divided into two phases. At the first phase ($G \leq G_s$), to enhance the exploration ability, the mutation strategy *rand/1* is utilized alone, where G_s is a predefined point to change the searching strategy (i.e., $G_s = \eta \times G_{max}$ and $\eta \in [0, 1]$). Thereafter, at the second phase ($G_s < G \leq G_{max}$), six mutation strategies including *rand/1*, *rand/2*, *best/2*, *current-to-best/1*, current-to-best/2 and *NM-based-rand/3* can be utilized based on a self-

adaptive manner. Other useful strategies including *rand/2*, *best/2*, *current-to-best/1* and current-to-best/2 are respectively as Eqs. (44) – (47):

DE/rand/2:

$$V_i^{G+1} = X_{r_1}^G + F.\left(X_{r_2}^G - X_{r_3}^G\right) + F.\left(X_{r_4}^G - X_{r_5}^G\right) \quad (44)$$

DE/best/2:

$$V_i^{G+1} = X_{best}^G + F.\left(X_{r_1}^G - X_{r_2}^G\right) + F.\left(X_{r_3}^G - X_{r_4}^G\right) \quad (45)$$

DE/current-to-best/1:

$$V_i^{G+1} = X_i^G + F.\left(X_{best}^G - X_i^G\right) + F.\left(X_{r_1}^G - X_{r_2}^G\right) \quad (46)$$

DE/current-to-best/2:

$$V_i^{G+1} = X_i^G + F.\left(X_{best}^G - X_i^G\right) + F.\left(X_{r_1}^G - X_{r_2}^G\right) + F.\left(X_{r_3}^G - X_{r_4}^G\right) \quad (47)$$

where X_{best}^G is the best solution vector achieved in generation G. Thereafter, based on current individual X_i^G and mutated individual V_i^{G+1}, trial individual T_i^{G+1} is formed as Eq. (37). Eventually, a greedy approach is done to select between trial vector T_i^{G+1} and individual X_i^G. It is worth noting that the feasibility of each mutated individual should be checked. After applying a mutation strategy, the elements in the first row of the mutated individual (chromosome) maybe exceed the admissible range [0, 1]. In such cases, the outlier elements should be corrected to zero if $\mathbb{a}_{1,j} < 0$, or they should be corrected to one if $\mathbb{a}_{1,j} > 1$.

4.2.4. Evaluation and selection of the parameters

In each generation of the algorithm, the setting of parameters F and CR as well as the selecting of a mutation strategy of each individual are performed based on a self-adaptive manner, which was introduced by Fan and Yan (2015a). In this approach, the triplex $\left(F_i^G, CR_i^G, MStr_i^G\right)$ as a vector of control parameters and mutation strategy ($MStr$) is associated with each individual X_i^G. The procedure of adaptation is summarized below.

Self-adaptive control of the parameters F and CR:

For a minimization problem, the gap between each mutated individual and the worst solution in the generation G is calculated by:

$$\Delta f_i^G = f_{max} - f\left(V_i^G\right) \qquad \forall\, i = 1, 2, \dots, NP \quad (48)$$

where f_{max} is the objective function value for the worst mutated individual (i.e., $f_{max} = \max_i\left(f\left(V_i^G\right)\right)$). The weighted value of each individual (w_i^G) can be calculated by:

$$w_i^G = \frac{\Delta f_i^G}{\sum_{i'=1}^{NP} \Delta f_{i'}^G} \quad (49)$$

Thereafter, the average value of mutation ($\bar{F}_w^G$) and crossover ($\overline{CR}_w^G$) control parameters in each generation are respectively calculated as follows.

$$\bar{F}_w^G = \sum_{i=1}^{NP} w_i^G \times F_i^G \tag{50}$$

$$\overline{CR}_w^G = \sum_{i=1}^{NP} w_i^G \times CR_i^G \tag{51}$$

Eventually, the mutation and crossover control parameters for the next generation are respectively updated as Eq. (52) and Eq. (53), where N is the normal distribution function, and σ_G is the standard deviation associated with generation G (i.e., $\sigma_G = 0.8 - 0.45(1 - (G/G_{max})^2)$) (Fan and Yan; 2015a).

$$F_i^{G+1} = N(\bar{F}_w^G, \sigma_G) \tag{52}$$

$$CR_i^{G+1} = N(\overline{CR}_w^G, \sigma_G) \tag{53}$$

Self-adaptive control of the mutation strategies:

For the first phase of generations, there is only one mutation strategy for each individual (i.e., $MStr_i^G = rand/1$ for $G \leq G_s$)). For the second phase of generations ($G_s < G \leq G_{max}$), we need to select a mutation strategy for each individual among six candidate strategies. At generation $G_s + 1$, we create a fair condition for all strategies. Therefore, at this generation, the number of individuals, which are mutated based on each of the six strategies is equal to $NP/6$. Thereafter, the self-adaptive procedure controls the number of mutation strategies which are used by the individuals in each generation; however, the selection of the strategies by each individual is performed randomly.

Due to the strength of the *NM-based-rand/3* mutation strategy in performing the exploitation tasks and to avoid local optimums, we consider a more conservative approach to control the number of individuals, which select this mutation strategy. Hence, a set of other five mutation strategies (i.e., $STR = \{rand/1, rand/2, best/2, current - to - best/1, \text{current} - \text{to} - \text{best}/2\}$) is utilized to explain the formulation. Moreover, $\mathcal{N}_{str}^G$ is defined as the set of individuals, which use the mutation strategy $str \in STR$ at the generation G (i.e., $\mathcal{N}_{str}^G = \{i | MStr_i^G = str\}$). Besides, v_{str}^G represents the number of members of the set $\mathcal{N}_{str}^G$ (i.e., $v_{str}^G = |\mathcal{N}_{str}^G|$). To determine the number of each mutation strategy, which should be used in the next generation, a trial vector $\hat{v}^G$ is computed as Eq. (54), and eventually, the values of v_{str}^{G+1} are computed based on Eq. (55).

$$\hat{v}_{str}^G = \text{round}\left(\frac{5}{6} NP \times \frac{\sum_{i \in \mathcal{N}_{str}^G} \Delta f_i^G}{\sum_{i=1}^{NP} \Delta f_i^G}\right) \qquad \forall\, str \in STR \tag{54}$$

$$v_{str}^{G+1} = \begin{cases} v_{str}^{G} + 1 & \text{if} \quad \hat{v}_{str}^{G} > v_{str}^{G} \\ v_{str}^{G} - 1 & \text{if} \quad \hat{v}_{str}^{G} < v_{str}^{G} \\ v_{str}^{G} & \text{otherwise} \end{cases} \qquad \forall\, str \in STR \tag{55}$$

The rest of individuals use the *NM-based-rand/3* mutation strategy in the next generation, hence the number of these individuals can be determined by:

$$v_{NM-based}^{G+1} = NP - \sum_{str \in STR} v_{str}^{G+1} \tag{56}$$

To self-adaptive control of the parameters of the NM-based mutation strategy, the weighted average of mutation scale factor ($\bar{F}_w^G$) is utilized. Hence, we consider the parameters as $\alpha_r = \alpha_e = \bar{F}_w^G$ and $\alpha_c = \alpha_s = \frac{1}{2}\bar{F}_w^G$.

5. Computational Experiments

The performance of the two proposed formulations of FJSSP-CDST and the proposed solving approach (SADE-NMMS) are tested in this section. Hence, two experiments are conducted to perform the evaluations. In the first experiment, the two proposed formulations are compared, then the performance of SADE-NMMS is compared with the objective values obtained by the MILP models and three different meta-heuristic algorithms. The MILP models are implemented in GAMS 24.1.3 and solved using the solver CPLEX. Moreover, the meta-heuristics are coded on MATLAB 2017. All test instances are performed on a computer with a 2.8 GHz Intel CPU and with 4 GB of installed memory.

5.1. Random instance generation

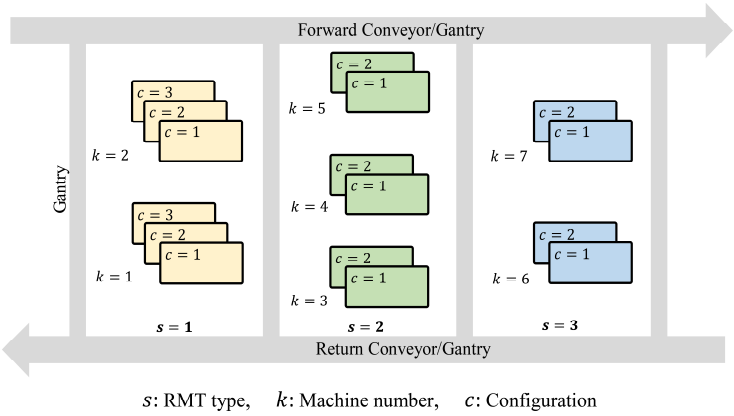
To do the experiments, two sets of random instance problems are generated. The size of each instance problem is dependent to the size of the shop floor, which is related to the number of RMTs ($|K|$) and the total number of machine configurations ($\sum c_k$), as well as the number of jobs ($|\mathcal{J}|$) and the total number of operations ($\sum n_i$). For the first set, two experiments are designed. At the first experiment, the MILP models are compared in term of computational complexities. At the second experiment, the best solution obtained by the MILP models in the previous experiment is utilized to validate the meta-heuristic algorithms. Finally, the performance of the meta-heuristics is tested using the second set, which includes some instance problems with larger sizes.

In the first set, we generate 12 small instances (Subset-A) and 16 medium instances (Subset-B). The considered levels are presented in Table 2. To more illustration, a shop floor with seven RMTs and 16 machine configurations (K7C16) is represented in Fig. 9. Also, the data for every one of the different instances includes the processing time of operations (p_{ijkc}) and the configuration-dependent setup times ($ST_{c_1,c_2,k}$). The processing times are

generated based on a uniform distribution between 40 and 100, and the setup times are generated based on a uniform distribution between 75 and 150. In the second set, we considered three different levels for the number of facilities in the shop floor. Thereafter, in each level, 30 random instance problems are generated. Thus, there are 90 large instances in the second set.

Table 2. Parameter levels for data generation

	First set	Second set
No. of machine configurations ($\|K\|, \sum c_k$)	A. (2, 4); (2, 5); (2, 6) B. (3, 7); (7, 16)	(10, 20); (10, 30); (20, 50)
No. of job operations ($\|\mathcal{J}\|, \sum n_i$)	A. (2, 8); (2, 15); (3, 10); (3, 20) B. $\|\mathcal{J}\| = [3-10], n_i = rand(4-12)$	$\|\mathcal{J}\| = rand(10-20)$ $n_i = rand(5-15)$



s: RMT type, k: Machine number, c: Configuration

Fig. 9. Shop floor with seven RMTs and 16 machine configurations (K7C16)

5.2. Models evaluation

In this section, two proposed MILP models are compared based on computational complexities and size, as two frequently used performance measures (Naderi and Azab, 2014). Table 4 represents the number of constraints and binary variables of different instance sizes. The position-based model needs fewer constraints than the sequence-based model, but it needs more binary variables. Both the number of constraints and binary variables are effective on the computational complexity. Hence, to compare the computational complexity of the proposed models, 28 instances, including 12 small instances (Subset-A) and 16 medium instances (Subset-B), are solved with a maximum time limit of 3600 seconds using CPLEX solver in GAMS 24.1.3 software. The results are presented in Table 5. The SB model optimally solves all 12 small instances, while the PB model solves nine of them. Indeed, the PB model cannot even optimally solve the small instances with 20 operations. Out of 16 instances with a medium size in the Subset-B, the SB and PB models optimally solve only four and two instances, respectively. Moreover, the average optimality

gap of the SB and PB models is 18.07% and 39.09%, respectively. Considering the computational complexity, the SB model outperforms the PB model.

Table 4. Comparison of the position- and sequence-based models

Problem size				#Constraints		#Binary variables	
$\|K\|$	$\sum c_k$	$\|\mathcal{J}\|$	$\sum n_i$	SBM	PBM	SBM	PBM
2	5	3	10	2'414	654	140	584
2	5	3	20	10'024	2'339	480	2'374
3	7	3	20	13'104	3'068	520	3'098
3	7	10	51	87'170	17'773	2'907	19'795
7	16	3	20	29'244	6'977	700	7'052
7	16	10	51	194'729	40'221	3'366	44'317

Table 5. Results of the position- and sequence-based models

#Instance	$\|\mathcal{J}\|$	$\sum n_i$	PB model		SB model		LB	
			C_{max}	Time (sec)/ opt. gap (%), Best bound	C_{max}	Time (sec)/ opt. gap (%), Best bound	C_{max}	Time (sec)
K2C4Ins01	2	8	**305***	0.515	**305***	0.218	277	0.14
K2C4Ins02	2	15	**620***	136.046	**620***	1	537	0.047
K2C4Ins03	3	10	**532***	7.5	**532***	0.578	365	0.078
K2C4Ins04	3	20	866	11.16%, 769	**860***	14.765	659	0.093
K2C5Ins01	2	8	**555***	0.562	**555***	0.234	311	0.094
K2C5Ins02	2	15	**908***	681.108	**908***	0.688	555	0.188
K2C5Ins03	3	10	**531***	3.812	**531***	0.625	403	0.063
K2C5Ins04	3	20	958	18.4%, 781	**872***	13.578	652	0.109
K2C6Ins01	2	8	**585***	1.282	**585***	0.281	392	0.032
K2C6Ins02	2	15	**1000***	160.47	**1000***	0.703	530	0.062
K2C6Ins03	3	10	**496***	10.343	**496***	0.672	356	0.125
K2C6Ins04	3	20	1226	19%, 993	**1155***	25.25	734	0.156
K3C7Ins01	3	20	613	28.22%, 467	**559***	15.937	436	0.078
K3C7Ins02	4	30	1035	40.77%, 613	**900**	27.97%, 648	658	0.156
K3C7Ins03	5	29	1182	59.73%, 476	**762**	32.92%, 511	598	0.094
K3C7Ins04	6	47	3098	78.44%, 668	**1379**	49.56%, 695	994	0.214
K3C7Ins05	7	59	5678	88.96%, 627	**2016**	67.61%, 653	1200	0.203
K3C7Ins06	8	61	5821	88.56%, 666	**1978**	64.18%, 708	1302	0.328
K3C7Ins07	9	71	8512	92.2%, 664	**2203**	68.5%, 694	1515	0.329
K3C7Ins08	10	51	3823	85.87%, 540	**1566**	61.24%, 607	1115	0.203
K7C16Ins01	3	20	**436***	201.922	**436***	5.109	436	0.047
K7C16Ins02	4	30	997	38.52%, 613	**613***	65.203	613	0.141
K7C16Ins03	5	29	**476***	2925.78	**476***	55.954	476	0.078
K7C16Ins04	6	47	3340	80%, 668	**725**	7.86%, 668	668	0.094
K7C16Ins05	7	59	6654	90.58%, 627	**933**	32.8%, 627	627	0.313
K7C16Ins06	8	61	7241	90.8%, 666	**921**	27.69%, 666	666	0.297
K7C16Ins07	9	71	8603	92.28%, 664	**1136**	41.55%, 664	664	10.922
K7C16Ins08	10	51	5545	91.13%, 492	**711**	24.05%, 540	540	0.125

* Optimum value

In addition to the comparison of two MILP models, Table 5 shows the value of the LBs calculated based on the model presented in Eqs. (27) – (35). Comparing the best bounds achieved by the implementation of the MILP models after 3600 seconds, some better values of LBs are extracted for seven instances (the underlined values in column eight of Table 5).

Generally, the experiments show that at least a value and the estimated best bounds achieved by the MILP models are extracted for the LB of each instance.

5.3. Evaluations of the proposed solving approach

To test the performance of SADE-NMMS, three different variants of DE algorithms are considered. These variants contain the classic DE (Storn and Price, 1997), the DE algorithm with self-adaptive mutation strategy and control parameters (SSCPDE) (Fan and Yan, 2015a), and the hybrid Nelder–Mead simplex search and DE algorithm (NMDE) (Wang et al. 2011). We utilize SSCPDE and NMDE in our experiments to evaluate the effect of our approach in the hybridization of NM simplex search and self-adaptation strategy on the DE algorithm. Moreover, to have a comparison with other meta-heuristics, we use a self-adaptive Cuckoo Optimization Algorithm (SA-COA), which has recently been developed in the related literature of a flexible job-shop scheduling problem by Abdollahzadeh-Sangroudi and Ranjbar-Bourani (2019). Actually, the classic COA was first introduced by Rajabioun (2011) inspired by the immigration and egg laying behavior of Cuckoo, a special kind of bird that lives in the nature. Recently, Abdollahzadeh-Sangroudi and Ranjbar-Bourani (2019) improved it to solve a kind of a flexible job-shop scheduling problem. Because of the existing some similarities between the structures of the habitat encoding used in their problem and the introduced encoding method in this paper, we accommodate the proposed SA-COA to solve the proposed problem as well.

On the other hand, each algorithm has some parameters, which need to be accurately calibrated to guarantee the best performance. To tune the parameters of the DE and NMDE algorithms, we utilize the Taguchi method that is a practical approach for parameter tuning of meta-heuristics used in the literature (Akbari-Jafarabadi et al. 2015; Parvasi et al. 2017; Akbari-Jafarabadi et al. 2017; Sahebjamnia et al. 2018). The results are summarized in Table 6. Although the SADE-NMMS and SSCPDE algorithms use a self-adaptive manner to control their parameters, the number of their population (NP) should be adjusted. We test four different levels of NP including 60, 90, 120 and 150 for each algorithm. In this experiment, we consider the medium-sized instances (Subset-B) as the test problems. The analysis of variance (ANOVA) test is used to analyze the results. The means plot and least significant difference (LSD) intervals for different levels of NP are represented in Fig. 10. As can be seen, $NP = 120$ provides the best results for both algorithms. To determine the value of parameters in the SA-COA, Abdollahzadeh-Sangroudi and Ranjbar-Bourani (2019) proposed an initial value for each parameter, which could be controlled in a self-adaptive manner during the solving approach. Also, we utilize their proposed method in our current paper.

Moreover, we consider a stopping condition based on a CPU time fixed to $\max\left(\left(\sum c_k\right) \times \left(\sum n_i\right)^2, 1000\right)$ milliseconds for each instance problem when it is solved by each of the algorithms. Run time limitation is an applicable and flexible measure to show the searching strength of the algorithms within a fixed time horizon (Roshanaei et al. 2009). Also, in this

research to compare the algorithms, we use the relative percentage deviation (RPD) as a common performance measure (Mahmoodjanloo et al. 2016; Fathollahi-Fard et al. 2019). It can be calculated by:

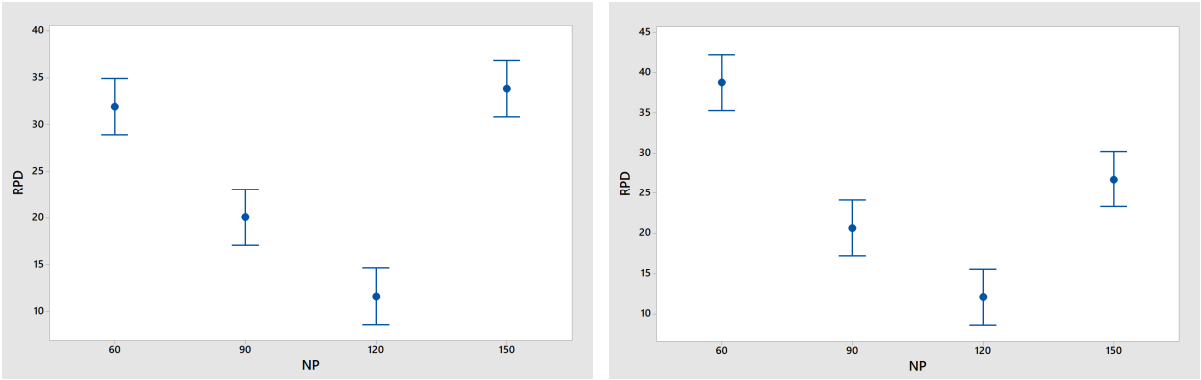
$$RPD = \frac{Alg\ - Min}{Min} \times 100 \tag{57}$$

where Alg is makespan of an instance obtained for a given algorithm, and Min is the best makespan obtained for each instance.

Table 6. Selected value of parameters for the DE and NMDE algorithms.

Algorithm	Parameters
DE	$NP = 90, F = 1, CR = 0.2$
NMDE	$NP = 60, F = 0.9, CR = 0.3$
	$\alpha_r = 1, \alpha_e = 1.5, \alpha_c = 0.5\ , \alpha_s = 0.5, \mathbb{Q} = 3$ *

* $\mathbb{Q}$ is the number of top individuals used to calculate the initial simplex centroid for the NM method



(a) SADE-NMMS (b) SSCPDE

Fig. 10. Means plot and LSD intervals (at the 95% confidence level) for the different levels of the population number (*NP*) of the SADE-NMMS and SSCPDE algorithms.

To evaluate the general performance of the four abovementioned algorithms, we use the first data set and compare the results with the objective values obtained by the sequence-based model (SBM). In this experiment, each instance problem is solved 20 times by each of the four algorithms. The results (including the best-obtained makespan, the mean value, and the standard deviation of each instance problem) are presented in Table 7. We bold the values, which are equal or less than the values obtained by the SBM.

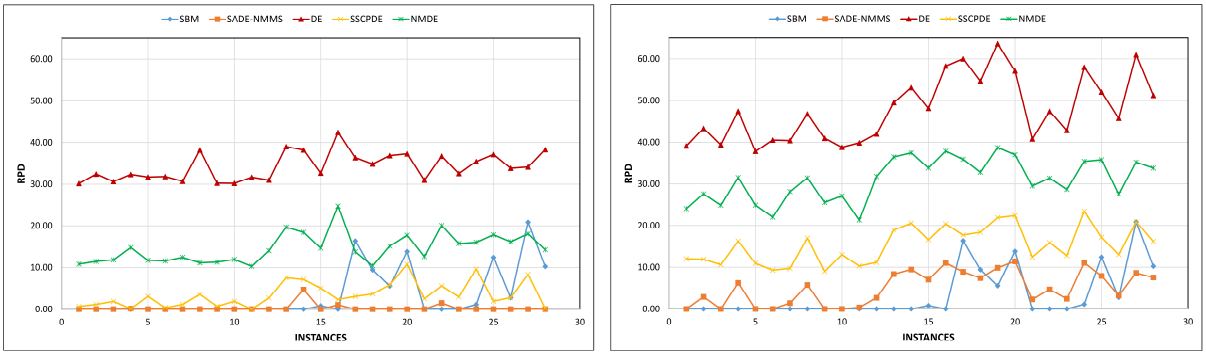
To better understand the performance of algorithms, the minimum and mean RPD values of the objective functions are separately represented in Fig 11. As can be seen, SADE-NMMS performs the best among the four algorithms. Among the 16 instances, whose optimum makespan is obtained by solving the SB model, SADE-NMMS can obtain the optimum solutions in 14 instances. Moreover, in ten items out of the rest of the instances, including 12 instance problems, which the CPLEX solver cannot obtain the optimum solution in a time

limit of 3600 seconds, SADE-NMMS obtained a better solution in a reasonable time (e.g., a solution of K3C7Ins05 with $C_{max} = 1733$ obtained by SADE-NMMS in 24.4 seconds while the CPLEX obtained a solution with $C_{max} = 2016$ in 3600 seconds).

In addition to the SADE-NMMS algorithm, SSCPDE also performs at an acceptable level, and it has the nearest performance to SADE-NMMS. Hence, we utilize the second data set to compare more precisely the performance of the tested algorithms. Also, the performance of the SA-COA is compared in this experiment. The ANOVA test is used to analyze the results in the experiment. As can be seen in Fig 12, the SADE-NMMS outperforms the other algorithms. Among the tested algorithms, the classic DE has the worst performance. This can be referred to the weaknesses of its mutation strategy in the exploitation phase of the searching process.

Table 7. Computational comparison of the algorithms on small and medium instance problems of the first data set.

#Ins.	MILP(SBM) C_{max}	SADE-NMMS			DE			SSCPDE			NMDE		
		Min	Ave.	St. Dev.	Min	Ave.	St. Dev.	Min	Ave.	St. Dev.	Min	Ave.	St. Dev.
K2C4Ins01	305*	**305**	**305**	0.0	397	424.3	14.7	307	341.7	15.7	338	378.4	24.9
K2C4Ins02	620*	**620**	637.8	27.6	821	888.5	48.2	627	693.8	49.7	691	791.1	60.9
K2C4Ins03	532*	**532**	**532**	0.0	695	741.6	35.2	542	588.4	26.7	595	664.4	43.2
K2C4Ins04	860*	861	913.5	41.2	1138	1268	86.8	**861**	1000	65.8	987	1131	100.5
K2C5Ins01	555*	**555**	**555**	0.0	731	764.9	25.1	572	615.7	27.1	620	693.1	49.0
K2C5Ins02	908*	**908**	**908**	0.0	1197	1276	42.5	911	992.3	51.5	1013	1108	73.8
K2C5Ins03	531*	**531**	538.5	21.7	694	745.6	37.7	537	582.6	35.1	597	680.5	51.9
K2C5Ins04	872*	**872**	921.9	24.0	1205	1280	52.7	903	1020	55.8	969	1146	65.4
K2C6Ins01	585*	**585**	**585**	0.0	762	824.5	28	589	637.7	29.5	651	734.5	57.1
K2C6Ins02	1000*	**1000**	**1000**	0.0	1302	1388	67	1019	1130	54.9	1119	1271	77.3
K2C6Ins03	496*	**496**	497.8	5.4	653	693.7	31.9	**496**	547	28.5	547	601.9	43.5
K2C6Ins04	1155*	**1155**	1186	10.1	1513	1640	61.8	1186	1284	82.9	1317	1522	116.3
K3C7Ins01	559*	**559**	605.6	25.2	777	835.8	44.4	601	664.9	38.9	669	762.9	55.5
K3C7Ins02	900	942	985.1	36.7	1244	1378	73.3	964	1085	74.6	1066	1237	110.8
K3C7Ins03	762	**756**	809.1	27.4	1003	1120	58.1	794	881.6	58.3	867	1012	88.9
K3C7Ins04	1379	1393	1531	74.6	1964	2183	164.2	1411	1660	118.6	1720	1901	113.5
K3C7Ins05	2016	**1733**	**1887**	88.1	2363	2773	251.6	**1786**	2040	152.0	1971	2355	214.0
K3C7Ins06	1978	**1808**	**1942**	85.5	2438	2797	241	**1874**	2141	126.7	1996	2401	218.9
K3C7Ins07	2203	**2087**	2293	123.0	2856	3415	281.1	2209	2544x	170.6	2403	2895	263.9
K3C7Ins08	1566	**1376**	**1533**	73.8	1889	2162	129.3	**1524**	1684	105.9	1620	1885	191.2
K7C16Ins01	436*	**436**	446	9.4	571	613.9	26.5	447	490.3	28.8	491	564.7	32.5
K7C16Ins02	613*	622	641.6	9.5	838	903.6	37.4	647	710.4	38.7	736	805.3	45.9
K7C16Ins03	476*	**476**	487.6	8.6	631	680	35.6	490	536.9	28.9	551	612.5	37.5
K7C16Ins04	725	**717**	796.2	32.5	971	1133	68	786	885	57.0	832	970.9	81.3
K7C16Ins05	933	**830**	**895.7**	26.7	1138	1262	60.6	**846**	972.3	47.6	978	1127	85.0
K7C16Ins06	921	**896**	925.1	20.8	1199	1306	65.4	**921**	1013	50.7	1041	1143	73.2
K7C16Ins07	1136	**940**	**1021**	54.1	1261	1513	134.1	**1018**	1136	72.4	1110	1271	119.1
K7C16Ins08	711	**645**	**693.5**	25.7	892	975.5	44.4	**646**	749.4	46.9	737	863	74.4



(a) Minimum values (b) Mean values

Fig. 11. Minimum and mean RPD values of different algorithms over 20 independent runs on 28 test instance problems of the first data set.

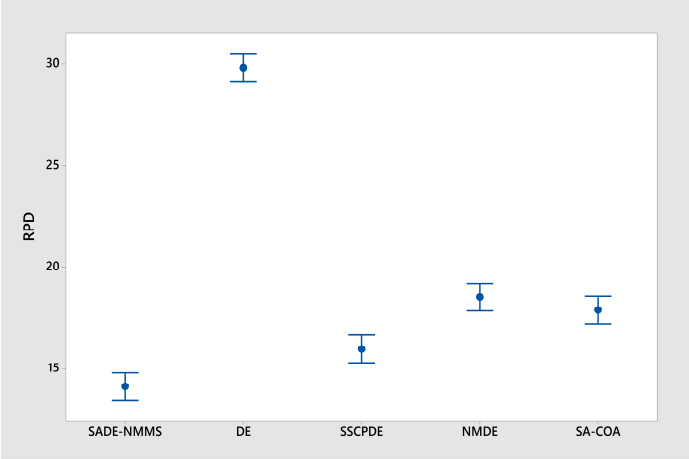


Fig. 12. Means plot and LSD intervals (at the 95% confidence level) for the tested algorithms on 90 instance problems of the second data set.

5.4. Sensitivity analysis

A significant observation in our experiments is the impact of reconfiguration ability of the machines on the scheduling decisions. This ability maybe leads to several changes in the configurations of a machine to obtain a feasible schedule. To show this ability, an example is illustrated in Fig. 13. In this Gantt chart, the best resultant schedule of the instance *K7C16Ins05* with $C_{max} = 830$ obtained by the SADE-NMMS algorithm is presented. In each box, two items are noted (i.e., the job's number and the configuration's number, in which the machine should perform that job). For example, *(J5, C2)* on the first box of *Machine-1* shows that *Job-5* should be performed on the second configuration of *Machine-1*. Besides, *Machine-1* performs the first and the second operations of *Job-5* on its second configuration. Thereafter, the machine should be reconfigured from the second to the third configuration with setup time $ST(C2, C3) = 150$ to perform some other operations. As can be seen, some machines need to be reconfigured one or more times in the schedule, and some others need no reconfiguration (e.g., *Machine-5*).

Regarding the impact of reconfiguration activities and the related setup times on the scheduling decisions, several analyses are done. For this purpose, we change the configuration-dependent setup times of several instances in a predefined range. In these experiments, we define $ST^{new}_{c_1,c_2,k} = \omega \times ST_{c_1,c_2,k}$ while ω is a constant coefficient. As an example, the trend of changing in makespan and the number of machine reconfigurations are presented in Fig. 14 for $0 \leq \omega \leq 4$. As expected, makespan has a non-decreasing behavior concerning an increase in the setup times, though for the larger instances the behavior is much more non-linear. On the other hand, changing the number of machine reconfigurations is more complex. As can be seen in Fig. 14(b), the diagrams have different trends in response to the variations of the setup times.

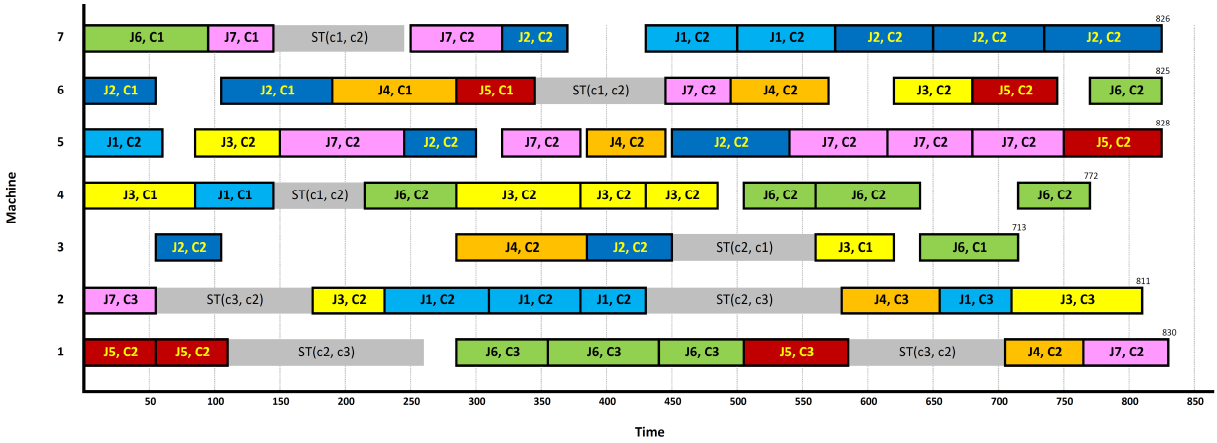
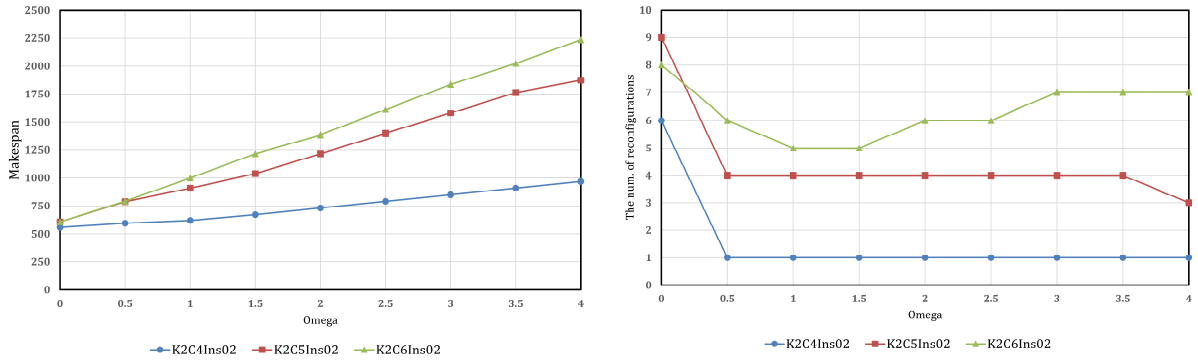


Fig. 13. Best resultant schedule of *K7C16Ins05* with $C_{max} = 830$ obtained by the SADE-NMMS algorithm.



(a) (b)

Fig. 14. Trend of changing in makespan and the number of machine reconfigurations based on variations in the setup times

To improve the searching process, a two-phase mechanism is utilized to balance between exploitation and exploration abilities of the proposed algorithm. In this mechanism, threshold value G_s is considered to change the searching policy, where $G_s = \eta \times G_{max}$ and $\eta \in [0, 1]$. In the first phase for $G \leq G_s$, we utilize the DE/rand/1 mutation strategy, which is

one of the most popular strategies in the literature that bears stronger exploration capabilities (Mallipeddi et al. 2011; Fan and Yan, 2015a; Fan and Yan, 2015b). Thereafter, a self-adaptive policy is used to automatically guide the searching process.

Now, we aim to evaluate the effectiveness of the two-phase searching strategy. Hence, we design another experiment to show the performance of the SADE-NMMS algorithm with different values of parameter η. In this experiment, the algorithm is used to solve some selected instances of Subset-B for $\eta = 0, 0.2, 0.4$ and 1. It is worth noting that the algorithm with $\eta = 0$ is a full self-adaptive version. On the other hand, the algorithm with $\eta = 1$ can be considered as a classic DE with one mutation strategy. The means plot and Least Significant Difference (LSD) intervals for different levels of η are represented in Fig. 15. As can be seen, $\eta = 0.2$ provides the best results for the algorithm. Moreover, it means that the designed two-phase policy has a positive effect on the searching process of the algorithm. Also, Fig. 16 shows the trend of optimization for the proposed algorithm with different values of η on two selected instances from Subset-B.

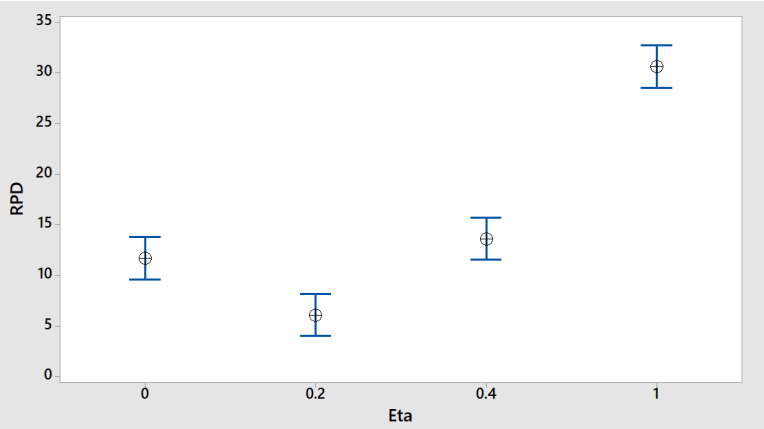
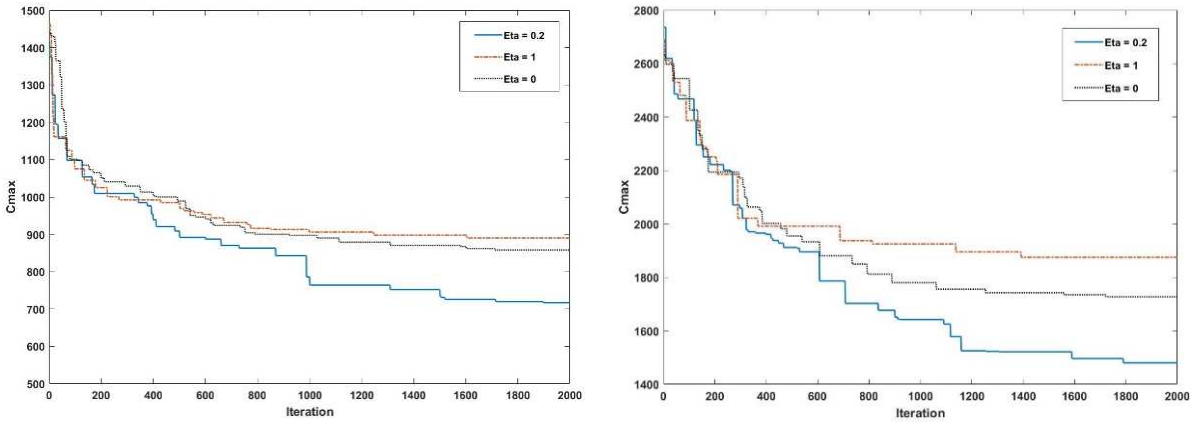


Fig. 15. Means plot and LSD intervals (at the 95% confidence level) for different values of η in algorithm SADE-NMMS.



(a) K7C16Ins08 (b) K3C7Ins08

Fig. 16. Effect of parameter η on the searching process of the algorithm SADE-NMMS

6. Conclusions

Reconfigurable machine tools (RMTs) have been developed to benefit from using several different machines that share many costly and common modules while being rarely used at the same time. They could obtain different configurations to satisfy manufacturing requirements. Hence, RMTs have a high potential to obtain both cost-effectiveness and responsiveness as the main objectives of market competition. Considering the complexity of scheduling in production systems, which utilize RMTs, some specialized models and algorithms should be developed. In this paper, the scheduling decisions in a shop-floor with RMTs, named the FJSSP with configuration-dependent setup times have been studied. At first, two different mathematical models with the position- and sequence-based decision variables have been formulated to minimize the completion time of the jobs (i.e., makespan). Moreover, a mathematical formulation has also been developed to calculate the lower bound of makespan. Thereafter, we tried to optimally solve the small- and medium-sized instances using each of the two models and CPLEX solver. The experiments showed that the sequence-based model outperformed the position-based model in all 28 instance problems. However, even the sequence-based model could not to optimally solve most of the medium-size problems (it only optimally solved three instances out of 16). Hence, regarding the computational complexity of the models, we utilized a self-adaptive DE algorithm and enhanced its effectiveness by introducing a new mutation strategy based on a searching approach hired from the Nelder-Mead method. The performance of the proposed method, named SADE-NMMS, and three other variants of the DE algorithm were first validated by comparison with the results of the sequence-based model for small- and medium-sized problems. Thereafter, another data set including larger-sized problems has been utilized to compare more precisely the performance of the tested algorithms. The ANOVA test was used to analyze the results in the experiment. It turned out that the SADE-NMMS outperforms the other algorithms.

For future studies, some of the real-world considerations (e.g., the uncertainty in configuration-dependent setup times or the effect of reliability on different machine configurations) can be discussed in the model. Besides, different machine configurations in addition to the different production rate maybe have different cost or quality to perform the operations. Hence, it will be interesting to develop some multi-objective models in this area. Finally, the proposed mutation strategy based on the Nelder-Mead method has a high potential to be utilized in other variants of the DE algorithm to improve their performance.

CRediT authorship contribution statement

Mehdi Mahmoodjanloo: Conceptualization, Writing - original draft, Software. **Reza Tavakkoli-Moghaddam:** Methodology, Supervision, Writing – review & editing. **Armand**

Baboli: Project administration, Resources, Visualization. **Ali Bozorgi-Amiri:** Data curation, Validation.

Acknowledgments

The authors would like to thank the Editor-in-Chief, Associate Editor and anonymous reviewers for their valuable comments on this paper for the improvements.

References

Abdollahzadeh Sangroudi, H., & Ranjbar-Bourani, M. (2019). Solving a flexible job shop lot sizing problem with shared operations using a self-adaptive COA. *International Journal of Production Research*, Article in Press.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345-378.

Aguilar, A., Roman-Flores, A., & Huegel, J. C. (2013). Design, refinement, implementation and prototype testing of a reconfigurable lathe-mill. *Journal of Manufacturing Systems*, *32*(2), 364-371.

Akbari-Jafarabadi, M., Tavakkoli-Moghaddam, R., Mahmoodjanloo, M., & Rahimi, Y. (2015). A three-level mathematical model for an r-interdiction hierarchical facilities location problem. *Iranian Journal of Operations Research*, *6*(2), 58-72.

Akbari-Jafarabadi, M., Tavakkoli-Moghaddam, R., Mahmoodjanloo, M., & Rahimi, Y. (2017). A tri-level r-interdiction median model for a facility location problem under imminent attack. *Computers & Industrial Engineering*, *114*, 151-165.

Azab, A., & Naderi, B. (2015). Modelling the problem of production scheduling for reconfigurable manufacturing systems. *Procedia CIRP*, *33*, 76-80.

Azulay, H. (2014). A design methodology for reconfigurable milling machine tools and an implementation, *Ph.D. dissertation*, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4), 369-375.

Chelouah, R., & Siarry, P. (2005). A hybrid method combining continuous tabu search and Nelder–Mead simplex algorithms for the global optimization of multi-minima functions. *European Journal of Operational Research*, *161*(3), 636-654.

Das, S., & Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1), 4-31.

Ersal, T., Stein, J. L., & Louca, L. S. (2004, January). A modular modeling approach for the design of reconfigurable machine tools. In: *ASME 2004 International Mechanical*

Engineering Congress and Exposition (pp. 393-399). American Society of Mechanical Engineers, Anaheim, California, USA, 13-19 November 2004.

Fan, Q., & Yan, X. (2015a). Differential evolution algorithm with self-adaptive strategy and control parameters for P-xylene oxidation process optimization. *Soft Computing*, *19*(5), 1363-1391.

Fan, Q., & Yan, X. (2015b). Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies. *IEEE Transactions on Cybernetics*, *46*(1), 219-232.

Fathollahi-Fard, A. M., Ranjbar-Bourani, M., Cheikhrouhou, N., & Hajiaghaei-Keshteli, M. (2019). Novel modifications of social engineering optimizer to solve a truck scheduling problem in a cross-docking system. *Computers & Industrial Engineering*, 137, 103-118.

Gadalla, M., & Xue, D. (2017). Recent advances in research on reconfigurable machine tools: a literature review. *International Journal of Production Research*, *55*(5), 1440-1454.

Gao, Z., Xiao, T., & Fan, W. (2011). Hybrid differential evolution and Nelder–Mead algorithm with re-optimization. *Soft Computing*, *15*(3), 581-594.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research*, *1*(2), 117-129.

Gholipour-Kanani, Y., Tavakkoli-Moghaddam, R., Cheraghalizadeh, R., & Mahmoodjanloo, M. (2012). A new mathematical model for a multi-criteria group scheduling problem in a cms solved by a branch-and-bound method. *In Proceedings of the 2012 international conference on industrial engineering and operations management*.

Gu, X., & Koren, Y. (2018). Manufacturing system architecture for cost-effective mass-individualization. *Manufacturing Letters*, *16*, 44-48.

Hasan, F., Jain, P. K., & Kumar, D. (2013). Machine reconfigurability models using multi-attribute utility theory and power function approximation. *Procedia Engineering*, *64*, 1354-1363.

Jamili, A., Shafia, M. A., & Tavakkoli-Moghaddam, R. (2011). A hybrid algorithm based on particle swarm optimization and simulated annealing for a periodic job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *54*(1-4), 309-322.

Koren, Y., & Kota, S. (1999). *U.S. Patent No. 5,943,750*. Washington, DC: U.S. Patent and Trademark Office.

Mahmoodjanloo, M., Parvasi, S. P., & Ramezanian, R. (2016). A tri-level covering fortification model for facility protection against disturbance in *r*-interdiction median problem. *Computers & Industrial Engineering*, *102*, 219-232.

Mallipeddi, R., Suganthan, P. N., Pan, Q. K., & Tasgetiren, M. F. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, *11*(2), 1679-1696.

Menchaca-Mendez, A., & Coello, C. A. C. (2009, May). A new proposal to hybridize the nelder-mead method to a differential evolution algorithm for constrained optimization. In *2009 IEEE Congress on Evolutionary Computation* (pp. 2598-2605). IEEE, Trondheim, Norway, 18-21 May 2009.

Moghaddam, S. K., Houshmand, M., & Fatahi Valilai, O. (2018). Configuration design in scalable reconfigurable manufacturing systems (RMS): A case of single-product flow line (SPFL). *International Journal of Production Research*, *56*(11), 3932-3954.

Moghaddam, S. K., Houshmand, M., Saitou, K., & Fatahi Valilai, O. (2019). Configuration design of scalable reconfigurable manufacturing systems for part family. *International Journal of Production Research*, Article in Press.

Moraglio, A., & Johnson, C. G. (2010). Geometric generalization of the nelder-mead algorithm. In: *European Conference on Evolutionary Computation in Combinatorial Optimization* (pp. 190-201). Springer, Istanbul, Turkey, 7-9 April 2010.

Moravec, P., & Rudolf, P. (2018). Combination of a particle swarm optimization and Nelder–Mead algorithm in a diffuser shape optimization. In: *Advances in Hydroinformatics* (pp. 997-1012). Springer, Singapore.

Naderi, B., & Azab, A. (2014). Modeling and heuristics for scheduling of distributed job shops. *Expert Systems with Applications*, *41*(17), 7754-7763.

Nelder, J.A., Mead, R.A. (1965), A simplex method for function minimization. *Computer Journal*, 7, 308–313

Padayachee, J., & Bright, G. (2012). Modular machine tools: Design and barriers to industrial implementation. *Journal of Manufacturing Systems*, *31*(2), 92-102.

Parvasi, S. P., Mahmoodjanloo, M., & Setak, M. (2017). A bi-level school bus routing problem with bus stops selection and possibility of demand outsourcing. *Applied Soft Computing*, *61*, 222-238.

Pérez, R., Molina, A., & Ramírez-Cadena, M. (2014). Development of an integrated approach to the design of reconfigurable micro/mesoscale CNC machine tools. *Journal of Manufacturing Science and Engineering*, *136*(3), 031003.

Ponsich, A., Tapia, M. G. C., & Coello, C. A. C. (2009). Solving permutation problems with differential evolution: an application to the job shop scheduling problem. In *2009 Ninth International Conference on Intelligent Systems Design and Applications* (pp. 25-30). IEEE, Pisa, Italy, 30 November – 02 December 2009.

Ponsich, A., & Coello, C. A. C. (2013). A hybrid differential evolution - Tabu search algorithm for the solution of job-shop scheduling problems. *Applied Soft Computing*, *13*(1), 462-474.

Qin, AK, Huang, VL, & Suganthan, PN (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2), 398-417

Qin, J., Liu, Y., & Grosvenor, R. (2016). A categorical framework of manufacturing for industry 4.0 and beyond. *Procedia CIRP*, *52*, 173-178.

Rajabioun, R. (2011). Cuckoo optimization algorithm. *Applied soft computing*, 11(8), 5508-5518.

Roshanaei, V., Naderi, B., Jolai, F., & Khalili, M. (2009). A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. *Future Generation Computer Systems*, *25*(6), 654-661.

Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, *153*, 253-267.

Rohaninejad, M., Kheirkhah, A., Fattahi, P., & Vahedi-Nouri, B. (2015). A hybrid multi-objective genetic algorithm based on the ELECTRE method for a capacitated flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 77(1-4), 51-66.

Sahebjamnia, N., Fathollahi-Fard, A. M., & Hajiaghaei-Keshteli, M. (2018). Sustainable tire closed-loop supply chain network design: Hybrid metaheuristic algorithms for large-scale networks. *Journal of Cleaner Production*, *196*, 273-296.

Shen, L., Dauzère-Pérès, S., & Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, *265*(2), 503-516.

Storn, R., & Price, K. (1997). Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341-359.

Wang, L., Xu, Y., & Li, L. (2011). Parameter identification of chaotic systems by hybrid Nelder–Mead simplex search and differential evolution algorithm. *Expert Systems with Applications*, *38*(4), 3238-3245.

Wu, X., Liu, X., & Zhao, N. (2018). An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem. *Memetic Computing*, 1-21.

Wu, X. and Liu, X. (2018, August). An improved differential evolution algorithm for solving a distributed flexible job shop scheduling problem. In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)* (pp. 968-973). IEEE, Munich, Germany, 20-24 August 2018.

Yuan, Y., & Xu, H. (2013). Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering*, *65*(2), 246-260.

Yuen, S. Y., & Zhang, X. (2015). On composing an algorithm portfolio. *Memetic Computing*, *7*(3), 203-214.

Zhang, H., Yan, Q., Zhang, G., & Jiang, Z. (2016). A chaotic differential evolution algorithm for flexible job shop scheduling. In: *Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems* (pp. 79-88). Springer, Singapore.

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing*, *30*(4), 1809-1830.

Zhao, F., Shao, Z., Wang, J., & Zhang, C. (2016). A hybrid differential evolution and estimation of distribution algorithm based on neighborhood search for job shop scheduling problems. *International Journal of Production Research*, *54*(4), 1039-1060.