Distributed job-shop rescheduling problem considering reconfigurability of machines: a self-adaptive hybrid equilibrium optimiser

M. Mahmoodjanloo, R. Tavakkoli-Moghaddama, A. Baboli, A. Bozorgi-Amiri

Distributed job-shop rescheduling problem considering reconfigurability of machines: A self-adaptive hybrid equilibrium optimizer

M. Mahmoodjanloo [a,b], R. Tavakkoli-Moghaddam [a,*], A. Baboli [b], A. Bozorgi-Amiri [a]

[a] *School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran*
[b] *LIRIS Laboratory, UMR 5205 CNRS, INSA of Lyon, 69621 Villeurbanne cedex, France*

The recent trend of globalization of the economy has been accelerated thanks to emerging new communication technologies. This forces some companies to be adapted to rapidly changing market requirements utilizing a multi-factory production network. Job scheduling in such a distributed manufacturing system, is significantly complicated especially in the presence of dynamic events. Furthermore, production systems need to be flexible to timely react to the imposed changes. Hence, reconfigurable machine tools (RMTs) can be used as a resource for flexibility in manufacturing systems. This paper deals with a distributed job-shop rescheduling problem, in which the facilities benefit from reconfigurable machines. Firstly, the problem is mathematically formulated to minimize total weighted lateness in a static state. Then, the dynamic version is extent based on a designed conceptual framework of rescheduling module to update the current schedule. Since the problem is NP-hard, a self-adaptive hybrid equilibrium optimizer algorithm is proposed. The experiments show that the proposed EO algorithm is extremely efficient. Finally, a simulation-optimization model is developed to evaluate the performance of the manufacturing system facing stochastic arriving jobs. The obtained results show that the production system can be very flexible relying on its distributed facilities and reconfigurable machines.

Keywords: Distributed manufacturing, Reconfigurable machine tools, Dynamic scheduling, Cyber-physical shop floor, simulation-optimisation model; meta-heuristic algorithms.

1. Introduction

Over the last decades, due to the globalization of the economy and its requirements, manufacturing systems need to be more flexible to survive in such a competitive environment. Hence, some companies have started to utilize a network of

* Corresponding author.
E-mail address: tavakoli@ut.ac.ir (R. Tavakkoli-Moghadddam).

geographically dispersed facilities as a distributed manufacturing system. In such a system, manufacturers have the opportunity to become closer to both customers and suppliers. Therefore, they can be more responsive to market requirements because of some important advantages such as accessibility to raw materials and low-cost production resources (Naderi and Azab, 2014).

To comply with and to take more advantages of such distributed systems, the different parts need to be coordinated using information technology. This lets to disperse components to have a more effective performance and a high competitiveness level for other players. That is while capabilities of new emerging smart factories are enhancing supported by Industry 4.0 technologies, such as the internet of things (IoT), cyber-physical systems (CPS), and big data. The required real-time data can be provided from the distributed shop floors using Industry 4.0 technologies. These data can include the job-related events (e.g., new arriving jobs or delayed jobs), resource-related events (e.g., accessibility of machines and their current configurations), and possible limitations enforced by the logistics system (e.g., raw materials availability). The capabilities of a connected manufacturing system help companies make more dynamic decisions to adapt to a customer-oriented market (Zhang et al. 2019). Coordination can be important for the flexibility of a system; however, it is not all. In other words, the infrastructure of production needs to respond main requirements of this market as well (e.g., the high demand fluctuations or the rapid introduction of new products).

To cope with these issues, taking advantages of reconfigurability for manufacturing systems and tools has been considered in recent years. As one of the useful contributions in this field, a new class of production machines, reconfigurable machine tools (RMTs), have been introduced. An RMT is able to satisfy a wide range of manufacturing requirements relying on its flexible and modular structure (Mahmoodjanloo et al. 2020a). One of the most important benefits of developing RMTs is the use of several different machines, which share common and many costly modules while being rarely used simultaneously, can be prevented (Gadalla and Xue 2017). The capacity and functionality of a manufacturing system by relying on the abilities of such machines can be adjusted easily.

As production scheduling is one of the most important basics in the production management field, and it has a significant impact on both supply chain responsiveness and efficiency, in this paper, we aim to study a dynamic job-shop scheduling problem

in a distributed manufacturing system, in which each facility contains some reconfigurable machine tools to perform assigned operations.

This paper is organized as follows. Section 2 reviews the related literature. Section 3 presents a mathematical model for the static state of the problem and its extension for a dynamic environment. Section 4 proposes an extension of a new meta-heuristic to solve the problem efficiently. Section 5 conducts numerical experiments. Finally, Section 6 concludes the paper and suggests a few future research directions.

2. Literature review

A distributed scheduling problem (DSP) in multi-factories is one of the well-known optimization problems, which has motivated several studies especially in the last two decades (Chaouch et al. 2017a). In addition to academic research studies, the production scheduling in a distributed environment, as a challenging problem, is also interesting among real-world applications (Wilkinson et al. 1996; Wang et al. 2007). Generally, the problem contains two main decisions including 1) determining the most suitable factory (i.e., facility) for each job and 2) scheduling operations of the assigned jobs on the available machines in each facility (Jia et al. 2003). Hence, the problem has a high level of complexity especially when we face a job shop scheduling problem (JSSP) in the machine environment of each facility. On one hand, the growing role of globalized and distributed manufacturing, and on the other, the need to find efficient algorithms to solve such a complex problem, have motivated researchers to pay more attention in this area. Following up, we are going to review the used solution approaches and the utilized performance indicators in the related researches.

For the information flow structure, DSPs can be studied in a centralized or decentralized decision-making system. In a decentralized (multi-agent) approach, scheduling sub-problems can be independently solved in each facility by an associated local decision-maker, who may have conflicting objectives with other local agents. The overall system objectives can be achieved through predefined communication mechanisms (Toptal and Sabuncuoglu 2010). Even though such an approach is flexible, and it can overcome the complexity of DSP as well; there is not any guaranty about the optimality of the final solution. On the other hand, a centralized (single-agent) approach considers the overall scheduling problem to achieve the global optimum solution. Although due to the high level of complexity, the global optimum is

hardly achieved; however, developing an efficient algorithm can guaranty achieving a high-quality solution within reasonable computation time.

Only a minor part of the literature has dealt with the distributed job-shop scheduling problem (DJSP) with a centralized decision-making approach (Chaouch et al. 2017a). As one of the pioneer studies with successful implementation, the DJSP was studied by Jia et al. (2002) to facilitate collaboration among geographically distributed plants of multinational companies. The authors used the World Wide Web (WWW) technology to develop a web-based system. They utilized a genetic algorithm (GA) to solve the problem with the considerations of low cost and short makespan. Thereafter, Jia et al. (2003) developed a modified GA to solve the DJSP by introducing a two-step encoding method to assign and schedule the jobs. Naderi and Azab (2014) for the first time utilized mathematical programming to model the DJSP and proposed two different mathematical formulations, which used sequence- and position-based variables to minimize makespan. They also proposed six simple heuristics to solve the problem. Later, Naderi and Azab (2015) improved the sequence-based mathematical model and developed a hybrid simulated annealing (SA) algorithm using a greedy local search method.

Chaouch et al. (2017b) proposed an improved version of the classic ant system algorithm by introducing an elitism mechanism and using the job-facility assignment heuristic (i.e., introduced by Naderi and Azab (2014)), to solve the DJSP. Wu et al. (2017) studied the effect of using different encoding methods on the performance of the GA in solving the DJSP. They also introduced a new chromosome representation and two heuristic rules to represent/extract three decisions including (1) job-to-facility assignment, (2) operation-to-machine assignment, and (3) operation sequencing decision. Their research confirmed the effect of appropriate chromosome representations on the output of the used algorithms. Chaouch et al. (2019) used a hybrid ant colony algorithm combined with a local search to minimize the global makespan over all the factories in the DJSP.

While the above-mentioned studies considered a traditional JSSP in a multi-factory environment to minimize a time-related objective (e.g., makespan or total tardiness), some recent papers in this field have tried to consider other aspects of the problem too. Chang and Liu. (2017) developed a hybrid GA to minimize makespan in the distributed flexible job-shop scheduling problem (DFJSP), in which jobs could be performed on different machines in each facility. They used an encoding scheme based on a roulette

wheel method to randomly allocate the jobs to facilities and assign the operations to eligible associated machines. Lu et al. (2018) proposed a chromosome scheme to enhance the performance of the GA to minimize makespan in the DFJSP. In addition to makespan, some other criteria have been considered especially in multi-objective DJSPs. Li et al. (2018) studied a multi-objective DFJSP and proposed a hybrid Pareto-based tabu search algorithm to minimize four criteria including makespan, maximal workload, total workload, and earliness/tardiness (E/T).

Wu et al. (2019) developed a mixed-integer linear programming (MILP) model for a DFJSP and proposed a multi-objective hybrid differential evolution and simulated annealing algorithm to minimize the E/T and total cost. Jiang et al. (2020) proposed an effective modified multi-objective evolutionary algorithm with decomposition (MMOEA/D) to minimize energy consumption and makespan. While in all the above-mentioned studies, it is assumed that all operations of a job should be performed on the machines of the same facility, Luo et al. (2020) and Gong et al. (2020) assume that operations of a job can be transferred among different facilities taking into account the jobs' transfer time and transfer energy consumption. For an overview of more related studies on production planning and scheduling problems in multi-factory production networks as well as a more comprehensive classification, see Lohmer and Lasch (2020).

As can be observed in the reviewed papers, the studied approaches developed to solve the problem of static distributed scheduling are often impractical in real-world environments. Moreover, at the dawn of the fourth industrial revolution (Industry 4.0), smart and distributed production systems can be supported by new emerging technologies and paradigms such as IoT and CPS. Such technologies provide real-time data to make efficient decisions in dynamic distributed production systems. Hence, designing efficient methods and tools to utilize the provided data to make better decisions can be significant. In contrary to multi-factory issues, developing smart scheduling systems for single-factory applications has been considered recently. For example, Romero-Silva and Hernández (2019) studied the role of CPS in different manufacturing contexts to provide companies in order to carry out a better scheduling task. They found that especially production systems with un-certain demands and complex production processes can significantly benefit from implementing a CPS at their shop-floor levels. Tian et al. (2019) proposed a production information management system using the industrial IoT technology to tackle the dynamic FJSP

on a rolling horizon. In this paper, inspired by such concepts, we aim to develop a dynamic scheduling model to be applied in a distributed manufacturing environment.

On the other hand, when we consider the dynamic aspects of a production environment, paying attention to changeable customer needs in terms of both product mix and volume can be significant. To support such requirements, utilizing a suitable substrate for production is inevitable. Hence, the paradigm of reconfigurable manufacturing systems (RMSs) has been introduced to answer the rapid changes in the variety or volume of the market demand. Indeed, reconfigurability in production can be defined as the ability to transform the system to be adjusted according to new requirements (Yelles-Chaouche et al. 2020). A production system can obtain such ability thanks to the reconfigurability features in its different components and sub-systems such as machines, workforce mapping, line/facility layout, material handling, and transportation systems. Reconfigurable machines, as one of the most important resources, play a significant role in such manufacturing systems. Nonetheless, despite the special capabilities, utilizing these machines due to the high level of their flexibilities creates new operational challenges, especially in the scheduling issues.

Recently, Mahmoodjanloo et al. (2020a) presented a new variant of a job-shop scheduling problem with configuration-dependent setup times (CDST), which contains reconfigurable machine tools on the shop floor. They presented the mathematical formulations of a single-factory scheduling problem and a hybrid meta-heuristic namely self-adaptive differential evolution with Nelder-Mead mutation strategy (SADE-NM) to efficiently solve the problem. Later, they developed a mathematical model based on operation-position decision variables for a dynamic distributed scheduling problem considering RMTs in shop floors (Mahmoodjanloo et al. 2020b). However, their research needs further improvements, which we will proceed to do by presenting the following contributions:

- Firstly, proposing a new mathematical model based on operation-sequence decision variables for a dynamic distributed flexible job-shop scheduling problem with configuration-dependent setup times (DFJSP-CDST) and comparing two models (i.e., sequence- and position-based formulations) based on their computational performances.
- Secondly, improving the performance of a newly introduced meta-heuristic, namely Equilibrium Optimizer presented by Faramarzi et al. (2020), by adding

a self-adaptive mechanism to effectively control its parameters and embedding a local search based on the VNS algorithm.

- Thirdly, developing an encoding scheme for the considered problem and applying the presented self-adaptive algorithm to solve real-world instances of reasonable large sizes.
- Finally, utilizing a simulation-optimization approach to evaluate the impact of using new modules for RMTs on the performance of a production system in a dynamic environment when demand is stochastic with Poisson distribution.

3. Problem statement and mathematical formulation

3.1. Problem definition

There is a distributed manufacturing system including several facilities, which have already been deployed in different geographical areas. Each facility $f \in F$ contains a set of reconfigurable machine tools $k \in K_f$ to process the assigned jobs. For each RMT, there is a set of possible configurations $C_{k,f}$ that can be obtained using appropriate modules. The process of changing the modules needs a setup time $ST^{f}_{c_1,c_2,k}$ that is dependent on two consecutive configurations c_1 and c_2, where $c_1 \neq c_2$. Actually, on each machine $k \in K_f$, to obtain a new configuration $c_2 \in C_{k,f}$ from the initial configuration $c_1 \in C_{k,f}$, it is needed to remove some old modules and/or add some new ones. It is supposed that an RMT can perform one or more operations in each configuration. The new configuration of the machine can let us carry out some new operations or fulfill some of the old ones at a different rate.

There is a set $\mathcal{J}$ of n jobs, where each one should be performed in one of the existing facilities. Each job $i \in \mathcal{J}$ has a set of n_i operations with a prespecified order (e.g., $O_{i,1} \rightarrow O_{i,2} \rightarrow \cdots \rightarrow O_{i,n_i}$). It is supposed that operation O_{ij} can be performed at least on one configuration of the existing machines in each facility. It is worth noting that, no setup is needed to carry out the operations in a machine configuration, while to switch to a different configuration, the machine needs a setup. The main decisions that can be made in the problem environment contain allocating each job to a facility and scheduling the assigned jobs on the machines in the associated facility.

The objective is to minimize total weighted lateness. Moreover, the scheduling environment can be classified into two main classes including static (offline) and dynamic (online) scheduling. Herein, firstly, a static mathematical model is developed

based on a sequence-based formulation. Afterward, the proposed model is extended for a dynamic situation where there are some jobs in each facility to be processed according to an incumbent schedule. Then the current schedule should be updated by arriving some new jobs. To perform such schedule updating tasks, we suppose a variable-order rescheduling strategy, in which all unprocessed operations can be rescheduled after assigning the new arriving jobs to the facilities, but to consider the stability of the current schedule as much as possible, deviation of the completion times in the new schedule will be controlled by adding a penalty term to objective function of the dynamic model.

3.2. Static scheduling model

In this subsection, we present a static mathematical model formulation, in which there are several jobs to be scheduled in the empty facilities.

Sets and indices:

F	Set of facilities, where facility index $f \in F$
K_f	Set of machines in the facility $f \in F$, where machine index $k \in K_f$
$C_{k,f}$	Set of configurations of machine $k \in K_f$, where configuration index $c \in C_{k,f}$
$\mathcal{J}$	Set of jobs, where job index $i \in \mathcal{J}$
N_i	Set of all operations of job $i \in \mathcal{J}$, where operation j of job i is denoted by O_{ij}

Parameters:

PT^f_{ijkc}	Processing time of operation O_{ij} on machine-configuration kc in facility f
R^f_{ijkc}	Binary parameter. 1 if operation O_{ij} can be processed on machine-configuration kc in facility f; 0, otherwise.
TT^f_i	Transportation time to the related customer when job i be sent from facility f
$ST^f_{c_1,c_2,k}$	Configuration-dependent setup time when the configuration is changed from c_1 to c_2 (i.e., $c_1 \neq c_2$ and $c_1, c_2 \in C_{k,f}$) on machine k in the facility f
D_i	Due date of job i
ω^E_i	Earliness penalty of job i per time unit
ω^T_i	Tardiness penalty of job i per time unit

Decision variables:

x_{fi}	1 if job i be assigned to facility f; 0, otherwise.
y_{ijkc}^{f}	1 if operation O_{ij} is processed on machine-configuration kc in facility f; 0, otherwise.
$U_{iji'j'}$	1 if operation O_{ij} is scheduled before operation $O_{i'j'}$; 0, otherwise
CO_{ij}	Completion time of operation O_{ij}
E_i	Earliness of job i
T_i	Tardiness of job i

Static model formulation:

$$\text{Min } z = \sum_{i \in \mathcal{J}} \omega_i^E \, E_i + \omega_i^T \, T_i \tag{1}$$

s.t.

$$\sum_{f \in F} x_{fi} = 1 \quad \forall \, i \in \mathcal{J} \tag{2}$$

$$\sum_{j \in N_i} \sum_{k \in K_f} \sum_{c \in C_{k,f}} y_{ijkc}^{f} \leq n_i \times x_{fi} \quad \forall \, f \in F, i \in \mathcal{J} \tag{3}$$

$$\sum_{f \in F} \sum_{k \in K_f} \sum_{c \in C_{k,f}} y_{ijkc}^{f} = 1 \quad \forall \, i \in \mathcal{J}, j \in N_i \tag{4}$$

$$y_{ijkc}^{f} \leq R_{ijkc}^{f} \quad \forall \, i \in \mathcal{J}, j \in N_i, f \in F, k \in K_f, c \in C_{k,f} \tag{5}$$

$$CO_{ij} \geq CO_{i,j-1} + \sum_{f \in F} \sum_{k \in K_f} \sum_{c \in C_{k,f}} PT_{ijkc}^{f} \, y_{ijkc}^{f} \quad \forall \, i \in \mathcal{J}, j \in N_i \tag{6}$$

$$\begin{aligned} CO_{ij} \geq CO_{i'j'} + PT_{ijkc_2}^{f} + ST_{c_1,c_2,k}^{f} \\ - \left(2 - y_{ijkc_2}^{f} - y_{i'j'kc_1}^{f} + U_{iji'j'}\right) \times M \;\; \forall f \in F, k \in K_f, c_1, c_2 \\ \in C_{k,f}, i, i' \in \mathcal{J}, j \in N_i, \; j' \in N_{i'}, O_{ij} \neq O_{i'j'} \end{aligned} \tag{7}$$

$$\begin{aligned} CO_{i'j'} \geq CO_{ij} + PT_{i'j'kc_2}^{f} + ST_{c_1,c_2,k}^{f} \\ - \left(3 - y_{ijkc_1}^{f} - y_{i'j'kc_2}^{f} - U_{iji'j'}\right) \times M \;\; \forall f \in F, k \in K_f, c_1, c_2 \\ \in C_{k,f}, i, i' \in \mathcal{J}, j \in N_i, \; j' \in N_{i'}, O_{ij} \neq O_{i'j'} \end{aligned} \tag{8}$$

$$CO_{i,n_i} + \sum_{f \in F} TT_i^{f} \, x_{fi} + E_i - T_i \leq D_i \quad \forall \, i \in \mathcal{J} \tag{9}$$

$$y_{ijkc}^{f}, x_{fi} \in \{0, 1\} \quad \forall \, f \in F, k \in K_f, c \in C_{k,f}, i \in \mathcal{J}, j \in N_i \tag{10}$$

$$CO_{ij}, E_i, T_i \geq 0 \quad \forall \, i \in \mathcal{J}, j \in N_i \tag{11}$$

The objective of the proposed model is to minimize the total weighted lateness, which is presented in Eq. (1). Constraints (2) ensure each job to be allocated to one of the existing facilities. Constraints (3) guarantee that all operations of a job be performed on the allocated facility, where $n_i = |N_i|$ is the total number of operations for job i. Constraints (4) state that each operation should be processed. Constraints (5) also prevent assigning of the operations to the infeasible machine-configurations. Constraints (6) ensure that all operations of a job should be performed based on the prespecified sequence, and minimum time between two consequent operations (i.e., the processing time of the precedence operation) should be respected. On each RMT k in a facility, Constraints (7) and (8) prevent the overlapping of the assigned operations, where M is a big positive number. For each job i, Constraints (9) calculate the amount of E/T time. Constraints (10) and (11) define the type of decision variables.

3.3. Rescheduling model

In this subsection, the presented static mathematical model is extended to be utilized in a dynamic environment. In this state, some new jobs arrive when several existing jobs have already been scheduled in each facility. The objective is to assign new jobs to the facilities as well as reschedule the operations of newly assigned jobs and the remained operations of the old jobs to minimize the total weighted lateness. The required real-time data of the distributed manufacturing system can be provided utilizing cyber-physical shop floors (CPSF). Herein, a scheduling module is designed to use the provided real-time data to update a current schedule in the shop floors. We can consider three basic levels under an IoT environment for the shop floors. To provide the autonomous analysis of system status as well as make a real-time response to dynamic events, a conceptual framework of information flow is presented in Fig. 1.

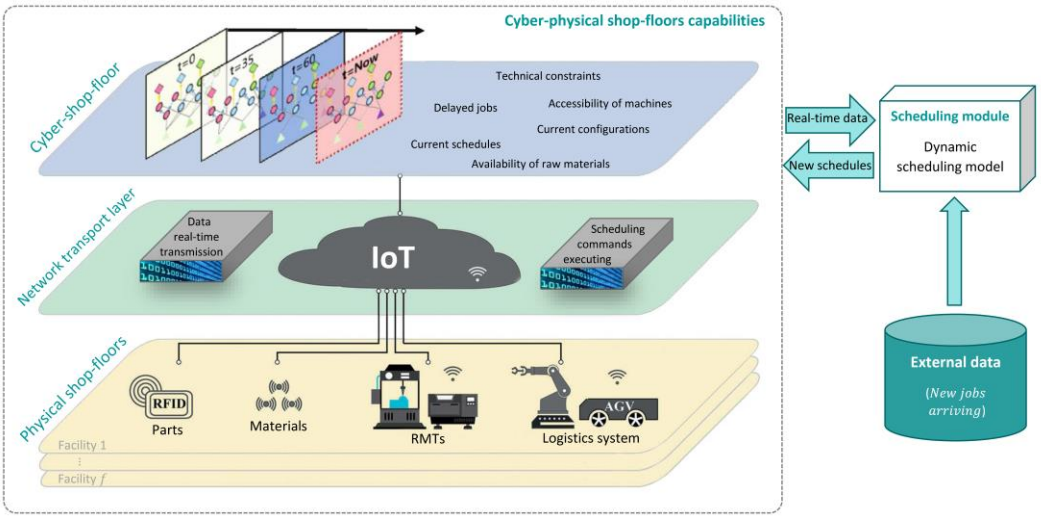


Fig. 1. Dynamic scheduling task using CPSF capabilities (Mahmoodjanloo et al. 2020b).

To extend the static model to be applied in a dynamic environment, we need to update some old sets and define some new sets/parameters as well. It is supposed that new jobs arrive at time t_1, and the responding times (Δt_f) can be estimated separately for each facility as different facilities maybe have different conditions to manage their logistic systems or provide the required materials. Hence, insertion time (t_2^f) can be calculated by:

$$t_2^{(f)} = t_1 + \Delta t_f \quad \forall f \in F \tag{12}$$

Therefore, the updated sets can be defined as follows:

$\mathcal{J}'$ Set of all uncompleted jobs (WIPs) until the insertion time, where $\mathcal{J}' \subseteq \mathcal{J}$

$\mathcal{J}''$ Set of new arrived jobs

N_i' Set of all new job operations (for $i \in \mathcal{J}''$) or unprocessed operations of existing jobs, where $N_i' \subseteq N_i$ for $i \in \mathcal{J}'$

$j_1^{(i)}$ Index of the first unprocessed operation of job i, where $j_1^{(i)} \in N_i'$ and $j_1^{(i)} - 1 \notin N_i'$ (note that $j_1^{(i)} = 1$ for $i \in \mathcal{J}''$)

$c_{k,f}'$ Index of the current configuration of machine k in facility f at time t_2^f, where $c_{k,f}' \in C_{k,f}$

CJ_i Completion time of job i in the current schedule ($CJ_i = CO_{i,n_i}^*$)

As a new job arrives, the scheduling module uses the real-time data obtained from CPSF to extract the state of the shop floors and perform a rescheduling task. Hence, it needs to determine the earliest possible start time of each current job (SJ_i) and each

machine ($SM_{k,f}$). Eq. (13) calculates the value of SJ_i for the existing jobs, and Eq (14) states a constraint to calculate SJ_i for the new jobs.

$$SJ_i = \max\left\{CO_{i,j_1^{(i)}-1}, t_2^{(f)}\right\} \quad \forall i \in \mathcal{J}' \ \text{ if } \ x_{fi}^* = 1 \tag{13}$$

$$SJ_i = \sum_{f \in F} t_2^{(f)} x_{fi} \quad \forall i \in \mathcal{J}'' \tag{14}$$

Also, the earliest possible start time of machine k in facility f can be extracted using Eq. (15).

$$SM_{k,f} = \begin{cases} t_2^{(f)} & \text{No operation is being processed on the machine at time } t_2^{(f)} \\ CO_{i,j_1^{(i)}-1} & \text{Operation } O_{i,j_1^{(i)}-1} \text{ is being processed on the machine at time } t_2^{(f)} \end{cases} \tag{15}$$

Fig. 2 presents a simple example to illustrate how the value of SJ_i and $SM_{k,f}$ can be extracted based on the introduced equations. In this example, there are two jobs (i.e., job_1 and job_2 with four and three operations, respectively), which have already been scheduled on two existing machines. It is supposed that a new job arrives at time $t_1 = 90$, and the rescheduling task (new job insertion) can be applied at time $t_2^{(f)} = t_1 + \Delta t_f = 180$. At this time, operation $O_{2,2}$ is being processed on the second machine. Hence, the earliest possible start time of the second machine is equal to the completion time of $O_{2,2}$, i.e., $SM_{2,f} = 200$, while such time for the first machine is equal to $t_2^{(f)}$, i.e., $SM_{1,f} = 180$. Also, we can determine the earliest possible start time of the jobs as $SJ_1 = \max\{170, 180\} = 180$, and $SJ_2 = \max\{200, 180\} = 200$.

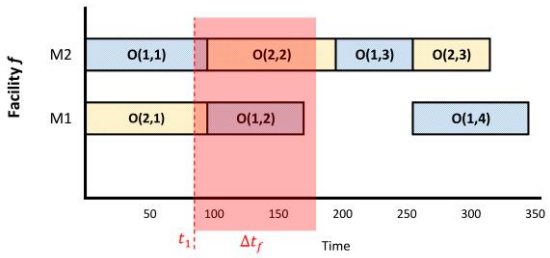


Fig. 2. An example to extract SJ_i and $SM_{k,f}$

Considering the optimal value of the variables $x_{fi}^* \, \forall i \in \mathcal{J}$ and the introduced sets/parameters as the initial conditions, the dynamic model can be developed as follow. Constraints (4) and (5) should be considered replacing updated set $\mathcal{J}' \cup \mathcal{J}''$ instead of the previous set $\mathcal{J}$. In Constraint (2), set $\mathcal{J}$ should be replaced by set $\mathcal{J}''$. Moreover, as the existing jobs cannot be reassigned to another facility, in Constraints (3) and (9), the previous optimum value of the related decision variable should be

considered as a parameter ($x^*_{fi}\ \forall i \in \mathcal{J}'$). For Constraints (7) and (8), the job-related sets should be updated as $i, i' \in \mathcal{J}' \cup \mathcal{J}'', j \in N'_i,\ j' \in N'_{i'}$. Constraint (6) should be replaced by Eqs. (16) - (18).

$$CO_{ij} \geq CO_{i,j-1} + \sum_{f \in F} \sum_{k \in K_f} \sum_{c \in C_{k,f}} PT^f_{ijkc}\, y^f_{ijkc} \quad \forall\, i \in \mathcal{J}' \cup \mathcal{J}'', j \in N'_i - \left\{j_1^{(i)}\right\} \tag{16}$$

$$CO_{i,j_1^{(i)}} \geq SJ_i + PT^f_{ij_1^{(i)}kc} + ST^f_{c'_{k,f},c,k} - \left(1 - y^f_{ij_1^{(i)}kc}\right) \times M \quad \forall i \in \mathcal{J}' \cup \mathcal{J}'' \tag{17}$$

$$CO_{i,j_1^{(i)}} \geq SM_{k,f} + PT^f_{ij_1^{(i)}kc} + ST^f_{c'_{k,f},c,k} - \left(1 - y^f_{ij_1^{(i)}kc}\right) \times M \quad \forall i \in \mathcal{J}' \cup \mathcal{J}'' \tag{18}$$

Eventually, the rescheduling problem can be solved considering the objective function (19), where new variables E'_i and T'_i measure the deviation of new schedule for the old jobs $i \in \mathcal{J}'$ based on Equation (20) considering associated weights δ_1 and δ_2.

$$\text{Min } z = \sum_{i \in \mathcal{J}' \cup \mathcal{J}''} \omega^E_i E_i + \omega^T_i T_i + \sum_{i \in \mathcal{J}'} \delta_1 E'_i + \delta_2 T'_i \tag{19}$$

$$CO_{i,n_i} + E'_i - T'_i = CJ_i \quad \forall\, i \in \mathcal{J}' \tag{20}$$

4. Solution approach

A distributed job-shop scheduling problem with reconfigurable machine tools is an extension of the classical job-shop scheduling problem. Hence, it is an NP-hard problem (Garey, Johnson, and Sethi 1976). In this section, to efficiently search the solution space of the problem a self-adaptive version of a new presented meta-heuristic algorithm, named Equilibrium Optimizer (EO), is developed. The EO algorithm has recently been introduced based on the concept of the behavior of dynamic and equilibrium state of the control volume mass balance models, i.e., inspired from the physics principle of mass conservation during entering, generating, and leaving in a control volume (Faramarzi et al. 2020). In the following section, firstly the main steps of EO are presented, and subsequently, the procedure of self-adaptive control parameters is introduced.

4.1. Equilibrium optimizer (EO) algorithm

Faramarzi et al. (2020) introduced EO that is a population-based meta-heuristic algorithm, in which the particles (i.e., solutions) and the associated concentrations (i.e., positions) can be considered as search agents. The concentration of each agent is

randomly updated for a set of best-so-far particles (i.e., equilibrium candidates) to achieve the equilibrium state (i.e., the best result). The concentration of search agents is updated based on Eq. (21) in each control volume (i.e., iteration).

$$\vec{C}_{new} = \vec{C}_{eq} + \left(\vec{C} - \vec{C}_{eq}\right).\vec{F} + \frac{\vec{G}}{\vec{\lambda}.V}\left(1 - \vec{F}\right) \tag{21}$$

where $\vec{C}$ and $\vec{C}_{new}$ are the concentration vectors in the current and next iterations, respectively. $\vec{C}_{eq}$ is a randomly selected candidate from a set of equilibrium concentrations, namely equilibrium pool ($C_{eq,pool}$). The updating equation contains three terms. The first and second terms have an important role in exploration aims, and the third term dependent on the value of random vector $\vec{\lambda}$, namely turnover rate, mostly plays the role of an exploiter to search solutions closer to the selected candidate. It is worth noting that small values in the turnover rate vector lead to exploration in the related dimensions. Besides, control volume V as a constant parameter can help the searching process by adjusting the scale of the denominator. Moreover, vector $\vec{F}$ is an exponential term to control exploration and exploitation aims in a searching process, which is presented by:

$$\vec{F} = a_1.Sign(\vec{r} - 0.5).\left[e^{-\vec{\lambda}t} - 1\right] \tag{22}$$

In the formulation of $\vec{F}$, the function *sign* is used to control the direction of search depending on $\vec{r}$, which is a random vector between 0 and 1. Time parameter t that is dependent on the iterations is used to control the behavior of parameter $\vec{\lambda}$ during the implementation of the algorithm. Eq. (23) is proposed for this parameter.

$$t = \left(1 - \frac{Iter}{Max\_Iter}\right)^{a_2 \times \frac{Iter}{Max\_Iter}} \tag{23}$$

where max_iter is the maximum number of iterations and $Iter$ is the current iteration of the algorithm which is equal to a position update for each particle. Besides, parameters a_1 and a_2 are respectively used to control exploration and exploitation aims.

Another important vector introduced in the algorithm is generation rate $\vec{G}$. It is utilized to help in the exploration using the participation probability of the selected equilibrium concentration. This vector is defined as Eq. (24).

$$\vec{G} = \overrightarrow{GCP}.\left(\vec{C}_{eq} - \vec{\lambda}.\vec{C}\right).\vec{F} \tag{24}$$

where the vector $\overrightarrow{GCP}$ is a generation rate control parameter, which is constructed by the repetition of the same value resulted from Eq. (25). In this formula, parameters r_1 and r_2 are random parameters between 0 and 1, and the generation probability (GP) is another input parameter of EO. The GP is the percentage of particles, which uses a generation term to update their states.

$$\overrightarrow{GCP} = \begin{cases} 0.5\, r_1 & r_2 \geq GP \\ 0 & r_2 < GP \end{cases} \quad (25)$$

Finally, it is worth noting that all multiplications of vectors used in the above-mentioned formulas are a type of element-wise multiplication (Hadamard product).

4.2. Proposed Self-Adaptive Equilibrium optimizer (SAEO)

In the classic EO presented in the previous subsection, there are three input control parameters including a_1, a_2, and GP. The authors proposed a value for each parameter, (i.e., $a_1 = 2$, $a_2 = 1$, and $GP = 0.5$). It is also mentioned that the value of parameters can be tuned for each problem to achieve an efficient level of exchanges between exploration and exploitation mechanisms. In addition to these control parameters, there is a random vector $\vec{\lambda}$, which has an important role in the search process especially by affecting the vector $\vec{F}$. This role is considerable because the turnover rate $\vec{\lambda}$ has an important effect simultaneously on exploration and exploitation mechanisms in different aspects of a solution space. The classic EO uses time-dependent parameter t to control the behavior of $\vec{\lambda}$ during the implementation of the algorithm. However, it seems that the utilized control mechanism has not enough effectiveness because it does not utilize the experience of used particles during the search process. Hence, we propose a self-adaptive mechanism to automatically control the turnover rate by the algorithm.

For a minimization problem, the gap between the concentration of each particle p and the worst particle in the iteration $Iter$ is calculated by:

$$\Delta fit_p^{Iter} = fit_{max}^{Iter} - fit\left(\vec{C}_p^{Iter}\right) \quad (26)$$

where $fit\left(\vec{C}_p^{Iter}\right)$ represent the fitness value of particle p in iteration $Iter$, and fit_{max}^{Iter} is the fitness value of the worst particle in iteration $Iter$, i.e., $fit_{max}^{Iter} = \max_p\left\{fit\left(\vec{C}_p^{Iter}\right)\right\}$. The weighted average value in search direction d of the solution space can be calculated by:

$$\bar{\lambda}_d^{Iter} = \sum_p w_p^{Iter} \times \lambda_{p,d}^{Iter} \quad (27)$$

where $\lambda_{p,d}^{Iter}$ is the d-th vector element, i.e., the value of turnover rate in search direction d of the vector $\vec{\lambda}_p^{Iter} = \left\|\lambda_{p,d}^{Iter}\right\|$. Besides, w_p^{Iter} is the associated weight of particle p in the iteration $Iter$ that is calculated by:

$$w_p^{Iter} = \frac{\Delta fit_p^{Iter}}{\sum_{p=1}^{nPop} \Delta fit_p^{Iter}} \quad (28)$$

Finally, the turnover rate of the particle p in the next iteration can be updated as Eq. (29), where N is the normal distribution function with the mean $\bar{\lambda}_d^{Iter}$ and the standard deviation σ_{Iter}.

$$\lambda_{p,d}^{Iter+1} = N\left(\bar{\lambda}_d^{Iter}, \sigma_{Iter}\right) \quad (29)$$

Hence, the next turnover vector can be obtained as $\vec{\lambda}_p^{Iter+1} = \left\|\lambda_{p,d}^{Iter+1}\right\|$ while the considered standard deviation in each iteration is calculated as Eq. (30). This mechanism lets the algorithm generate a wider range of dispersion in its initial iterations. Thereafter, the standard deviation should be reduced to extract a less dispersion range around the weighted mean of obtained turnover rates. The value (limit) of σ_{Iter} approaches from $\sigma_1 + \sigma_2$ (at the beginning of a search process) to σ_1 (at the end of the search process). Theoretically, we should consider the constraint $0 < \sigma_1 + \sigma_2 \leq 1$ while $\sigma_1, \sigma_2 > 0$.

$$\sigma_{Iter} = \sigma_1 + \sigma_2 \left(1 - \left(\frac{Iter}{Max\_Iter}\right)^2\right) \quad (30)$$

4.3. Details of the proposed hybrid SAEO to solve dynamic DFJSP-CDST

In this section, the main steps of the proposed hybrid SAEO-VNS algorithm are presented to solve DFJSSP-CDST.

Encoding procedure:

An equilibrium optimizer is a continuous algorithm. To solve DFJSSP-CDST as a discrete combinatorial optimization problem, we need to use an encoding approach. Herein, the approach proposed by Mahmoodjanloo et al. (2020a) is utilized to represent feasible solutions to the problem. They used a two-dimensional matrix ($A_{2\times n}$) with real random values to determine operations scheduling and machine configuration decisions in a static single-facility FJSSP-CDST problem. We enhance

their encoding approach by adding the possibility of job assignment decisions for a dynamic distributed version. Hence, a new row is added to the matrix. The new matrix ($A_{3\times n}$) can be utilized to randomly generate a solution that represents a feasible schedule for the DFJSSP-CDST. In this matrix, the number of columns is equal to the total number of operations of uncompleted jobs in the distributed production systems (i.e., $n = \sum_{i\in\mathcal{J}'}|N_i|$).

Table 1 represents a simple instance designed to illustrate the encoding scheme. In this instance, there are two facilities each one contains two RMTs. Four jobs should be assigned and scheduled in these facilities.

Table 1. Data for the illustrated instance

	Job1				Job2			Job3			Job4		
f, k, c	1	2	3	4	1	2	3	1	2	3	1	2	3
1, 1, 1					75				87	91		52	57
1, 1, 2		72		86	54			96					
1, 2, 1		65	55	61		62	81			59	86		76
1, 2, 2	93	97	63					67	96	62			
2, 1, 1	81		71					83	85		95		
2, 1, 2				80		97	77	75	60				60
2, 1, 3	60	58										84	
2, 2, 1					75				87	91		52	
2, 2, 2		72		86	54			96					55
D_i		430				650			400			400	

The random matrix presented in Fig. 3 illustrates the encoding scheme for this instance as well. To extract the random solution associated with this code, three steps should be followed. At Step 1, each job is assigned to one of the facilities based on the average value of its related elements in the third row of matrix A. For this purpose, we divide the interval [0,1] to $|F|$ equal and incompatible subsets.

In the illustrated example of Fig. 3, in which there are two facilities ($|F| = 2$), if the average value of elements for a job be smaller than 0.5, the job should be assigned to the first facility, otherwise, it should be assigned to the second facility. Thereafter, the sequence priority of operations in each facility will be determined based on their elements in the second row of matrix A. Based on this procedure, after considering the precedence relation of operations for each job, each operation, which has a smaller value, achieves the next position in the sequence priority list. Moreover, for each operation, a machine-configuration will be selected based on a greedy method, utilizing the related elements in the first row of matrix A. For this purpose, a probability matrix ($\bar{P}^f = \|\bar{P}^f_{ijkc}\|$) is first calculated for each facility based on Eq. (31). For example,

as the first job has already been assigned to the second facility (in Step 1), the related probability values of O_{11} is calculated as $[0.43, 0, 0.57, 0, 0]^T$ (i.e., the associated column in matrix $\bar{P}^2$, see Appendix A). Eventually, since $A_{11} = 0.53 > 0.43$, operation O_{11} should be processed on the third configuration of machine 2 in the second facility.

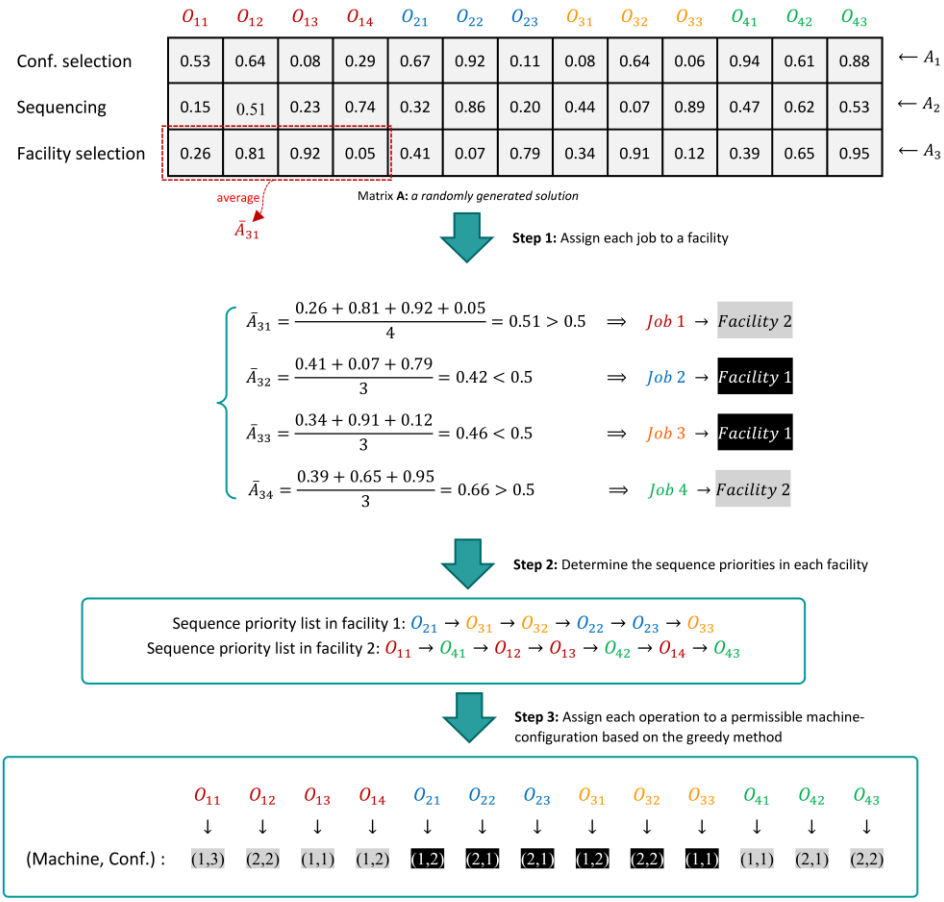


Fig. 3. Random solution (i.e., particle) for the example.

$$\bar{P}^f_{ijkc} = \frac{\frac{1}{PT^f_{ijkc}}}{\sum_{k \in K_f, c \in C_{k,f}} \frac{1}{PT^f_{ijkc}}} \quad \forall f \in F, i \in \mathcal{J}, j \in N_i, x_{fi} = 1, PT^f_{ijkc} \neq 0 \qquad (31)$$

After assigning the jobs to the facilities and determining the related machine-configuration of each operation, an active schedule of operations in each facility can be extracted based on the obtained sequence priority and using the G&T method (Giffler and Thompson 1960) as a well-known constructive heuristic algorithm in the related literature (Sha and Hsu 2006; Ahmadian et al. 2021). The procedure of the G&T method is presented in Appendix B. The resultant schedule of the randomly generated

code in Fig. 3 is represented in Fig. 4. The value of the total weighted E/T is 542 for this schedule ($\omega_i^E = 1$ and $\omega_i^T = 2$).

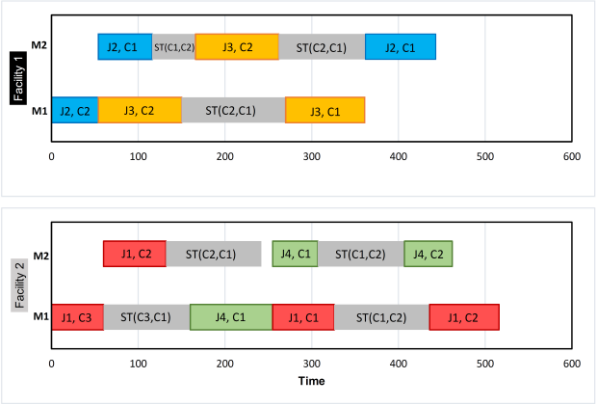


Fig. 4. Resultant schedule of the particle

Local search function:

To solve the DFJSP-CDST as a combinatorial optimization problem, it is needed to improve the performance of the EO algorithm, which is originally developed for continuous optimization, by adding a local search method. In this hybrid approach, the searching mechanisms of EO are used for global exploration/exploitation of continuous search space (i.e., encoding space) while the proposed local search enhance the exploitation ability for searching in discrete environment (i.e., decoding space) at vicinity of solutions obtained in each iteration.

Hence, considering the structure of solutions in the studied problem, seven heuristic neighborhood search (HNS) methods are proposed as follows:

HNS_1: Select two facilities randomly, select a random job in each facility, and replace two selected jobs.

HNS_2: Select two facilities based on a random greedy approach. The first facility should be selected regarding to the total tardiness-related probability ($PrTT_f$) (i.e., calculated in Eq. (32)), and the second facility should be selected regarding to the total earliness-related probability ($PrTE_f$) (i.e., calculated in Eq. (33)). Remove a random selected job from the first facility, and insert it to the second one.

$$PrTT_f = \frac{\sum_{i \in \mathcal{J}' \cup \mathcal{J}''} T_i \times \hat{x}_{fi}}{\sum_{i \in \mathcal{J}' \cup \mathcal{J}''} T_i} \quad \forall f \in F \tag{32}$$

$$PrTE_f = \frac{\sum_{i \in \mathcal{J}' \cup \mathcal{J}''} E_i \times \hat{x}_{fi}}{\sum_{i \in \mathcal{J}' \cup \mathcal{J}''} E_i} \quad \forall f \in F \tag{33}$$

where $\hat{x}_{fi}$ represents the value of the assignment variable in the related solution.

HNS_3: Select a facility randomly, and change the priorities of its two randomly selected operations.

HNS_4: Select an operation randomly, and change its machine-configuration based on the random greedy approach (if possible).

HNS_5: Select randomly an operation that is immediately positioned after a setup operation. Change the configuration of the related machine to a randomly selected possible configuration.

HNS_6: Select randomly an operation that is immediately positioned before a setup operation. Change the configuration of the related machine to a randomly selected possible configuration.

HNS_7: Identify the operations which are immediately positioned between two setup operations. For each identified operation, if the related configurations of the previous and the next operations are same, change the configuration of the selected operation to it (if possible).

It is worth noting that, after applying each HNS in decoding space, the related action should be transformed by an appropriate random change in encoding space. The proposed local search process can be developed based on the introduced HNSs as presented in Procedure_1 utilizing a pipe neighborhood change step (Hansen et al. 2017).

Procedure_1. Variable Neighborhood Search.

Inputs: Particle ($\vec{C}$)

Algorithm parameter: R_{max}

Set $\vec{C}' = \vec{C}$
Set $r = 0$, i.e., the number of iterations without improvement
Set $k = 1$, i.e., the index of Heuristic Neighborhood Search method
While $r \leq R_{max}$
    Find $\vec{C}'_{new}$ **Calling** $HNS_k(\vec{C}')$
    if $fit(\vec{C}'_{new}) < fit(\vec{C}')$ **Then**
        Set $\vec{C}' = \vec{C}'_{new}$
        Set $r = 0$
    else
        Set $k \leftarrow k + 1$
        If $k = 7$ **Then** $k = 1$
        Set $r \leftarrow r + 1$
    end
    Apply $HNS_7(\vec{C}')$
end

Output: Improved particle ($\vec{C}'$)

Rescheduling function:

To apply the proposed algorithm in a dynamic environment, it is needed to be adjusted for the implementation of a rescheduling function. Moreover, the required data can be provided utilizing industrial IoT. Digital twin sends real-time data from shop floors, so the last situation of all machines and works in progress (i.e., uncompleted jobs) will be specified once a new event occurred (e.g., arriving at a new job). The conceptual framework of the scheduling module has been presented in Fig. 5.

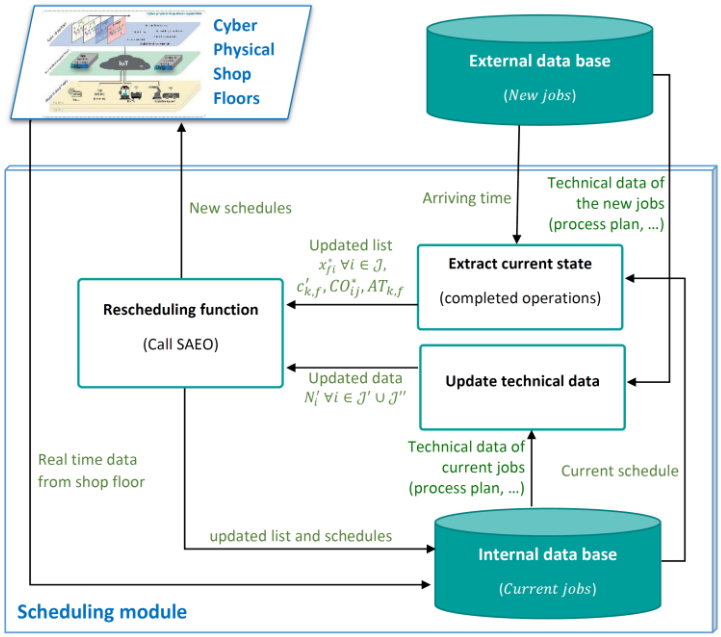


Fig. 5. Conceptual framework of the scheduling module

Based on the presented data flow system in Fig. 5, the introduced encoding procedure should be updated to be used in a dynamic environment. The updating process of a particle has been presented in Procedure_2.

Procedure_2. Updating process of a particle.

Inputs: Old particle, Updated list, Updated data (see Fig. 5)	
1	**Delete** all columns related to the operations of completed jobs
2	**Freeze** all columns related to the completed operations of WIPs
3	**Freeze** the third row of the particle (for WIPs)
4	**Generate** a random particle for newly arrived jobs
5	**Add** the new particle at the end of the old one
Output: Updated particle	

The following pseudo-code presents the steps of the hybrid SAEO-VNS algorithm to solve the dynamic DFJSSP-CDST.

Procedure_3. Pseudo-code of the hybrid SAEO-VNS to solve the dynamic DFJSSP-CDST.

Problem parameters: Old particles, Updated list of jobs, and Updated technical data

Algorithm parameters: $(a_1, GP, \sigma_1, \sigma_2)$ and $(Max\_Iter, nPop)$

Initialization:
  For $p = 1{:}nPop$
    Update the particle $\vec{C}_p$ **Calling** Procedure_2($\vec{C}_p$, Updated list, Updated data)
    Extract resultant schedule of the updated particle $\vec{C}_p$ **Calling** Procedure_B1
    Calculate $fit(\vec{C}_p)$, i.e., the fitness of resultant schedule (Total weighted lateness)
Set $Iter = 1$
While $Iter \leq Max\_Iter$
  Update equilibrium pool ($C_{eq,pool}$)
    Find four best particles as $\vec{C}_{eq\_1}, \vec{C}_{eq\_2}, \vec{C}_{eq\_3}, \vec{C}_{eq\_4}$
    Calculate $\vec{C}_{eq\_ave} = (\vec{C}_{eq\_1}, +\vec{C}_{eq\_2} + \vec{C}_{eq\_3} + \vec{C}_{eq\_4})/4$
    Construct $C_{eq,pool} = \{\vec{C}_{eq\_1}, \vec{C}_{eq\_2}, \vec{C}_{eq\_3}, \vec{C}_{eq\_4}, \vec{C}_{eq\_ave}\}$
    For Each particle in $C_{eq,pool}$ **Do** local search **Calling** Procedure_1
  Calculate $\bar{\lambda}^{Iter}$, i.e., the weighted average value of turnover rate using Eqs. (26)-(28)
  Calculate σ_{Iter} using Eq. (30)
  Set $Iter \leftarrow Iter + 1$
  For $p = 1{:}nPop$
    Select a random particle ($\vec{C}_{eq\_*}$) from $C_{eq,pool}$
    Generate random vector $\vec{r}$ and $\vec{\lambda}_p^{Iter}$ using Eq. (29)
    Calculate $\vec{F}$ using Eq. (22)
    Calculate $\vec{G}$ using Eqs. (24) and (25)
    Update concentration $\vec{C}_p \leftarrow \vec{C}_{eq\_*} + (\vec{C}_p - \vec{C}_{eq\_*}).\vec{F} + \frac{\vec{G}}{\vec{\lambda}_{p\_V}^{Iter}}(1 - \vec{F})$
    Extract resultant schedule of the updated particle $\vec{C}_p$ **Calling** Procedure_B1
    Calculate $fit(\vec{C}_p)$

Outputs: The best schedule, Data of the last generation

5. Computational results

The purpose of this section is to verify the performance of the proposed algorithm based on numerical experiments. Hence, the performance of algorithm SAEO-VNS is compared with the results from the MILP model solved by the CPLEX solver and the results obtained from applying three other meta-heuristics. The comparisons are performed among outputs of the mentioned algorithms based on 32 randomly generated instance problems. The MILP model is implemented in GAMS 24.1.3 and solved utilizing the solver CPLEX. Also, the meta-heuristic algorithms are coded on MATLAB 2019. All test instances are performed on a computer with a 2.90 GHz Intel (R) Core (TM) i7-7820HK CPU and with RAM 32.0 GB.

5.1. Random instance generation

To evaluate the results of the proposed algorithm, two computational experiments are designed based on several random instance problems. Using the range of parameters, which are presented in Table C1 (in Appendix C), two sets of problems are generated to test the methods in static and dynamic states. At the first set, 16 instance problems in different sizes are generated to test the scheduling methods (static state). Afterward, 16 instance problems are considered to test the rescheduling methods (dynamic state). General specification of the instance problems utilized for static (Subset-S) and dynamic (Subset-D) states respectively are presented in Tables C2 and C3.

5.2. Evaluation of results

This section aims to test the performance of the proposed MILP models and the proposed algorithm. The obtained results of the proposed SB models (in both static and dynamic states) are compared with the results of the PB models introduced by Mahmoodjanloo et al (2020b). Moreover, to test the performance of the SAEO-VNS algorithm, the obtained results are compared with the results of the classic EO (Faramarzi et al. 2020) embedded by VNS (EO-VNS) and two self-adaptive algorithms which have been recently used in the literature (i.e., including SADE-NM (Mahmoodjanloo et al. 2020a) and SA-COA (Abdollahzadeh-Sangroudi and Ranjbar-Bourani 2019)).

To guarantee the best performance, each algorithm needs some accurately calibrated parameters. Fortunately, three of the four foregoing algorithms have a self-adaptive approach to control the main parameters while only needing some initial values. Hence, we use the proposed initial values in associated original papers in our study. For the classic EO, we find that the proposed input parameters ($a_1 = 2$, $a_2 = 1$, and $GP = 0.5$) have acceptable performance in our problem too. We also use $a_1 = 2$ and $GP = 0.5$ as two input parameters in SAEO; however, about the parameters σ_1 and σ_2, a computational experiment is performed. In this experiment, we consider the first data set as the test problem. The analysis of variance (ANOVA) test is used to analyze the results. The means plot and least significant difference (LSD) intervals (at the 95% confidence level) for different levels of these parameters are represented in Fig. 6. Hence, the selected values $\sigma_1 = \sigma_2 = 0.25$ are used in the following experiments. Moreover, for both hybrid algorithms (i.e., EO-VNS and SAEO-VNS), the maximum

number of iterations without improvement in the embedded VNS is considered as $R_{max} = 20$.

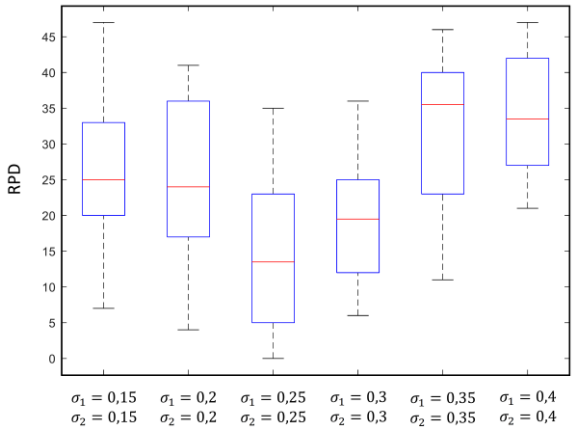


Fig. 6. Calibration of input parameters σ_1 and σ_2 for SAEO

To comparison the four meta-heuristics in a fair situation, a stopping condition is considered based on CPU time. For each instance problem, we fix it to $max\{2000, |N_i'|^2\}$ milliseconds. Moreover, the best value for the population number ($nPop$) of each algorithm was determined based on another experiment on the first data set. Based on the results of ANOVA tests, we use $nPop_{EO} = nPop_{SAEO} = nPop_{SA-COA} = 90$ and $nPop_{SADE-NM} = 120$. The means plot and LSD intervals (at the 95% confidence level) for different levels of $nPop$ of each algorithm are represented in Fig. D1 (in Appendix D). It is worth noting that to compare the results of considered algorithms on instance problems with different sizes, we use the relative percentage deviation (RPD) as a common performance measure (Mahmoodjanloo et al. 2020c). For a problem with minimization objective, the RFD can be calculated by Eq. (34), where Min represents the best-obtained value of the objective function, and $Alg.$ represent the obtained value by the algorithm.

$$RPD = \frac{Alg. - Min}{Min} \tag{34}$$

To evaluate the performance of solution methods, we first solve 16 instance problems in Subset-S using the static-version of models/algorithms. Thereafter, obtained solutions by the SB model were used as current schedules of the system to run a dynamic version of mathematical models for the related instances in Subset-D. Besides, for each instance, the best solution obtained by meta-heuristics was used to run a dynamic version of algorithms for the related instance in Subset-D. We consider

a maximum run time of 3600 seconds for both mathematical models. The objective function value (OFV) and the related CPU times (if an optimum solution is obtained in time limitation, otherwise the optimality gap) are presented in Table 2.

Also, each instance is solved 30 times by each meta-heuristic. The mean values (Ave.) and the standard deviations (S.D.) have been presented in Table 2. As can be seen in this table, none of the SB and PB models can find a feasible solution for the instances with more than three facilities within the predefined time limit. Moreover, they only solve respectively four and two instances optimally. Anyway, the experiments show that the SB model outperforms the PB model. On the other hand, SAEO-VNS has obtained better results among the meta-heuristics. A comparison among the best solutions obtained by SAEO-VNS and the SB model can prove the acceptable performance of the proposed algorithm because SAEO-VNS can find more qualified solutions in a reasonable time (Fig. 7).

Table 2. Computational comparison of the algorithms on instance problems of two data sets.

Ins.	SB model		PB model		SADE-NM		SA-COA		EO-VNS		SAEO-VNS	
ID	OFV.	Time/ Gap%	OFV.	Time/ Gap%	Ave.	S.D.	Ave.	S.D.	Ave.	S.D.	Ave.	S.D.
S1	87.3	182	87.3	305	98.2	6.8	105.7	7.0	106.4	7.5	90.2	3.6
S2	63.6	1905	68.7	8%	72.5	5.2	75.6	7.3	78.5	7.5	67.5	3.5
S3	99.3	15%	135.6	59%	111.1	10.0	121.9	11.4	134.9	14.8	110.0	6.3
S4	172.1	43%	-	-	187.2	13.2	214.4	12.2	193.7	21.8	168.4	14.5
S5	144	23%	211.2	83%	159.4	10.3	168.5	14.4	152.8	16.9	148.2	9.0
S6	191.2	51%	-	-	184.6	16.2	207.8	17.1	215.1	20.3	167.9	14.4
S7	136.6	29%	204.5	96%	146.3	9.3	157.3	13.1	156.7	17.2	137.1	10.1
S8	-	-	-	-	219.8	18.7	242.3	20.1	244.5	24.1	209.4	15.2
S9	-	-	-	-	302.5	17.8	326.7	19.7	318.5	33.9	264.5	17.4
S10	-	-	-	-	359.6	21.4	405.9	27.8	414.7	44.1	316.7	22.2
S11	-	-	-	-	364.6	23.3	420.8	31.2	423.1	33.3	338.7	25.0
S12	-	-	-	-	366.1	23.9	412.2	28.3	439.6	42.9	356.6	28.2
S13	-	-	-	-	280.8	15.9	296.7	22.8	310.8	27.5	260.4	19.7
S14	-	-	-	-	321.5	26.0	359.6	30.1	350.8	35.9	297.1	24.4
S15	-	-	-	-	348.0	23.2	386.9	27.1	429.9	43.5	308.5	23.9
S16	-	-	-	-	356.3	24.0	370.3	30.8	384.7	44.1	327.4	20.9
D1	91.3	146	91.3	270	99.9	4.2	104.1	4.6	104.6	7.2	96.5	4.4
D2	77.6	2304	85.2	11%	86.1	4.9	93.0	5.2	83.4	7.6	80.4	3.5
D3	103.5	8%	126.6	32%	115.3	6.8	128.4	8.5	123.9	10.6	109.4	5.2
D4	212.4	41%	-	-	197.6	10.1	201.0	12.2	198.0	18.8	190.5	9.8

D5	154.9	31%	187.3	59%	157.8	8.7	141.2	8.6	150.0	12.2	133.3	6.9
D6	214.2	56%	-	-	199.6	10.5	199.1	10.5	187.8	17.5	186.7	8.3
D7	197.1	22%	284	78%	189.3	9.3	201.4	9.4	168.5	18.1	168.7	9.3
D8	-	-	-	-	209.3	11.2	220.5	12.4	208.9	21.9	198.1	11.4
D9	-	-	-	-	233.0	14.0	252.5	17.1	222.7	19.8	214.7	13.6
D10	-	-	-	-	263.8	16.9	288.2	15.5	275.0	27.9	248.6	17.3
D11	-	-	-	-	246.7	15.3	238.6	15.9	264.5	28.4	207.5	14.8
D12	-	-	-	-	283.4	18.5	329.6	20.7	314.3	34.5	283.5	18.8
D13	-	-	-	-	171.6	13.3	206.1	17.2	227.3	21.7	182.2	12.6
D14	-	-	-	-	265.7	19.5	272.3	21.5	275.0	31.7	238.5	15.1
D15	-	-	-	-	286.7	21.9	321.8	18.5	315.1	33.6	297.3	17.3
D16	-	-	-	-	280.4	21.3	306.9	20.8	341.3	34.9	277.4	18.3

However, the ANOVA test is used to better understand the performance of algorithms. The results show that SAEO-VNS outperforms other algorithms as well (Fig. 8). We also use the Fisher individual tests, to pairwise compare the differences of mean RPDs (Fig. 9a). Based on this test performed at the 95% confidence level, if an interval does not contain the zero line, it means that the related two algorithms are not significantly different. The comparison of differences between the results obtained by EO-VNS and SAEO-VNS algorithms shows that the proposed self-adaptive policy to control the parameters of the EO algorithm is significantly efficient. Moreover, the obtained individual values by the rival algorithms (presented in Fig. 9b) illustrate a more suitable performance for SAEO-VNS because it obtains a greater number of the most qualified solutions. Moreover, we also checked the performance of the proposed algorithm without applying the local search step. Results show that the proposed local search can improve the performance of the hybrid algorithm about 5%.

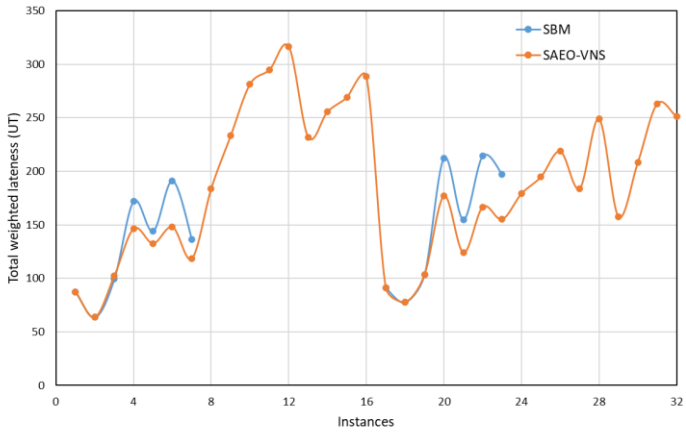


Fig. 7. Comparison SBM vs. the best solution obtained by SAEO-VNS

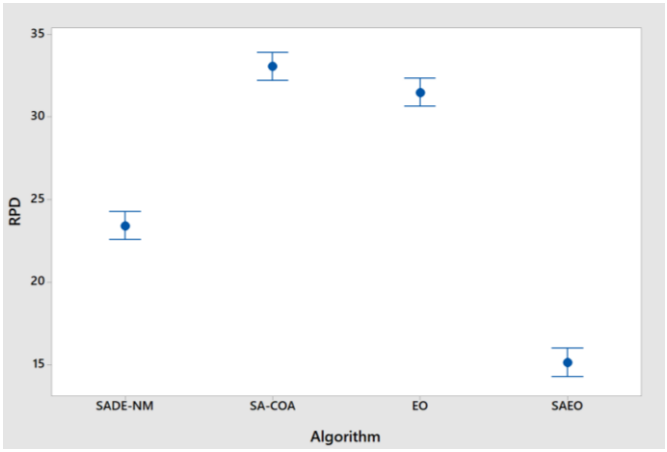
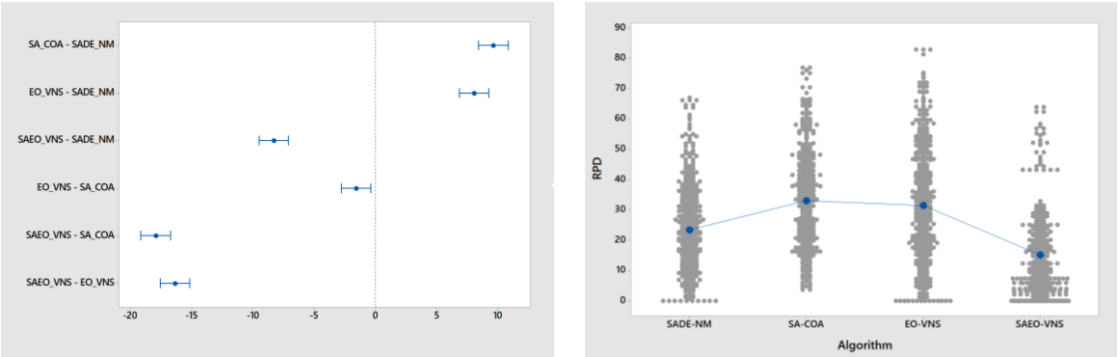


Fig. 8. Means plot and LSD intervals (at the 95% CI) for the different used algorithms

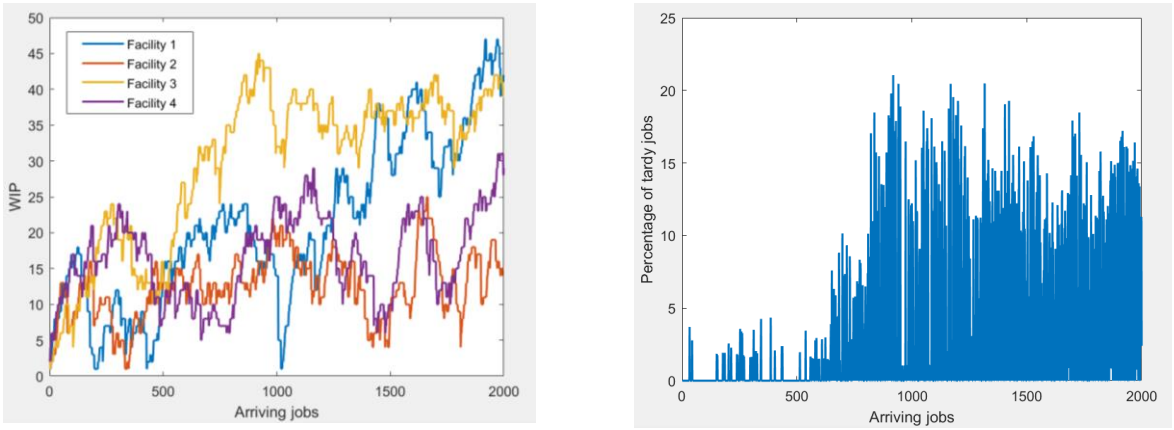


(a) (b)

Fig. 9. Fisher individual tests at 95% CI, (a) Differences of means plot, (b) Individual value plot of RPDs

5.3. Managerial insights

In this section, to evaluate the performance of the proposed method facing new arriving jobs when demand is stochastic, we develop a simulation model of a dynamic production environment based on the conceptual framework presented in Fig. 5. New jobs arrive randomly following a Poisson distribution. In other words, the time interval between two sequential events follows an Exponential distribution with a specified mean parameter λ. Hence, when a new job arrives, it is assigned to one of the facilities, where its operations are scheduled. We utilize the shop floor of instance S10 as the production environment and run our simulation model using parameter $\lambda = 100$. The number of work-in-progress (WIP) and the percentage of tardy jobs are represented in Fig. 10.

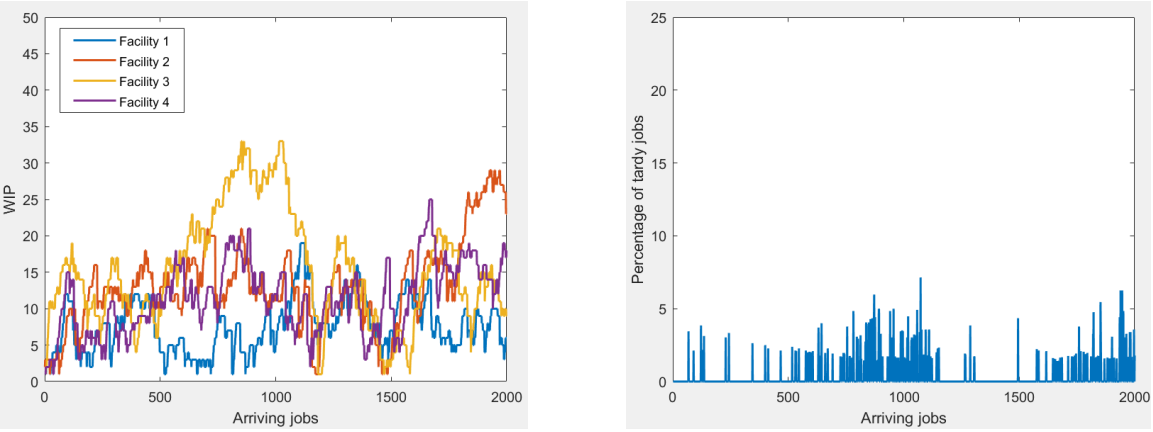


(a) (b)

Fig. 10. Simulation results of a dynamic distributed job-shop scheduling problem for Ins. S10

To have a more stable analysis, we can consider the first 1000 events as a worming period and focus on the remaining results. The simulation shows that on average 4.3071% (at most 20.4819%) of jobs will face tardiness. This provides managers the possibility of evaluating various scenarios to more effectively control the production system.

For example, to reduce the mean value of tardy jobs, the capacity of the distributed production system should be increased. On the other hand, each facility utilizes several RMTs. This leads to a level of higher flexibility in the production system. In fact, instead of buying new machines, we can change the overall capacity by adding/replacing new modules to the existing RMTs. Now, the question is which modules should be added/enhanced, on which machines, in which facilities? The simulation model can help the managers to answer such questions by predicting the value of considered key performance indicators based on existing scenarios. To have more illustration, we randomly select one configuration of each RMT in facilities 2 and 3 of the instance problem S10, and replaced it with a new module that performs the same operations 10% faster (instance problem S10_v2). The simulation shows that on average 0. 2480% (at most 7. 1429%) of jobs will face tardiness. The number of WIPs and the percentage of tardy jobs is represented in Fig. 11.



(a) (b)

Fig. 11. Simulation results of a dynamic distributed job-shop scheduling problem for Ins. S10_v2

6. Conclusions

Due to the globalization of the economy and rapidly changing market requirements, some companies have started to use the benefits of distributed manufacturing systems to have the opportunity of more adaptation by becoming closer to both customers and suppliers. Moreover, to increase the flexibility in manufacturing systems, 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. Job scheduling in a distributed system that contains RMTs is very complex especially when the decision-making environment is dynamic. Today, interconnection in a network of geographically dispersed facilities thanks to the use of the new emerging technologies of Industry 4.0 can provide manufacturers to utilize real-time data to make efficient decisions in a dynamic environment. To the best of our knowledge, there is no study to tackle this problem. In this paper, we studied a distributed job-shop rescheduling problem, in which the facilities benefit from reconfigurable machines. Firstly, the problem was modeled and solved using mathematical programming. Then, regarding the high level of complexity, a self-adaptive version of a newly introduced meta-heuristic algorithm named Equilibrium Optimizer (EO) was developed to efficiently solve medium- and larger-sized problems in a reasonable time. And finally, a simulation-optimization model was developed to evaluate the performance of the manufacturing system facing stochastically arriving jobs.

Obtained results show that the production system can be very flexible relying on its distributed facilities and reconfigurable machines. Moreover, accessibility to real-time data and using efficient decision tools can provide managers with a quick response to

changing market requirements. For future studies, some of the other dynamic job-related events (e.g., rush jobs and job cancelation) or resource-related events (e.g., machine breakdowns and shortage of materials) can be considered. Moreover, in this research, we used a single-agent approach which can better operate in optimality measures rather than multi-agent (MA) approaches. However, MA can be more effective in adjusting with dynamic environment. Hence, developing a multi-agent approach can be recommended in such an environment.

References

Ahmadian, M. M., A. Salehipour, and T. C. E. Cheng. 2021. "A meta-heuristic to solve the just-in-time job-shop scheduling problem." *European Journal of Operational Research* 288 (1): 14-29.

Chang, H. C. and T. K. Liu. 2017. "Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms." *Journal of Intelligent Manufacturing* 28(8): 1973-1986.

Chaouch, I., O. B. Driss, and K. Ghedira. 2017a, "A survey of optimization techniques for distributed job shop scheduling problems in multi-factories." In: *Computer Science On-line Conference* (pp. 369-378). Springer, Cham.

Chaouch, I., O. B. Driss, and K. Ghedira. 2017b, "Elitist ant system for the distributed job shop scheduling problem." In: *Proceeding of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (pp. 112-117). Springer, Cham.

Chaouch, I., O. B. Driss, and K. Ghedira. 2019. "A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm." *Applied Intelligence* 49(5): 1903-1924.

Faramarzi, A., M. Heidarinejad, B. Stephens, and S. A. Mirjalili, 2020. "Equilibrium optimizer: A novel optimization algorithm." *Knowledge-Based Systems* 191: 105190.

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

Garey, M. R., D. S. Johnson, and R. Sethi. 1976. "The Complexity of Flow-shop and Job-shop Scheduling." *Mathematics of Operations Research* 1 (2): 117–129.

Giffler, B. and G. L. Thompson. 1960. “Algorithms for solving production-scheduling problems.” *Operations Research* 8 (4): 487-503.

Gong, G., R. Chiong, Q. Deng, and Q. Luo. 2020. “A memetic algorithm for multi-objective distributed production scheduling: minimizing the makespan and total energy consumption.” *Journal of Intelligent Manufacturing*, Article in Press, 1-24.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423-454.

Jia, H. Z., J. Y. Fuh, A. Y. Nee, and Y. F. Zhang. 2002. “Web-based multi-functional scheduling system for a distributed manufacturing environment. *Concurrent Engineering*. 10 (1): 27-39.

Jia, H. Z., A. Y. Nee, J. Y. Fuh, and Y. F. Zhang. 2003. “A modified genetic algorithm for distributed scheduling problems.” *Journal of Intelligent Manufacturing* 14 (3-4): 351-362.

Jiang E. D., L. Wang, and Z. P. Peng. 2020. “Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition.” *Swarm and Evolutionary Computation* 58: 100745.

Li, J. Q., P. Duan, J. Cao, X. P. Lin, and Y. Y. Han. 2018. “A hybrid Pareto-based tabu search for the distributed flexible job shop scheduling problem with E/T criteria.” *IEEE Access* 6: 58883-58897.

Lohmer, J. and R. Lasch. 2020, “Production planning and scheduling in multi-factory production networks: a systematic literature review.” *International Journal of Production Research*, Article in Press, 1-27.

Lu, P. H., M. C. Wu, H. Tan, Y. H. Peng, and C. F. Chen. 2018, “A genetic algorithm embedded with a concise chromosome representation for distributed and flexible job-shop scheduling problems.” *Journal of Intelligent Manufacturing*, 29 (1): 19-34.

Luo, Q., Q. Deng, G. Gong, L. Zhang, W. Han, and K. Li. 2020. “An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers.” *Expert Systems with Applications 160*: 113721.

Mahmoodjanloo, M., R. Tavakkoli-Moghaddam, A. Baboli, and A. Bozorgi-Amiri. 2020a. “Flexible job shop scheduling problem with reconfigurable manufacturing tools: A modified differential evolution algorithm.” *Applied Soft Computing* 94: 106416.

Mahmoodjanloo, M., R. Tavakkoli-Moghaddam, A. Baboli, and A. Bozorgi-Amiri. 2020b. “Dynamic distributed job-shop scheduling problem consisting of reconfigurable machine tools.” In: B. Lalic, V. Majstorovic, U. Marjanovic, G. von Cieminski and D. Romero (Eds.), *Advances in Production Management Systems (APMS): Towards Smart and Digital Manufacturing* (Vol. 2), IFIP Working Group 5.7, Vol. 592, Springer Nature, Cham, Switzerland, pp. 460-468

Mahmoodjanloo, M., R. Tavakkoli-Moghaddam, A. Baboli, and A. Jamiri. 2020c, “A multi-modal competitive hub location pricing problem with customer loyalty and elastic demand.” *Computers & Operations Research*, *123*, 105048.

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

Naderi, B., and A. Azab. 2015. “An improved model and novel simulated annealing for distributed job shop problems.” *The International Journal of Advanced Manufacturing Technology* 81(1-4): 693-703.

Romero-Silva, R. and G. Hernández-López. 2019. “Shop-floor scheduling as a competitive ad-vantage: A study on the relevance of cyber-physical systems in different manufacturing contexts.” *International Journal of Production Economics* 224: 107555.

Sha, D. Y. and C. Y. Hsu. 2006. “A hybrid particle swarm optimization for job shop scheduling problem.” *Computers & Industrial Engineering* 51 (4): 791-808.

Toptal, A., and I. Sabuncuoglu. 2010. “Distributed scheduling: a review of concepts and applications.” *International Journal of Production Research* 48 (18): 5235-5262.

Tian, S., T. Wang, L. Zhang, and X. Wu. 2019. “The internet of things enabled manufacturing enterprise information system design and shop floor dynamic scheduling optimization.” *Enterprise Information Systems*, Article in Press, 1-26.

Yelles-Chaouche, A. R., E. Gurevsky, N. Brahimi, and A. Dolgui. 2020. “Reconfigurable manufacturing systems from an optimisation perspective: A focused review of literature.” *International Journal of Production Research*, Article in Press, 1-19.

Wang, L., H. Y. Feng, N. Cai, and W. Jin. 2007. “An effective approach for distributed process planning enabled by event-driven function blocks.” In: *Process Planning and Scheduling for Distributed Manufacturing* (pp. 1-30). Springer, London.

Wu, M. C., C. S. Lin, C. H. Lin and C. F. Chen. 2017. “Effects of different chromosome representations in developing genetic algorithms to solve DFJS scheduling problems.” *Computers & Operations Research* 80: 101-112.

Wu, X., X. Liu, and N. Zhao. 2019. “An improved differential evolution algorithm for solving a distributed assembly flexible job shop scheduling problem.” *Memetic Computing* 11 (4): 335-355.

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

Appendix A. Calculation of the probability matrix $\bar{P}^f$

For the presented instance in Table 1, the associated probability matrix $\bar{P}^f$ can be calculated as Table A1. For example, the first column of the probability matrix $\bar{P}^2$, i.e., the probability vector to select a random machine-configuration to perform operation O_{11} in the second facility, can be calculated using the related vector of processing time $[81, 0, 60, 0, 0]^T$ and utilizing Eq. (31) as $[0.43, 0, 0.57, 0, 0]^T$. The first and the third elements are calculated as $0.43 = (\frac{1}{81})/(\frac{1}{81} + \frac{1}{60})$ and $0.57 = (\frac{1}{60})/(\frac{1}{81} + \frac{1}{60})$.

Table A1. Data for the illustrated instance

	Job1				Job2			Job3			Job4		
	1	2	3	4	1	2	3	1	2	3	1	2	3
$\bar{P}^1$					0.42				0.52	0.25		1	0.57
		0.35		0.41	0.58			0.41					
		0.39	0.53	0.59		1	1			0.38	1		0.43
	1	0.26	0.47					0.59	0.48	0.37			
$\bar{P}^2$	0.43		1					0.34	0.29		1		
				0.52		1	1	0.37	0.42				0.48
	0.57	0.55										0.38	
					0.42				0.29	1		0.62	
		0.45		0.48	0.58			0.29					0.52

Appendix B. Procedure of decoding scheme

The list of used parameters and notations:

PS	Partial schedule of operations
Ω	Subset of operations that can be scheduled.
PT_{ij}	Processing time of operation O_{ij}
ES_{ij}	Earliest time that operation $O_{ij} \in \Omega$ can be started.
EF_{ij}	Earliest time that operation $O_{ij} \in \Omega$ can be finished.

It is worth noting that the procedure is called for each facility after assigning the jobs to the facilities and allocation of each operation to a machine-configuration. Hence, in the above-mentioned notations, the indices of facilities and machine-configurations are ignored.

Procedure_B1. Pseudo-code of the decoding method based on the G&T approach.

Inputs: PT_{ijk}, ST_k, *Sequence priority*

Initialization:
  Set $PS = \emptyset$
  Extract all operations without predecessors and set them in Ω
  Set $Stop = False$
Do
  Calculate ES_{ij} for $O_{ij} \in \Omega$, i.e., equal to the maximum of completion time of the previous operation of job j and sum of the completion time of machine k and its related sequence-dependent setup time in PS
  Set $EF_{ij} = ES_{ij} + PT_{ij}$
  Determine $\mathrm{EF}^* = \min_{O_{ij} \in \Omega}\{EF_{ij}\}$
  Find the related machine k^* that present EF^*
  Identify the subset of operations $O_{ij} \in \Omega'$ ($\Omega' \subseteq \Omega$) which requires machine k^* and $ES_{ij} < \mathrm{EF}^*$
  Select the operation $O_{ij} \in \Omega'$ which has the highest priority in the *Sequence priority*
  Add the selected operation to PS
  Update the set Ω by removing the selected operation and adding its successor (if there is any)
  If $\Omega = \emptyset$ **Then** set $Stop = True$
Until $Stop = False$
Return PS

Appendix C. Test data

Table C1 presents the parameter levels for random instance generation. Also, general specification of the instance problems utilized for static and dynamic states respectively are presented in Tables C2 and C3.

Table C1. Parameter levels for a random instance generation

Parameters	Values
$\|F\|$	$\{2, 3, 4, 5\}$
$\|K_f\|$	$\{2, 3, \dots, 7\}$
$\|C_{k,f}\|$	$\{2, 3\}$
$\|\mathcal{J}\|$	$\{4, 5, \dots, 15\}$
$\|N_i\|$	Random number (5-15)
t_1	Exponential distribution (100)
Δt_f	Uniform distribution (10-30)
PT^f_{ijkc}	Uniform distribution (40-100)
$ST^f_{c_1,c_2,k}$	Uniform distribution (75-150)
D_i	$rand(2, 5) \times \max_f\{\sum_j \max_{k,c}\{PT^f_{ijkc}\}\}$
(ω^E, ω^T)	$\left(\frac{1}{3}, \frac{2}{3}\right)$
(δ_1, δ_2)	$\left(\frac{1}{3}, \frac{1}{3}\right)$

Table C2. General specification of the instance problems utilized for static state (Subset-S).

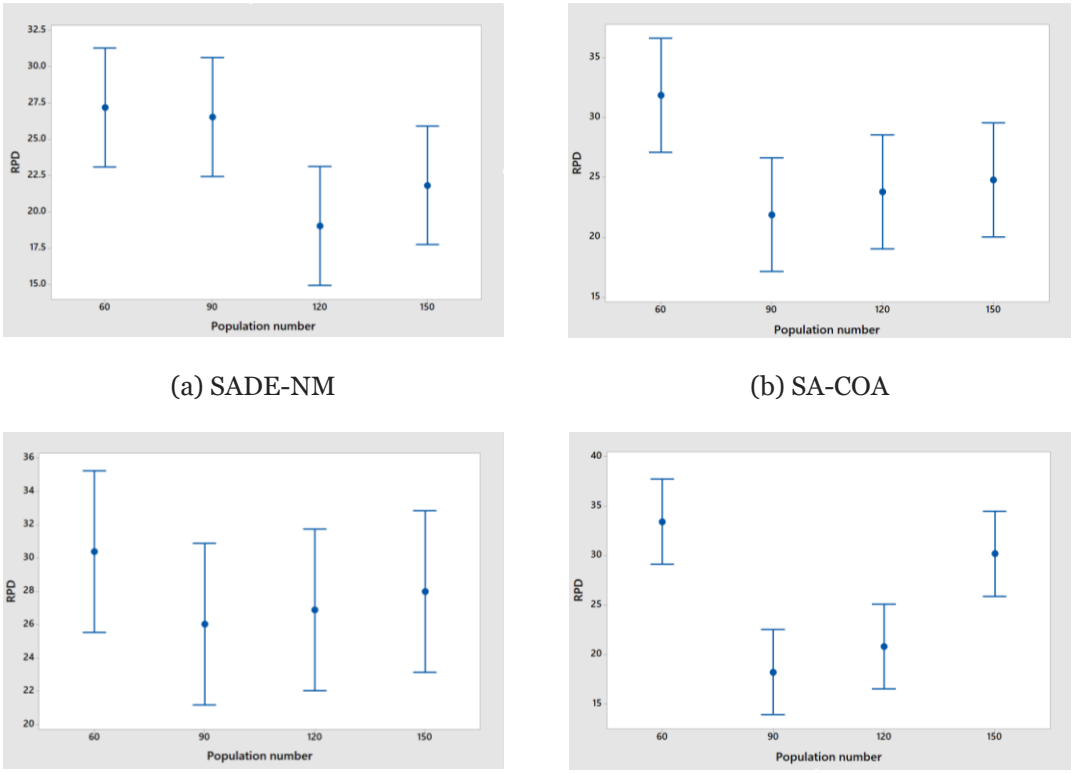
Instance ID	Facilities No. $\|F\|$	RMTs No. per facility	Configurations No. per RMT	Jobs No. $\|\mathcal{J}\|$	Total operations $\|N_i\|$
S1	2	2	2	4	38
S2	2	3	3	5	32
S3	2	4	2, 3	6	54
S4	2	5	3	7	77
S5	3	3	2	7	61
S6	3	4	2, 3	8	86
S7	3	5	3	9	63
S8	3	6	3	10	96
S9	4	4	2	12	111
S10	4	5	3	13	140
S11	4	6	2, 3	14	139
S12	4	7	3	15	152
S13	5	4	2	12	111
S14	5	5	3	13	140
S15	5	6	2, 3	14	139
S16	5	7	3	15	152

Table C3. General specification of the instance problems utilized for dynamic state (Subset-D).

Instance ID	Shopfloor environment	New jobs arriving time: t_1	New jobs No. $\|\mathcal{J}''\|$	Total operations of new jobs
D1	S1	164	1	6
D2	S2	194	1	12
D3	S3	220	2	13
D4	S4	15	2	19
D5	S5	59	1	9
D6	S6	42	1	12
D7	S7	279	2	18
D8	S8	77	2	23
D9	S9	246	2	17
D10	S10	27	2	24
D11	S11	123	3	19
D12	S12	104	3	28
D13	S13	31	2	21
D14	S14	53	2	22
D15	S15	134	3	37
D16	S16	61	3	34

Appendix D. Results of parameter tuning

Fig. D1 depicts the means plot and LSD intervals for the different levels of the population number of the proposed algorithms.



(a) SADE-NM (b) SA-COA

(c) EO (d) SAEO

Fig. D1. Means plot and LSD intervals (at the 95% confidence level) for the different levels of the population number ($nPop$) of the SADE-NM, SA-COA, EO and SAEO algorithms.