International Journal of Engineering

Journal Homepage: www.ije.ir

Flexible Job Shop Scheduling Problem Considering Upper Bounds for the Amount of Interruptions Between Operations and Machines Maintenance Activities

K. Mahdavi[a], M. Mohammadi*[a], F. Ahmadizar[b]

[a] *Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran*
[b] *Department of Industrial Engineering, University of Kurdistan, Sanandaj, Iran*

P A P E R I N F O

Paper history:
Received 03 March 2024
Received iH revised form 26 March 2024
Accepted 02 April 2024

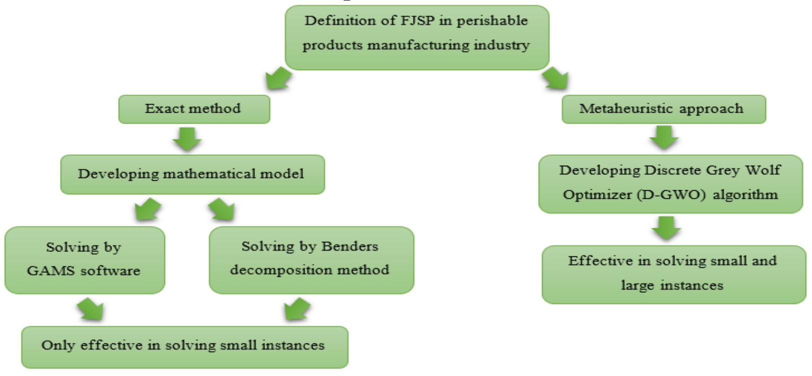
Keywords:
Job Shop Scheduling
Upper Bound For Interuptions
Maintenance Activity
Metaheuristic Algorithm

A B S T R A C T

In modern production environments where perishable products are manufactured in a job shop system, machine reliability is of utmost importance, and delays during job processing are not acceptable. Therefore, it becomes crucial to consider machines maintenance activities and set upper bounds for interruptions between job operations. This paper tackels the Flexible Job Shop Scheduling Problem taking into account these factors. The study is conducted in two phases. Initially, a novel Mixed-Integer Linear Programming (MILP) model is elaborated for the problem and juxtaposed with the Benders decomposition method to assess computational efficiency. Nevertheless, owing to the elevated complexity of the problem, attaining an optimal solution for instances of realistic size poses an exceptionally challenging task using exact methods. Thus, in the second stage, a Discrete Grey Wolf Optimizer (D-GWO) as an alternative approach to solve the problem is proposed. The performance of the extended algorithms is evaluated through numerical tests. The findings indicate that for small instances, the Benders decomposition method outperforms other approaches. Nevertheless, as the instances grow in size, the efficiency of exact methods diminishes, and the Discrete Grey Wolf Optimizer (D-GWO) performs better under such conditions. Overall, this study highlights the importance of considering machines maintenance activities and interruptions in scheduling of job shop for the production of perishable products. The proposed model and Benders decomposition method in small instances, and the metaheuristic algorithm in large instances provide viable solutions.

doi: *10.5829/ije.2024.37.08b.13*

Graphical Abstract


Definition of FJSP in perishable products manufacturing industry
Exact method
Metaheuristic approach
Developing mathematical model
Developing Discrete Grey Wolf Optimizer (D-GWO) algorithm
Solving by GAMS software
Solving by Benders decomposition method
Effective in solving small and large instances
Only effective in solving small instances

1. INTRODUCTION

The job shop scheduling problem (JSP) is a scheduling challenge frequently encountered in different manufacturing contexts. The complexity of scheduling arises from the distinctive constraints and limitations inherent in each manufacturing environment. The perishable products industry is one such sector that

*Corresponding Author Institutional Email: *Mohammadi@khu.ac.ir* (M. Mohammadi) presents its own set of special circumstances and limitations. With perishable products, the product production process is done without delay or with a permissible delay (lower than upper bound delay), and the produced products are packaged and stored immediately. Experiencing significant delays during production can severe consequences and incur high expenses. In production environments where the manufacturing system follows a job shop format, the scheduling problem transforms into a JSP with the additional constraint of maximum delay between operations. Moreover, anticipating potential machine breakdowns makes integrating maintenance into the schedule essential. Therefore, manufacturing perishable products within a job shop system involves navigating numerous restrictions.

Mahdavi et al. (1) in their study examined a particular scheduling problem known as the no wait flexible job shop scheduling problem (FJSP). This involved incorporating various aspects like machine maintenance and processing restrictions. To achieve minimal total lateness for the jobs, they introduced a Mixed-Integer Linear Programming (MILP) model, which was based on a model previously proposed by Gao et al. (2). Nevertheless, given the intricacy of the problem, they also developed an alternative solution approach called the Imperialist Competitive Algorithm (ICA), specifically designed to handle large instances of the problem.

This research takes the problem studied by Mahdavi et al. (1) a step further. Instead of focusing only on no wait scenarios where jobs cannot wait between operations, they consider a more general case. This new formulation allows for an independent maximum waiting time to be defined for each job's operations. Consequently, the no wait problem becomes a specific example within this broader framework, where the maximum waiting time for every job is set to zero. Additionally, the researchers propose a novel approach to formulating the Mixed-Integer Linear Programming (MILP) model, aiming to minimize the total cost associated with both early and late job completions. This approach is inspired by the work of Ozguven et al. (3).

This research explores a FJSP, where the goal is to minimize the combined cost of jobs finishing too early or too late. This problem takes into account three primary constraints:

- An independent upper bound on waiting time between operations of each job
- Periodic machines maintenance activities.
- An independent due date for each job so that if the processing of the job is not completed in the due date, earliness and tardiness fine will be imposed.

The problems associated with the conditions and limitations of the perishable food manufacturing industry, operating within a job shop environment, are the focus of this research. On the one flip side, periodic maintenance activities for machines and the ability to process operations by machines at each stage are a set of common constraints of any job shop production environment. On the other hand, allowing the upper bound for the amount of waiting time between operations of each job is a challenging restriction in the perishable food manufacturing industry. In previous research studies within the realm of FJSP, two scenarios have been explored: one where there is no possibility of delay time between job operations (referred to as no-wait FJSP), and another where there are no limits on the delay time between job operations. In the proposed problem in this research, it is possible to create a delay time between job operations, but an upper bound is defined for each job so that the delay time between its operations does not exceed its upper bound. On the other hand, the objective function focuses on achieving a balanced schedule by minimizing the total deviation from desired completion times for all jobs. That is to say that each job should be processed very close to its due date. Therefore, no wait FJSP is not necessarily suitable, and creating a waiting time between the operations of each job according to its upper bound may help to improve the objective function. Thus, according to the due date of each job, creating a delay time between the operations must be managed in such a way that the least earliness and tardiness penalty is imposed. In the next section, the latest literature will be reviewed.

2. LITERATURE REVIEW

The JSP was initially raised by Manne (4) and Wagner (5). They demonstrated that the JSP, widely recognized as NP-hard problem in literature, is extremely intricate. El Khoukhi et al. (6) proposed a FJSP with restrictions on machine accessibility to minimize the makespan. They suggested a Mixed-Integer Linear Programming (MILP) model and owing to the intricacy of the problem, they proposed a new solution method utilizing the ant nest algorithm. Yousefi Yegane et al. (7) investigated the FJSP to minimize the makespan. Since job splitting is an important technique to reduce completion time, they assumed that preemption is allowed. They proposed a memetic algorithm owing to intricacy of the problem and inability of exact methods in solving large instances.

Benttaleb et al. (8) studied a JSP with two machines, one of them being out of reach during certain periods. In their research, they surveyed the optimality of the Jackson algorithm and developed a exploratory method, and provided upper and lower limit for the problem. Shen et al. (9) examined a FJSP where the setup time required to switch between jobs depends on the specific jobs involved. They developed a Mixed-Integer Linear Programming (MILP) model to find a schedule that minimizes the total time it takes to complete all the jobs. Additionally, they developed a TS that incorporated numerous novel structures to tackle this problem. Tamssaouet et al. (10) focused on a JSP to minimize the makespan in instances that machines are not always accessible. They proposed TS with locality functions and a specific diversity structure.

Caldeira and Gnanavelbabu (11) present an enhanced version of Jaya Algorithm (JA) specifically tailored for solving FJSP with the primary objective of minimizing the makespan. In this study, it is assumed that there are setup times for machines and transfer times between them. Samarghandi (12) researched a no-wait JSP focused on minimizing the makespan while factoring in job due dates. His approach involved transforming the original problem into a related one (CP) and creating models for both. To solve large instances, he designed a Genetic Algorithm (GA). The results demonstrated that the proposed method performed more effectively for the transformed problem (CP) compared to the original JSP formulation.

Zhang et al. (13) offered an enhanced Genetic Algorithm (GA) for a multi goal FJSP by considering machines processing restrictions. Li et al. (14) took the Jaya Algorithm (JA) a step further by creating an improved version specifically designed for FJSP. This enhanced JA incorporates the limitations of machine capabilities within the optimization process. Ying and Lin (15) focused on a no-wait JSP to minimize the makespan. They offered a new solution method. Zhu and Zhou (16) explored a FJSP where jobs have priority constraints and processing times are uncertain, represented as ranges instead of exact values. They introduced a novel optimization method aimed at minimizing the overall duration of the makespan interval. Zhu and Zhou (17) introduced an efficient method called the Grey Wolf Optimizer (GWO) for tackling a complex scheduling problem in FJSP. This problem involves balancing multiple objectives while adhering to specific job priority constraints. Zhang et al. (18) offered an enhanced memetic algorithm designed to solve the FJSP, with the added complexity of factoring in transportation times between machines. Defersha et al. (19) suggested a two-phase Genetic Algorithm (GA) to address a FJSP that involves setup times. In this investigation, machines may not be accessible at all times for processing jobs, and each machine needs a specific cooldown period after finishing an operation before it can handle the next one. Ozolins (20) tackled a no-wait JSP with the goal of minimizing the makespan. Their novel approach utilizes dynamic programming (DP) as an exact solution method. This method effectively solves problems of medium size within a sensible timeframe. Izadi et al. (21) investigated the integration of production and distribution scheduling, allowing outsourcing to minimize total costs. They proposed a mathematical model for small problems and a hybrid Genetic Algorithm (GA) approach for large ones, incorporating dominance properties to find optimal solutions.

Gao et al. (22) investigated a no-wait JSP with due date restrictions. They offered two mathematical models. Then, they suggested a metaheuristics algorithm called RTL-ABC. Boyer et al. (23) investigated a FJSP with hard restrictions such as machine valence, time delays. They offered a Mixed-Integer Linear Programming (MILP) model to tackle this problem and also devised a metaheuristic approach to effectively solve larger instances of the problem.

Torkashvand et al. (24) studied a new three stage production-assembly problem to minimize the maximum completion time of all jobs. They presented a Mixed Integer Linear Programming (MILP) model to solve small instances. Due to high complexity of problem, they developed a new improved Genetic Algorithm (GA) to solve large instances. Valenzuela et al. (25) investigated a no wait JSP and to solve large scales of the problem, they offered a cooperative coevolutionary algorithm. Fan and Su (26) presented a JSP with conveyor-based CFTs. In this investigation, the operations are carried out on the machines that are conjoined in a row through the conveyor. They offered a mathematical model for small instances and a metaheuristic for large instances. Şahman and Korkmaz (27) introduced innovative versions of the Artificial Algae Algorithm (AAA) to address discrete optimization problems. In this research, three encoding strategies were incorporated with AAA to tackle the JSP.

Tutumlu and Saraç (28) investigated a FJSP with job-splitting. They showed that taking into account job splitting in the JSP contributes to identifying improvement opportunities and aligns the problem more closely with real world conditions. In this research, they proposed a MIP model, and for large sizes, a hybrid Genetic Algorithm is presented. Gong et al. (29) investigated a new type of FJSP. According to the simulation of the real world situation, in this context, certain operations within a job do not have specific order restrictions, leading to the proposal of FJSP with discrete operation order flexibility. The objective is to minimize both the makespan and total energy consumption. They presented a model for small instances and improved a memetic algorithm for large instances. Xie et al. (30) delved into a type of scheduling problem called the distributed FJSP, which builds upon the traditional FJSP. They offered a mathematical model for the problem and owing to complexity associated with large instances, they proposed a new algorithm called HGTSA.

Liu et al. (31) investigated a FJSP which is an expansion of the flexible manufacturing. In this research, they presented a mathematical model and suggested an enhanced Genetic Algorithm (GA) with a three-surface encoding strategy. Berterottiere et al. (32) investigated an extension of the FJSP where transportation resources are considered. They extended the classical disjunctive graph model and offered a novel metaheuristic that utilizes a locality function, enabling the exploration of a wide range of moves.

To our best of knowledge, this research is the first to concurrently take into account machines' processing capability, machines' maintenance activities, and an upper limit on the waiting time between job operations in the context of the FJSP. The primary contributions of this paper can be summarized as belows:

- The FJSP in the perishable food manufacturing industry is investigated and a Mixed-Integer Linear Programming (MILP) model, incorporating priority variable, is formulated for the proposed problem. Then, a Benders decomposition method based on the model is offered as an exact method for solving small instances. The supremacy of Benders method in some instances than Gams software shows its high efficiency.
- A modified version of the Grey Wolf Optimizer (D-GWO) is used to tackle larger problems, and its effectiveness is evaluated.
- A comprehensive analysis for the proposed solution methods is presented and their performance, in problems with different sizes, is specified.

The remaining sections of this research will be organized as belows. Section 3 characterizes the problem and presents the mathematical model of the investigated problem. In section 4, the Benders decomposition method is presented to optimally solve small instances of the problem. In section 5, the metaheuristic algorithm (Discrete Grey Wolf Optimizer) is offered to solve large instances. Computational results are mooted in section 6. Finally, conclusions are provided in section 7.

3. PROBLEM DESCRIPTION AND FORMULATIONS

The FJSP considers a scenario with multiple machines and jobs. Each job needs processing on specific machines, but there's a twist! Instead of being assigned to a single machine for each operation, jobs have the option to select from a pool of accessible machines within each stage. This problem involves m stages, each containing parallel machines that can work independently, and n jobs, each consisting of a order of operations with flexible machine choices. The other suppositions of the problem are as follows:

- From the outset, both jobs and machines are readily available for scheduling. However, keep in mind that each machine is limited to handling only one operation at any given time.
- Each job follows a predetermined sequence of operations, outlining the exact steps it takes to be completed.
- The processing path of each job may not involve machines from all stages. This means some jobs might only require a subset of the available stages.
- No multitasking allowed! Each job is restricted to being on one machine at any given time.
- Preemption, or interrupting and resuming operations, is prohibited.
- The gap duration between two successive operations of the same job must be lower than its upper limit of interruption.
- If each job has not been processed on its due date, a penalty is imposed.
- Not all machines are capable of handling every operation. In other words, each machine is capable of processing a certain set of jobs (and not necessarily all of them).
- The machine structure of each stage is parallel and unrelated.
- Because of maintenance activities, each machine must be periodically unaccessible.
- Each inaccessible interval last for a certain period. In other word, the interval length is fixed.
- The objective is to create a realistic schedule that ensures jobs finish as close to their deadlines as possible, without being completed too early or too late.

This section introduces the notations used to define the key components of the model: indices, parameters, and decision variables.

Indices:

i, h: represent the indices of jobs, ranging from 1 to n.

j: denotes the index of operations, ranging from 1 to J_i for a specific job i.

k: represents the index of machines, ranging from 1 to m.

r: is the index of the inaccessibility period.

Parameters:

n: represent the whole number of jobs

m: represent the whole number of machines

J_i: represent the whole number of operations of job i

p_{kij}: represents the processing time of operation o_{ij} if it is carried out on machine k

d_i : denotes the due date of job i

ubw_h : The upper bound of waiting time between operations of job h.

sm_{kr}: denotes the start time of the r-th inaccessibility period on machine k.

fm_{kr}: represents the end time of the r-th inaccessibility period on machine k. $(fm_{kr} - sm_{kr} = T)$

M: represents a large numerical value.

Decision variables:

T_i: tardiness of job i.

E_i: earliness of job i.

V_{ijk}: V_{ijk} takes the value of 1 if operation o_{ij} is performed on machine k; otherwise V_{ijk} is 0.

Z_{ijhgk}: Z_{ijhgk} is 1 if o_{ij} precedes operation o_{hg} on machine k; otherwise Z_{ijhgk} is 0.

s_{ijk}: denotes the start time of operation o_{ij} on machine k

c_{ijk}: represents the completion time of operation o_{ij} on machine k.

c_i: denotes the completion time of job i.

B_{ijkr}: binary variable in unavailability constraints

This section describes a mathematical model for the problem using an approach called the priority variable-based model. This approach relies on a specific type of variable (represented by Z_{ijhgk}) introduced by Manne (4). It is important to note that Z_{ijhgk} being 1 does not necessarily mean o_{ij} comes immediately before o_{hg}. This variable only needs to be defined when i is less than h, because the order of operations within the same job is already fixed. The model presented here utilizes this concept of priority variables to formulate the problem as a Mixed-Integer Linear Programming (MILP).

This modeling approach was initially introduced by Ozguven et al. (3) for formulating the FJSP and we have adopted it for our FJSP as follows:

$$Min \sum_i (E_i + T_i) \tag{1}$$

$$S.T.$$

$$c_i \geq \sum_{k \in M_{ij}} c_{ijk} \quad , \forall\, i, j = J_i \tag{2}$$

$$s_{ijk} + c_{ijk} \leq V_{ijk}.M \quad , \quad \forall\, i, j \quad \forall k \in M_{ij} \tag{3}$$

$$c_{ijk} \geq s_{ijk} + p_{kij} - M.(1 - V_{ijk}) \quad , \forall\, i, j \quad \forall k \in M_{ij} \tag{4}$$

$$s_{ijk} \geq c_{hgk} - M.\ Z_{ijhgk} \quad , \forall\, i \leq h, \forall j, g \quad \forall k \in M_{ij} \cap M_{hg} \tag{5}$$

$$s_{hgk} \geq c_{ijk} - M.(\,1 - Z_{ijhgk}), \quad \forall\, i \leq h, \forall j, g \quad \forall k \in M_{ij} \cap M_{hg} \tag{6}$$

$$\sum_{k \in M_{ij}} s_{ijk} \geq \sum_{k \in M_{ij}} c_{ij-1k} \quad , \quad \forall\, i, j = 2, \ldots, J_i \tag{7}$$

$$\sum_{k \in M_{ij}} s_{ijk} \leq \sum_{k \in M_{ij}} c_{ij-1k} + ubw_i \quad , \forall\, i, j = 2, \ldots, J_i \tag{8}$$

$$\sum_{k \in M_{ij}} V_{ijk} = 1 \quad , \quad \forall\, i, j \tag{9}$$

$$T_i \geq c_i - d_i \quad , \forall\, i \tag{10}$$

$$E_i \geq d_i - c_i \quad , \forall\, i \tag{11}$$

$$s_{ijk} < (sm_{k,r} * V_{ijk}) + M * B_{ijkr} \quad , \forall\, i, j, k, r \tag{12}$$

$$c_{ijk} < (sm_{k,r} * V_{ijk}) + M * B_{ijkr} \quad , \forall\, i, j, k, r \tag{13}$$

$$s_{ijk} < (sm_{k,r} * V_{ijk}) + M(1 - B_{ijkr}) \quad , \forall\, i, j, k, r \tag{14}$$

$$c_{ijk} < (sm_{k,r} * V_{ijk}) + M(1 - B_{ijkr}) \quad , \forall\, i, j, k, r \tag{15}$$

$$s_{ijk} > (fm_{k,r} * V_{ijk}) + M(1 - B_{ijkr}) \quad , \forall\, i, j, k, r \tag{16}$$

$$c_{ijk} > (fm_{k,r} * V_{ijk}) + M(1 - B_{ijkr}) \quad , \forall\, i, j, k, r \tag{17}$$

$$s_{ijk} \geq 0 \quad , \quad c_{ijk} \geq 0 \quad , \forall\, i, j, k \tag{18}$$

$$c_i \geq 0 \quad , \quad T_i \geq 0 \quad , \quad E_i \geq 0, \forall\, i, j, k \tag{19}$$

$$z_{ijhgk} \in \{0,1\} \quad , \forall\, i \leq h, \forall j, g \quad \forall k \in M_{ij} \cap M_{hg} \tag{20}$$

$$V_{ijk} \in \{0,1\} \quad , \quad \forall\, i, j, k \tag{21}$$

$$B_{ijkr} \in \{0,1\} \quad , \quad \forall\, i, j, k, r \tag{22}$$

Equation 1 determines goal function as minimizing whole earliness and tardiness of jobs. Constraint 2 specifies the completion times of the jobs. Constraint 3 describes each operation has beginning time and completion time when is devoted to a machine; If not devoted, both times are set to 0. Constraint 4 ensures the processing time on a machine matches the difference between start and end times. Constraints 5 and 6 prevent two operations from being processed simultaneously on any machine. Constraints 7 and 8 describe the operation precedence constraints and guarantee that the interruption between two successive operations of any job does not exceed its upper bound. Constraint 9 guarantees that each operation is devoted to precisely one machine. Constraints 10 and 11 specifies the tardiness and earliness of each job. Constraints 12 to 17 characterize inaccessibility periods for machines and compel each operation oij be processed between periods when the machine is active. Constraints 18 to 22 characterize the type of decision variables.

4. BENDERS DECOMPOSITION METHOD

Benders decomposition, developed by Benders [33] is a well-established optimization technique frequently used to tackle intricate integer problems and identify optimal solutions. This method divides the problem model into two main problems (including complicated variables such as integers and binaries) and a subproblem (including other variables). In this method, the optimal solution is determined by utilizing upper and lower limits. In minimization problems by solving the master problem (MP), the value of its goal function is considered as a lower bound and to obtain the upper bound, a dual subproblem (DSP) must be solved. Then, the sum of the goal function value of the DSP optimal solution and the goal function value of the MP optimal solution is considered as the upper bound. In the Benders decomposition method, first, the MP, SP, and DSP problems must be identified and the MP problem solved to obtain a lower bound and the values of the complicated variables (minimization problem). Then, by fixing the complicated variables according to the obtained values (in the MP problem), the DSP problem is solved and an upper bound is created for the goal function of the problem. In each step of the method, the discrepancy between the upper and lower bounds is evaluated, and if it is less than a specific value, the algorithm stops and the optimal solution is placed between the upper and lower limit values. If the algorithm does not stop an optimality cutting plane is added to the MP problem according to the DSP solution; thus, the MP problem is solved again and the values of the complicated variables and the lower bound of the problem are obtained. This procedure is repeated to reduce the discrepancy between the upper and lower bounds until this discrepancy falls below a predetermined value. Figure 1 depicts a flowchart of the Benders decomposition method.

As mentioned above, the presented mathematical model is used as the basic model of the Benders decomposition method; so at the beginning, the MP and SP problems will be as follows:

MP:

$$Min\ Z \tag{23}$$

$$\textstyle\sum_k V_{ijk} = 1 \quad , \quad \forall\, i,j \tag{24}$$

$$Constraints\ 3.20\ to\ 3.22 \tag{25}$$

$$Z \geq 0 \tag{26}$$

SP:

$$Min \textstyle\sum_i (E_i + T_i) \tag{27}$$

$$\text{Constraints } 2\ to\ 8 \tag{28}$$

$$\text{Constraints } 10\ to\ 19 \tag{29}$$

In the next step, the dual subproblem is evaluated and the Benders decomposition method is implemented according to the flowchart in Figure 1. According to the flowchart in Figure 1, if the DSP problem is infinite, the corner orientations that lead to the infinity of the DSP problem must be found and eliminated in the DSP problem. In other words, a feasibility cutting plane corresponding to each of the corner orientations must be added to the MP problem.

5. METAHEURISTIC ALGORITHM

The JSP is recognized as an high complexity optimization problem in the field of production scheduling. Since the JSP is a specific type of FJSP, it follows that FJSP is also NP-hard and exhibits high

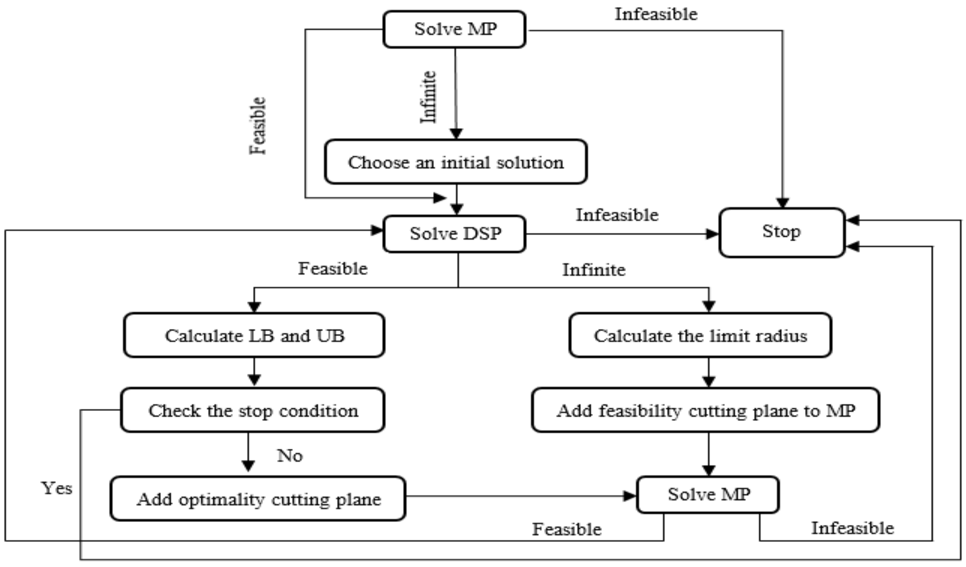


Figure 1. Benders decomposition method flowchart

complexity. In the preceding sections, a mathematical model and Benders decomposition method were presented with the aim of obtaining the optimal solution. Nevertheless, the FJSP is of such complexity that even powerful optimization methods struggle with large instances. To tackle this, a new approach called D-GWO is introduced. This method is designed to rapidly generate good solutions, although not necessarily optimal, even for large and intricate instances.

5. 1. Solution Representation Job-based encoding is like a recipe for this process. Each job is like an ingredient, and its position in the recipe tells you exactly when and on which machine it needs to be processed. This ensures all jobs are completed in the right order, and makes comparing different production schedules like comparing different recipes – you can observe how altering the order of jobs influences the outcome. In this representation, the length of this solution code is the number of jobs (n). Each job position in the code tells you when it's processed on the machines. For instance, in Figure 2, the solution depicts a scenario with three jobs. Each job is comprised of multiple operations and follows a distinct processing route.

Decoding is a crucial step in the process of evaluating the goal function for a given solution to a JSP. The decryption algorithm offered by Brizuela et al. (33) is specifically tailored for problems with no wait constraints, and it likely provides an effective method for transforming the encoded solution into a schedule that can be evaluated to determine the goal function's value. The algorithm consists of the following phases:

Phase 1: Initializing a workless times list for each machine. At the commencement of the schedule, when no jobs have commenced, all machines are entirely workless except for intervals allocated for maintenance tasks.

Phase 2: Handling the operations of the first job that hasn't been processed yet in the encoded solution sequentially.

Phase 3: Updating the list of workless times for each machine.

Phase 4: Verifying whether all jobs are done. If so, the algorithm halts; otherwise, it goes back to step 2.

To provide a clearer understanding of the decoding approach, let's take an example involving 3 jobs and 3 stages. It's worth noting that it's quite fascinating that not all stages of machines may be necessary for processing each job.

For instance, in the stage 1, there are two machines: M_1 and $\overline{M_1}$. In the stage 2, there are two machines: M_2

3	1	2

Figure 2. An encoded solution for a problem with three jobs

and $\overline{M_2}$. Finally, in the stage 3, only machine M_3 is needed. Let's examine the jobs in more detail. The first job comprises two operations. The first operation must be processed on $\overline{M_1}$ in stage 1, while the second operation is assigned to M_3 in stage 3. The second job also consists of two operations. The first operation should be processed on $\overline{M_2}$ in stage 2, and the second operation on $\overline{M_1}$ in stage 1. The third job encompasses three operations. The first operation is processed on M_1 in stage 1, the second operation on M_3 in stage 3, and the third operation on M_2 in stage 2. Let's consider the processing times for these operations. The first and second operations of job 1 require 3 and 1 units of time, respectively. For job 2, the first and second operations need 3 and 4 units of time, respectively. As for job 3, its operations require 1, 2, and 1 units of time, respectively. The due dates for jobs 1, 2, and 3 are 15, 18, and 16, respectively. Additionally, the upper bounds for jobs 1, 2, and 3 are 3, 0, and 5, respectively. There are also periods of unavailability for the machines. Every 5 time units or every 7 time units, the machines undergo preventive maintenance and repair, rendering them unavailable for a period of two time units. Initially, the idle times for the machines are listed in Table 1. Now, let's consider the encrypted solution presented in Figure 2.

The initial step in the decoding process, based on the encoded solution shown in Figure 2, involves processing job 3. To ensure job 3 is completed precisely on time, the timetable displayed in Figure 3 is used for scheduling. Once job 3 is scheduled, the idle times of the machines are updated and recorded in Table 2. Next, the focus shifts to determining the optimal start time for job 1. This ensures that job 1's operations are processed on the required machines and the job is completed as closely as possible to its due date. The scheduling of job 1 is carried out using the schedule illustrated in Figure 4. After scheduling jobs 3 and 1, the idle times of the machines are once again updated and documented in Table 2. Following this, the processing start time for job 2 is examined to ensure that all its operations are completed on different required machines and the job is finished close to its due date. The timetable depicted in Figure 5 is used to schedule job 2. It is observed from Figure 5 that

TABLE 1. List of idle times for machines at the commencement

Machine	Idle time
M_1	$[0,7] \cup [9,16] \cup [18,25] + \ldots$
$\bar{M}_1$	$[0,7] \cup [9,16] \cup [18,25] + \ldots$
M_2	$[0,5] \cup [7,12] \cup [14,19] + \ldots$
$\bar{M}_2$	$[0,7] \cup [9,16] \cup [18,25] + \ldots$
M_3	$[0,5] \cup [7,12] \cup [14,19] + \ldots$

TABLE 2. List of idle times for machines

Machine	**Idle time**	
	after scheduling job 3	**after scheduling job 1 and job 3**
M_1	[0,7] ∪ [10,16] ∪ [18,25] + …	[0,7] ∪ [10,16] ∪ [18,25] + …
$\bar{M}_1$	[0,7] ∪ [9,16] ∪ [18,25] + …	[0,7] ∪ [12,16] ∪ [18,25] + …
M_2	[0,5] ∪ [7,12] ∪ [14,15] ∪ [16,19] + …	[0,5] ∪ [7,12] ∪ [14,15] ∪ [16,19] + …
$\bar{M}_2$	[0,7] ∪ [9,16] ∪ [18,25] + …	[0,7] ∪ [9,16] ∪ [18,25] + …
M_3	[0,5] ∪ [7,10] ∪ [14,19] + …	[0,5] ∪ [7,10] ∪ [15,19] + …

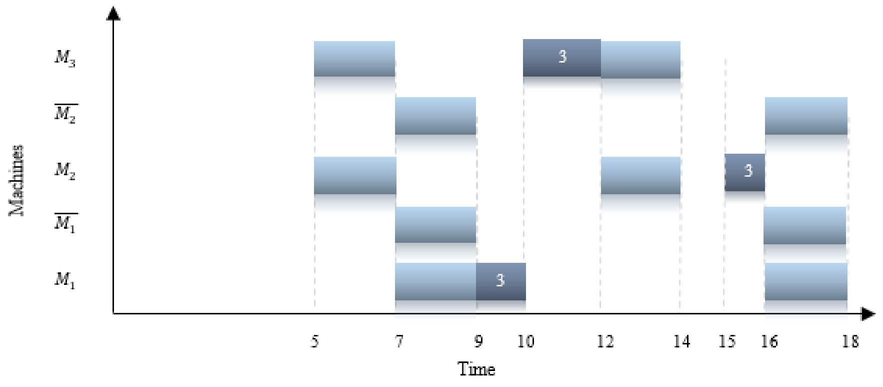


Figure 3. Schedule for job 3 based on the provided encoded solution

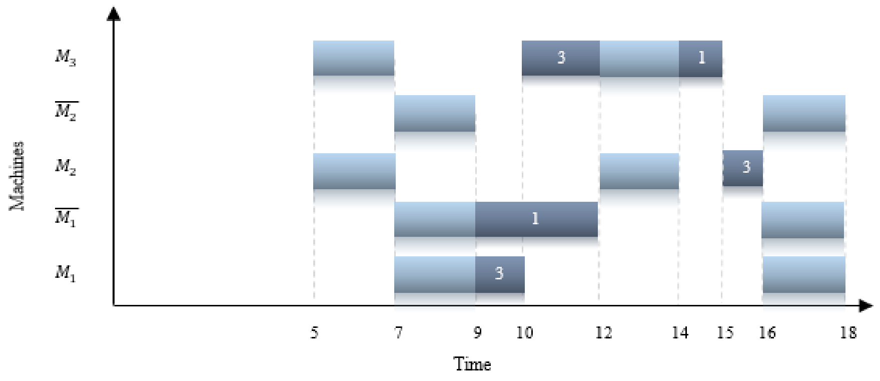


Figure 4. Schedule for job 3 and job 1 based on the provided encoded solution

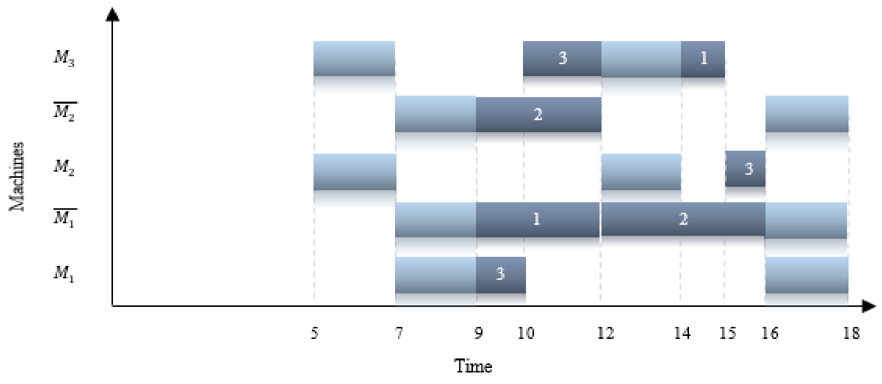


Figure 5. Schedule for all jobs based on the provided encoded solution

job 2 has an upper bound of zero, indicating no interruption is permitted between its operations. In order to optimize the objective function, job 2 is scheduled to commence at time unit 9, aligning with its due date. Once all the jobs have been processed, the decoding algorithm is terminated, and the schedule from Figure 5 is selected as the final schedule for the encoded solution presented in Figure 2.

5. 2. Discrete Grey Wolf Optimizer (D-GWO) The Grey Wolf Optimizer (GWO), introduced by Mirjalili et al. (34), is a population-based metaheuristic algorithm that has gained popularity for solving continuous optimization problems. This algorithm is inspired by grey wolves and this inspiration comes from both their social structure (living) and hunting behavior in their natural habitat. Grey wolves are known to live in groups of 5 to 12 individuals and exhibit a strict social structure. The group is led by alpha wolves, who hold a crucial role in decision-making. The rest of the pack follows the decisions made by the alpha wolves. Beta wolves occupy the next level in the hierarchy and follow the commands of the alpha wolves, relaying those commands to the rest of the group. At the lowest level, omega wolves provide protection and submit to the dominant wolves. Wolves that do not fit into these categories are classified as delta wolves, following the alpha and beta wolves while having dominance over the omega wolves. An intriguing behavior observed in wolves is group hunting, which involves three primary stages: encircling the prey, hunting, and attacking the prey. The first two stages, encircling the prey and hunting, are focused on exploring the search space. The final stage, attacking the prey, emphasizes exploitation.

In the process of encircling the prey, let's consider $X(\mathrm{t})$ as the position of a wolf and $X_p(\mathrm{t})$ as the position of the prey at the current iteration of the algorithm. This process can be represented by Equations 30 to 33 (34):

$$(t+1) = X_p(t) - A.D(t) \tag{30}$$

$$D(t) = \left|C.X_p(t) - X(t)\right| \tag{31}$$

$$A = \alpha(2r_1 - 1) \tag{32}$$

$$C = 2r_2 \tag{33}$$

Here, C and A are vectors, and r_1 and r_2 are random numbers ranging from 0 to 1. The value of α gradually decreases from 2 to 0.

As depicted in Equation 30, wolves reduce their distances from the prey ($\mathrm{X_p(t)}$). This distance is influenced by two factors: A, which gradually diminishes, and D, which signifies the distance from the prey's location. With the progression of the algorithm's iteration count, the wolves draw nearer to the prey. This behavior enables them to encircle the prey, given that their initial locations are randomly determined.

During the hunting process, the alpha wolf takes the lead within the group, with the beta and delta wolves also potentially participating. The position of the prey (optimal point) is denoted by $\mathrm{X_p(t)}$, and it is usually unknown. Despite the unknown location, it's assumed that all three wolves have a good understanding of where the prey might be. The GWO mimics how wolves collaborate during a hunt to find prey. It maintains a virtual pack of potential solutions, with alpha, beta, and delta representing the leaders who have an intuitive understanding of the optimal solution (the prey). These leaders' positions are constantly updated and stored. The remaining wolves (other potential solutions) don't have this innate knowledge. Instead, they strategically adjust their positions by considering the positions of alpha, beta, and delta, employing Equations 34 to 37 (34).

$$\begin{aligned} X_1(t) &= X_\alpha(t) - A_1.D_\alpha(t) \\ D_\alpha(t) &= |C_1.X_\alpha(t) - X(t)| \end{aligned} \tag{34}$$

$$\begin{aligned} X_2(t) &= X_\beta(t) - A_2.D_\beta(t) \\ D_\beta(t) &= \left|C_2.X_\beta(t) - X(t)\right| \end{aligned} \tag{35}$$

$$\begin{aligned} X_3(t) &= X_\delta(t) - A_3.D_\delta(t) \\ D_\delta(t) &= |C_3.X_\delta(t) - X(t)| \end{aligned} \tag{36}$$

$$X(t+1) = \big(X_1(t) + X_2(t) + X_3(t)\big)/3 \tag{37}$$

In Equation 37, the updated position of a wolf $X(t+1)$ is determined as the average of the positions of the alpha, beta, and delta wolves.

As previously mentioned, the GWO is commonly utilized for continuous optimization problems. However, in recent times, researchers have made efforts to adapt it for discrete optimization problems due to the significant achievements of the traditional GWO in continuous optimization. One such adaptation is the Discrete Grey Wolf Optimizer (D-GWO), introduced by Hosseini Shirvani (35). This new version tackles the challenge by introducing Binary operators and Walking Around approache. Binary operators handle the discrete nature of the problem, allowing the algorithm to work with specific values instead of continuous ranges. Walking around helps the algorithm maintain a balance between exploring the entire search space (exploration) and focusing on promising areas (exploitation). Additionally, Jiang et al. (36) have developed another variant of the D-GWO algorithm with the objective of minimizing the makespan. This variant is specifically designed to tackle two combinatorial optimization problems in the manufacturing domain: JSP and FJSP. In this algorithm, a searching operator based on crossover operations is utilized to ensure the algorithm operates within the discrete domain. Furthermore, an adaptive mutation method is introduced to maintain population diversity and prevent premature convergence. For the purpose of this study, the D-GWO presented by Hosseini Shirvani (35) is employed to solve large instances of the proposed FJSP. Below, you will find further details regarding the D-GWO.

5. 2. 1. Wolf Representation As indicated in section 4-1, the algorithm proposed herein represents each solution (wolf) through a sequence, dictating the order of job execution on the machines. In a scenario involving n jobs, this representation comprises a sequence with n elements, ensuring that each job appears precisely once in the sequence. This guarantees that every job is accounted for and executed within the solution.

5. 2. 2. Binary Vectors and Operators The path taken by an individual wolf towards its prey is guided by the three top-ranking wolves, namely alpha (W_α), beta (W_β), and delta (W_δ). Consequently, novel binary vectors and operators are introduced to leverage the collective knowledge of these leader wolves concerning the traversed discrete search space. This involves the use of binary vectors $\mathrm{Token_i}$ and $\mathrm{Adjuster_i}$ for comparing each wolf with the leader wolves. To execute this, each wolf needs to rearrange jobs in its representation to align with the leaders. In the vector, a zero value indicates that the corresponding job does not need to be changed. It's worth noting that the first and last jobs are exempt from alterations, denoted by being set to zero in the Token vector.

In the proposed algorithm, the movement of an individual wolf towards its prey is guided by the three leading wolves: alpha (W_α), beta (W_β), and delta (W_δ). To leverage the knowledge of these leader wolves about the traversed discrete search space, new binary vectors and operators are introduced. Two binary vectors, $\mathrm{Token_i}$ and $\mathrm{Adjuster_i}$, are utilized for comparison between each wolf and the leader wolves. The purpose is to ensure that each wolf relocates jobs in its representation similar to the leaders. The value of zero in the Token vector signifies that the corresponding job does not need to be changed. However, the first and last jobs are exempted from any changes, so they are set to zero in the Token vector. During initialization, all jobs are considered for potential changes, resulting in an initial Token vector value of one. The operator\ is utilized to signify the differences in corresponding jobs between two wolves (35). For example, let's consider the case where n=6, and $\mathrm{Token_1} = (0,1,1,0,1,0)$ for W_1, and $\mathrm{Token_\alpha} = (0,0,1,1,1,0)$ for W_α. The operation $\mathrm{Token_1} \setminus \mathrm{Token_\alpha}$ yields:

$$Token_1 \setminus Token_\alpha = (0,1,1,0,1,0) \setminus (0,0,1,1,1,0) = (0,1,0,1,0,0)$$

As shown, the output bit remains zero for the same positions since no changes are necessary (35). Another operator, "⊗", is used to determine if a bit should be changed based on the corresponding Adjuster value. If $\mathrm{Token_1} = (0,1,1,0,0,0)$ and $\mathrm{Adjuster_1} = (0,0,1,0,1,0)$, the operation $\mathrm{Token_1} \otimes \mathrm{Adjuster_1}$ results in:

$$Token_1 = (0,1,1,0,0,0) \otimes (0,0,1,0,1,0) = (0,1,0,0,1,0)$$

The Adjuster vector serves as a guide for the Token vector to prevent duplicate changes on specific jobs (35).

5. 2. 3. Description of Customized D-GWO Algorithm parameters are divided into two categories:
1. General Parameters:
• Population size (PS): This parameter determines the number of wolves in each iteration of the algorithm.
• Maximum iterations (MI): This sets the limit on how many times the algorithm will repeat its calculations. It's like the maximum number of attempts the wolves have to find the prey.
2. Instance-related parameters:
• Quantity of jobs (n)
• Count of stages and machines within each stage
• Route of job processing
• Capability of each machine
• Processing time for each job on every machine
• Intervals for maintenance
• Job due dates
• Upper bound for waiting time between operations

D-GWO Algorithm:

Step 1: Generate the initial population of solutions (wolves) equal to PS and set $Z = 1$.

Step 2: Generate binary vectors of Token and Adjuster for each wolf:

$$Token_i = (b_{i1}, b_{i2}, \dots, b_{in}) = \vec{1}$$

$$Adjuster_i = (a_{i1}, a_{i2}, \dots, a_{in}) = \vec{1}$$

Step 3: Calculate the value of the goal function of each wolf according to the method of section 4-1.

The top-performing solution is identified as the alpha wolf (W_α). The second-best solution is designated as the beta wolf (W_β). The third-best solution is recognized as the delta wolf (W_δ).

Step 4: if $Z \le MI$ then set $i = 1$ and proceed to step 5, otherwise ($Z > MI$) proceed to step 10.

Step 5: if $i \le PS$ then proceed to step 6, otherwise ($i > PS$) proceed to step 9.

Step 6: In the exploration phase, call algorithm EXP for encircling the prey (update the position W_i).

Step 7: In the exploitation phase, call algorithm SWP for Swap mutation (update the position W_i).

Step 8: Calculate the value of the goal function of wolf W_i according to the method of section 4-1. If there is a change in the alpha, beta, and delta wolves, update them and set $i = i + 1$ and move on to step 5.
Step 9: Add one to the number of iterations of the algorithm ($Z = Z + 1$) and go to step 4.
Step 10: Among the solutions in the last iteration (final wolves), choose the best wolf (W_α) and set it as the final solution for the problem.

EXP Algorithm:
The EXP algorithm is employed to adjust the position of each wolf (W_i) and takes the following parameters as input:
• $W_i , W_\alpha , W_\beta , W_\delta$
• $Token_i , Token_\alpha , Token_\beta , Token_\delta$
• $Adjuster_i , Adjuster_\alpha , Adjuster_\beta , Adjuster_\delta$
The procedure of the algorithm unfolds as belows:
Step 1: Calculate the F_1 , F_2 , F_3:
• F_1 is equal to the value of the goal function of W_α
• F_2 is equal to the value of the goal function of W_β
• F_3 is equal to the value of the goal function of W_δ
• $F_1 \le F_2 \le F_3$
Step 2: Calculate the P_1 , P_2 , P_3:

$$P_1 = \frac{F_1}{F_1+F_2+F_3}$$

$$P_2 = \frac{F_2}{F_1+F_2+F_3} \qquad \begin{cases} P_1 \le P_2 \le P_3 \\ P_1 + P_2 + P_3 = 1 \end{cases}$$

$$P_3 = \frac{F_3}{F_1+F_2+F_3}$$

Step 3: Calculate the $TokenDiff_1$, $TokenDiff_2$, $TokenDiff_3$ and set $j = 1$:

$$TokenDiff_1 = Token_i \setminus Token_\alpha$$

$$TokenDiff_2 = Token_i \setminus Token_\beta$$

$$TokenDiff_3 = Token_i \setminus Token_\delta$$

Step 4: Choose a random number in $(0 , 1)$ as q:
• If $q < P_1$ then $Adjuster_i(j)$ is obtained as $Adjuster_i(j) = TokenDiff_1(j)$
• If $P_1 \le q < P_2$ then $Adjuster_i(j)$ is obtained as $Adjuster_i(j) = TokenDiff_2(j)$
• If $P_2 \le q \le P_3$ then $Adjuster_i(j)$ is obtained as $Adjuster_i(j) = TokenDiff_3(j)$
Step 5: If $j = n$, then move on to step 6, otherwise Add one to j ($j = j + 1$) and move on to step 4.
Step 6: make change on $Adjuster_i$ vector as follow:
$Adjuster_i(1) = Adjuster_i(n) = 0$
Step 7: Revise W_i to encircle the prey by adjusting its trajectory according to $Adjuster_i$:
Exchange job j and job k where $Adjuster_i(j) = Adjuster_i(k) = 1$ and $j \ne k$.
Step 8: Consider W_i as output of algorithm.

SWP Algorithm:
Algorithm SWP is used to make swap mutation on each wolf and receives W_i and n as inputs.
Below are the outlined steps of the algorithm:
Step 1: Choose two random integer number in $(1 , n)$ as j and k.
Step 2: Set $a = W_i(j)$ and $b = W_i(k)$
Step 3: make change on W_i as follow:
$W_i(j) = b \ , \ W_i(k) = a$
Step 4: Consider W_i as output of algorithm.

6. COMPUTATIONAL RESULTS

This research uses GAMS software, equipped with the CPLEX solver, to tackle a Mixed-Integer Linear Programming (MILP) model. Each instance of the problem is given a maximum of two hours (7200 seconds) to find the optimal solution. If an optimal solution is found within this time limit, it is reported. The D-GWO algorithm, along with its components, is implemented in Python. All computational experiments are carried out on a laptop equipped with an Intel Core i7-4600U CPU running at 2.10 GHz. The laptop is equipped with 8GB of memory and runs on the Windows 10 operating system. To fine-tune the parameters of the D-GWO algorithm, various values are considered for each parameter. Through initial experiments and solving numerical problems, the appropriate values are selected. Specifically, the PS is set to $10 \times n$, and the MI is set to 100 (n indicates the number of jobs).

In the research, the D-GWO algorithm begins by randomly generating an initial population of solution candidates. Since the algorithm is random in nature, each problem is solved 5 times by applying D-GWO. The goal function values for the obtained solutions are then reported, including the minimum, average, and maximum values. To facilitate result comparison, a Relative Percentage Deviation (RPD) is employed to gauge the goal function value for each solution. The RPD is computed using the formula in Equation 38.

$$RPD = \left(\frac{f(sol) - f(sol_{best})}{f(sol_{best})}\right) \times 100 \qquad (38)$$

In this formula, sol_{best} denotes the best solution acquired for a specific problem from the solutions obtained. The RPD calculates the percentage difference between a solution and the best solution, providing a relative measure of how far each solution is from the optimal value. It helps in comparing the performance of different solutions and identifying how close they are to the best possible outcome.

The considered problem can be examined from two points of view: If the upper bound of waiting time is zero, it implies that the operations of each job must be executed consecutively without any breaks. In this case, the problem transforms into a no wait FJSP with additional constraints. On the other hand, if the upper bound of waiting time is greater than zero, it allows for the consideration of waiting time between the operations of each job. This introduces flexibility in scheduling and provides an opportunity to utilize the gap time between operations effectively. Since the objective function focuses on achieving a balanced schedule by minimizing the total deviation from desired completion times for all jobs, it is crucial to make effective use of the waiting time between operations. By strategically scheduling the operations and utilizing the gap time, it is possible to improve the objective function and minimize the penalties associated with earliness and tardiness. Therefore, in the latter scenario where waiting time is allowed, optimizing the schedule to minimize the total earliness and tardiness can be achieved by effectively managing the gap time between operations.

Table 3 showcases the results obtained from solving various instances. In each instance, some jobs have UBW ≠ 0 therefor waiting time between their operations can be considered, and some of them must be processed as no wait (UBW = 0) and their operations must processed in a row without any waiting time. According to the information provided in Table 3, the RPD values for instances where either the mathematical model or Benders decomposition method found the optimal solution are highlighted in bold. In all instances enumerated in Table 3, each job necessitates precisely one machine from each stage for processing. For small problems with a maximum of 7 jobs, both GAMS and Benders decomposition find optimal solutions within a reasonable timeframe. Additionally, the D-GWO algorithm demonstrates excellent performance by consistently providing the optimal solution every 5 times. The Benders decomposition method proves to be advantageous over Gams software, particularly in instances 9 and 10, where Gams software fails to guarantee an optimal solution within 7200 seconds. In contrast, the Benders decomposition method not only offers the optimal solution but also provides a suitable lower bound for instances 11 to 15. In large instances, both the Benders decomposition method and Gams software lose their efficiency, but the D-GWO algorithm performs remarkably well by delivering a suitable solution within a reasonable time. The solutions obtained from the D-GWO algorithm are reliable as it consistently provides the optimal solution in small instances, similar to exact methods. The algorithm's convergence is confirmed by the minimal deviation of its provided solutions for each instance. Overall, the results indicate that the proposed D-GWO algorithm performs effectively and can produce satisfactory solutions within a reasonable timeframe, particularly for the larger instances of the problem.

TABLE 3. Comparison results among the MILP model, Benders method, and D-GWO

Instance	**No. job**	**No. stages**	**GAMS**		**Benders**		**D-GWO**			
			Output (RPD)	**Run time (s)**	**Output (RPD)**	**Run time(s)**	**Output (RPD)**			**Run time (s)**
							Min	**Avg**	**Max**	
1	5	5	**0**	2011	**0**	1839	**0**	**0**	**0**	53
2	5	7	**0**	2894	**0**	2388	**0**	**0**	**0**	59
3	5	9	**0**	3362	**0**	2811	**0**	**0**	**0**	55
4	6	5	**0**	4251	**0**	3634	**0**	**0**	**0**	76
5	6	7	**0**	4984	**0**	3893	**0**	**0**	**0**	67
6	6	9	**0**	5375	**0**	4059	**0**	**0**	**0**	81
7	7	5	**0**	6311	**0**	4507	**0**	**0**	**0**	95
8	7	7	**0**	6798	**0**	5113	**0**	**0**	**0**	107
9	7	9	**0**	7200	**0**	5418	**0**	**0**	**0**	118
10	8	5	**0**	7200	**0**	5924	**0**	**0**	**0**	109
11	8	7	0.03	7200	0 *LB*	6469	0	0	0	143
12	8	9	0.05	7200	0 *LB*	6713	0	0.01	0.04	151
13	9	5	0.14	7200	0 *LB*	6753	0	0.02	0.04	142
14	9	7	0.19	7200	0.03 *LB*	6854	0	0.02	0.05	159
15	9	9	0.26	7200	0.07 *LB*	6831	0	0.05	0.08	191
16	10	5	0.37	7200	-	-	0	0.03	0.15	185

17	10	7	0.35	7200	-	-	0	0.03	0.10	229
18	10	9	0.38	7200	-	-	0	0.08	0.12	217
19	15	5	1.08	7200	-	-	0	0.36	0.52	384
20	15	7	1.36	7200	-	-	0	0.27	0.63	415
21	15	9	1.98	7200	-	-	0	0.39	0.84	430
22	20	5	3.12	7200	-	-	0	0.51	1.38	576
23	20	7	4.02	7200	-	-	0	0.76	1.65	558
24	20	9	4.31	7200	-	-	0	1.11	1.78	603
25	25	5	-	7200	-	-	0	1.32	1.93	753
26	25	7	-	7200	-	-	0	1.20	2.15	798
27	25	9	-	7200	-	-	0	1.07	2.01	774
28	30	5	-	7200	-	-	0	1.29	2.28	879
29	30	7	-	7200	-	-	0	1.16	1.92	961
30	30	9	-	7200	-	-	0	1.19	2.18	994
Average			0.73	6299	-	-	0	0.36	0.66	345.4

7. CONCLUSIONS

This paper focuses on the FJSP that incorporates machines' maintenance activities and an upper bound for interruption between job operations. The objective function focuses on achieving a balanced schedule by minimizing the total deviation from desired completion times for all jobs. This problem has significant applications in the production of perishable products. In one hand, delays in production can significantly impact the freshness and quality of these products, potentially leading to spoilage and wasted resources. In other hand, early completion can also be detrimental if it leads to products reaching their expiration date before they can be sold or consumed. Production processes in such environments often involve job shop scheduling, where production is carried out without delay or with a permissible delay (lower than the upper bound delay), and the products are promptly packaged and stored. To address this issue, the researchers proposed a Mixed-Integer Linear Programming (MILP) model and compared it with the Benders decomposition method. The Benders decomposition method demonstrated superior quality compared to the model. However, exact methods were found to be ineffective in solving large instances of the problem. Therefore, the researchers developed a D-GWO as an alternative approach. Both the D-GWO and the exact methods exhibited good performance in solving small instances of the problem.

Moreover, the D-GWO proved capable of solving real-sized instances and demonstrated favorable performance in terms of both solution quality and runtime.

In future studies, we could look into letting operations be paused and restarted, as long as the break between pauses is shorter than the allowed maximum for that particular job. Additionally, including immediate repairs for unexpected problems would make the situation more realistic.

8. REFERENCES

1. Mahdavi K, Mohammadi M, Ahmadizar F. Efficient scheduling of a no-wait flexible job shop with periodic maintenance activities and processing constraints. Journal of Quality Engineering and Production Optimization. 2023. https://doi.org/10.22070/JQEPO.2023.16882.1246
2. Gao J, Gen M, Sun L. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. Journal of Intelligent Manufacturing. 2006;17:493-507. https://doi.org/10.1007/s10845-005-0021-x
3. Özgüven C, Özbakır L, Yavuz Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. Applied Mathematical Modelling. 2010;34(6):1539-48. https://doi.org/10.1016/j.apm.2009.09.002
4. Manne AS. On the job-shop scheduling problem. Operations research. 1960;8(2):219-23. https://doi.org/10.1287/opre.8.2.219
5. Wagner HM. An integer linear-programming model for machine scheduling. Naval research logistics quarterly. 1959;6(2):131-40. https://doi.org/10.1002/nav.3800060205
6. El Khoukhi F, Boukachour J, Alaoui AEH. The “Dual-Ants Colony”: A novel hybrid approach for the flexible job shop scheduling problem with preventive maintenance. Computers & Industrial Engineering. 2017;106:236-55. https://doi.org/10.1016/j.cie.2016.10.019
7. Yegane BY, Kamalabadia IN, Khanlarzadeb N. Critical path method for lot streaming problem in flexible job shop environment. International Journal of Engineering-Transactions B: Applications. 2017;30(2):261-9. https://doi.org/10.5829/idosi.ije.2017.30.02b.13
8. Benttaleb M, Hnaien F, Yalaoui F. Two-machine job shop problem under availability constraints on one machine: Makespan minimization. Computers & Industrial Engineering. 2018;117:138-51. https://doi.org/10.1016/j.cie.2018.01.028

9. Shen L, Dauzère-Pérès S, Neufeld JS. Solving the flexible job shop scheduling problem with sequence-dependent setup times. European journal of operational research. 2018;265(2):503-16. https://doi.org/10.1016/j.ejor.2017.08.021

10. Tamssaouet K, Dauzère-Pérès S, Yugma C. Metaheuristics for the job-shop scheduling problem with machine availability constraints. Computers & Industrial Engineering. 2018;125:1-8. https://doi.org/10.1016/j.cie.2018.08.008

11. Caldeira RH, Gnanavelbabu A. Solving the flexible job shop scheduling problem using an improved Jaya algorithm. Computers & Industrial Engineering. 2019;137:106064. https://doi.org/10.1016/j.cie.2019.106064

12. Samarghandi H. Solving the no-wait job shop scheduling problem with due date constraints: A problem transformation approach. Computers & Industrial Engineering. 2019;136:635-62. https://doi.org/10.1016/j.cie.2019.07.054

13. Zhang G, Hu Y, Sun J, Zhang W. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. Swarm and evolutionary computation. 2020;54:100664. https://doi.org/10.1016/j.swevo.2020.100664

14. Li J-q, Deng J-w, Li C-y, Han Y-y, Tian J, Zhang B, et al. An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. Knowledge-Based Systems. 2020;200:106032. https://doi.org/10.1016/j.knosys.2020.106032

15. Ying K-C, Lin S-W. Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm. Computers & Industrial Engineering. 2020;146:106615. https://doi.org/10.1016/j.cie.2020.106615

16. Zhu Z, Zhou X. Flexible job-shop scheduling problem with job precedence constraints and interval grey processing time. Computers & Industrial Engineering. 2020;149:106781. https://doi.org/10.1016/j.cie.2020.106781

17. Zhu Z, Zhou X. An efficient evolutionary grey wolf optimizer for multi-objective flexible job shop scheduling problem with hierarchical job precedence constraints. Computers & Industrial Engineering. 2020;140:106280. https://doi.org/10.1016/j.cie.2020.106280

18. Zhang G, Sun J, Lu X, Zhang H. An improved memetic algorithm for the flexible job shop scheduling problem with transportation times. Measurement and Control. 2020;53(7-8):1518-28. https://doi.org/10.1177/0020294020948094

19. Defersha FM, Rooyani D. An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. Computers & Industrial Engineering. 2020;147:106605. https://doi.org/10.1016/j.cie.2020.106605

20. Ozolins A. A new exact algorithm for no-wait job shop problem to minimize makespan. Operational Research. 2020;20(4):2333-63. https://doi.org/10.1007/s12351-018-0414-1

21. Izadi L, Ahmadizar F, Arkat J. A hybrid genetic algorithm for integrated production and distribution scheduling problem with outsourcing allowed. International Journal of Engineering, Transactions B: Applications,. 2020;33(11):2285-98. https://doi.org/10.5829/IJE.2020.33.11B.19

22. Gao J, Zhu X, Bai K, Zhang R. New controllable processing time scheduling with subcontracting strategy for no-wait job shop problem. International Journal of Production Research. 2022;60(7):2254-74. https://doi.org/10.1080/00207543.2021.1886368

23. Boyer V, Vallikavungal J, Rodríguez XC, Salazar-Aguilar MA. The generalized flexible job shop scheduling problem. Computers & Industrial Engineering. 2021;160:107542. https://doi.org/10.1016/j.cie.2021.107542

24. Torkashvand M, Ahmadizar F, Farughi H. Distributed production assembly scheduling with hybrid flowshop in assembly stage. International Journal of Engineering, Transactions B: Applications,. 2022;35(5):1037-55. https://doi.org/10.5829/IJE.2022.35.05B.19

25. Valenzuela-Alcaraz VM, Cosio-Leon M, Romero-Ocaño AD, Brizuela CA. A cooperative coevolutionary algorithm approach to the no-wait job shop scheduling problem. Expert Systems with Applications. 2022;194:116498. https://doi.org/10.1016/j.eswa.2022.116498

26. Fan H, Su R. Mathematical modelling and heuristic approaches to job-shop scheduling problem with conveyor-based continuous flow transporters. Computers & Operations Research. 2022;148:105998. https://doi.org/10.1016/j.cor.2022.105998

27. Şahman MA, Korkmaz S. Discrete artificial algae algorithm for solving job-shop scheduling problems. Knowledge-Based Systems. 2022;256:109711. https://doi.org/10.1016/j.knosys.2022.109711

28. Tutumlu B, Saraç T. A MIP model and a hybrid genetic algorithm for flexible job-shop scheduling problem with job-splitting. Computers & Operations Research. 2023;155:106222. https://doi.org/10.1016/j.cor.2023.106222

29. Gong G, Tang J, Huang D, Luo Q, Zhu K, Peng N. Energy-efficient flexible job shop scheduling problem considering discrete operation sequence flexibility. Swarm and Evolutionary Computation. 2024;84:101421. https://doi.org/10.1016/j.swevo.2023.101421

30. Xie J, Li X, Gao L, Gui L. A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. Journal of Manufacturing Systems. 2023;71:82-94. https://doi.org/10.1016/j.jmsy.2023.09.002

31. Liu Z, Zha J, Yan J, Zhang Y, Zhao T, Cheng Q, et al. An improved genetic algorithm with an overlapping strategy for solving a combination of order batching and flexible job shop scheduling problem. Engineering Applications of Artificial Intelligence. 2024;127:107321. https://doi.org/10.1016/j.engappai.2023.107321

32. Berterottière L, Dauzère-Pérès S, Yugma C. Flexible job-shop scheduling with transportation resources. European Journal of Operational Research. 2024;312(3):890-909. https://doi.org/10.1016/j.ejor.2023.07.036

33. Brizuela CA, Zhao Y, Sannomiya N, editors. No-wait and blocking job-shops: Challenging problems for GA's. 2001 IEEE International Conference on Systems, Man and Cybernetics e-Systems and e-Man for Cybernetics in Cyberspace (Cat No 01CH37236); 2001: IEEE.

34. Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. Advances in engineering software. 2014;69:46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007

35. Shirvani H. A novel discrete grey wolf optimizer for scientific workflow scheduling in heterogeneous cloud computing platforms. Scientia Iranica. 2022;29(5):2375-93. https://doi.org/10.24200/SCI.2022.57262.5144

36. Jiang T, Zhang C. Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. Ieee Access. 2018;6:26231-40. https://doi.org/10.1109/ACCESS.2018.2833552

Persian Abstract

چکیده

در محیط‌های تولید مدرن که محصولات فاسد شدنی در یک سیستم کار کارگاهی تولید می‌شوند، قابلیت اطمینان ماشین‌آلات از اهمیت بالایی برخوردار است و تأخیر در پردازش کار قابل قبول نیست. بنابراین، در نظر گرفتن فعالیت‌های تعمیر و نگهداری ماشین‌آلات و تعیین مرزهای بالایی برای وقفه بین عملیات کاری بسیار مهم است. این مقاله با در نظر گرفتن این عوامل به مسئله زمان‌بندی کار کارگاهی انعطاف‌پذیر (FJSP) می‌پردازد. مطالعه در دو فاز انجام می شود. در مرحله اول، یک مدل ریاضی جدید برای مسئله مورد بررسی توسعه داده شده و از نظر کارایی محاسباتی با روش تجزیه Benders مقایسه می‌شود. با این حال، دستیابی به یک راه حل بهینه برای مسائل با اندازه واقعی با یک روش دقیق به دلیل ساختار NP-hard آن بسیار دشوار است. بنابراین، در مرحله دوم، یک الگوریتم بهینه‌ساز گرگ خاکستری گسسته (D-GWO) برای حل این مسأله پیشنهاد شده است. آزمایش‌های عددی برای ارزیابی عملکرد الگوریتم‌های توسعه‌یافته مورد استفاده قرار گرفته‌اند. یافته‌ها نشان می‌دهند که در نمودهای کوچک، روش تجزیه Benders عملکرد بهتری دارد، اما با افزایش اندازه نمودها، روش‌های دقیق کارایی خود را از دست می‌دهند و الگوریتم D-GWO در این شرایط بهتر عمل می‌کند. به طور کلی، این مطالعه اهمیت در نظر گرفتن فعالیت‌های تعمیر و نگهداری ماشین‌آلات و وقفه در سیستم زمان‌بندی کار کارگاهی انعطاف‌پذیر برای تولید محصولات فاسد شدنی را نشان می‌دهد. مدل ریاضی پیشنهادی و روش تجزیه Benders در نمونه‌های کوچک و الگوریتم D-GWO در نمونه‌های بزرگ راه‌حل‌های مناسبی را ارائه می‌دهند.