



A Mathematical Model and a Firefly Algorithm for an Extended Flexible Job Shop Problem with Availability Constraints

Willian Tessaro Lunardi¹ , Luiz Henrique Cherri², and Holger Voos¹

¹ Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, 6 rue Coudenhove-Kalergi, 1359 Luxembourg City, Luxembourg
`{willian.tessarolunardi,holger.voos}@uni.lu`

² Institute of Mathematics and Computer Sciences (ICMC), University of São Paulo, 400 Avenida Trabalhador São-Carlense, São Paulo 13566-590, Brazil
`lhcherri@icmc.usp.br`

Abstract. Manufacturing scheduling strategies have historically ignored the availability of the machines. The more realistic the schedule, more accurate the calculations and predictions. Availability of machines will play a crucial role in the Industry 4.0 smart factories. In this paper, a mixed integer linear programming model (MILP) and a discrete firefly algorithm (DFA) are proposed for an extended multi-objective FJSP with availability constraints (FJSP-FCR). Several standard instances of FJSP have been used to evaluate the performance of the model and the algorithm. New FJSP-FCR instances are provided. Comparisons among the proposed methods and other state-of-the-art reported algorithms are also presented. Alongside the proposed MILP model, a Genetic Algorithm is implemented for the experiments with the DFA. Extensive investigations are conducted to test the performance of the proposed model and the DFA. The comparisons between DFA and other recently published algorithms shows that it is a feasible approach for the stated problem.

Keywords: Firefly algorithm · Flexible job-shop scheduling
Metaheuristics · Mixed integer linear programming
Availability constraints

1 Introduction

The flexible job shop problem (FJSP) is an extension of the job shop problem (JSP) where is assumed that there is often more than one machine that is able to process a particular manufacturing task. The FJSP can be decomposed into two sub-problems: the machine selection problem (MS) and the operations sequencing problem (OS). Most of the FJSP studies have purely focused on assumptions that machines are continuously available. Nevertheless, in a real-world situation,

continuous availability of machines is not normally feasible. Machine unavailable periods might be consequent of pre-scheduling, preventive maintenance, shift pattern, or the overlap of two consecutive time horizons in the rolling time horizon planning algorithm.

There are various types of availability constraints in production systems. They can be categorized as *fixed* and *non-fixed*. The unavailable period of a fixed availability constraint starts at a fixed time point. Unavailable periods can also be categorized as *crossable* when it allows an operation to be interrupted and resumed, and *non-crossable* when it prevents the interruption of any operation. *Resumable* means that an operation can continue the processing when it is released from an interruption resultant of an unavailable period and *non-resumable* means an operation must be reprocessed fully after interrupted by an unavailable period [9].

Most existing literature focuses on the problem of integrating production scheduling with unavailable periods in the context of a single machine, parallel machine and flow shop (especially two-machine problems). The FJSP with non-resumable operations was addressed in [3]. The periods of unavailability are non-crossable, non-fixed and flexible within an end-time window and have to be determined during the scheduling procedure. In [14], a Genetic Algorithm (GA) was proposed to solve the multi-purpose machine (MPM) scheduling problem with fixed non-crossable unavailable periods in a job shop environment with non-resumable operations. A filtered beam search (FBS) [9], was proposed to solve the FJSP with non-fixed and fixed non-crossable unavailable periods and non-resumable operations.

In this paper, we put forward a mixed integer linear model (MILP) and a discrete firefly algorithm (DFA) for solving the FJSP with fixed crossable unavailable periods and resumable operations. In order to evaluate the performance of our methods, as well to be close to situations that may happen in industrial reality, we propose a new set of instances with fixed availability data. In addition, as the FJSP-FCR is an extension of the traditional FJSP. We also used traditional FJSP instances for the computational experiments. These instances include 35 open problems for FJSP. Through experimental studies, the merits of this work are clearly demonstrated.

The remainder of this paper is structured as follows. The problem formulation and the model are presented in Sect. 2. The discrete firefly algorithm and solution representation are discussed in Sect. 3. Numerical results are reported in Sect. 4. Finally, conclusions are presented at the end of this work.

2 MILP Model

The formulation of the FJSP-FCR can be given as follows. There is a set of n jobs and a set of m machines. Each job i consists of a sequence of J_i operations. M denotes the set of all machines. Each operation $O_{ij}(i = 1, \dots, n; j = 1, \dots, J_i)$ has to be processed on a machine k out of a set of given compatible machines M_{ij} ($k \in M_{ij}, M_{ij} \subseteq M$). In this work, we extend the classical FJSP formulation and

we consider that operations are resumable and machines are not continuously available. Each machine k has M_k crossable unavailable periods. We denote U_{kr} as the r th crossable unavailable period on machine k , with su_{kr} and cu_{kr} being respectively the unavailable period starting and completion time.

The notations used in this paper are summarized below.

Indices

- k : index of machines, $k = 1, \dots, m$;
- r : index of unavailabilities, $r = 1, \dots, M_k$;
- i, h : index of jobs, $i, h = 1, \dots, n$;
- j, g : index of operation sequences, $j, g = 1, \dots, J_i$;

Parameters

- J_i : total number of operation of job i ;
- M_k : total number of unavailable periods at machine k ;
- O_{ij} : the j th operation of job i ;
- M_{ij} : machines able to perform operation O_{ij} ;
- p_{ijk} : processing time of O_{ij} on machine k ;
- U_{kr} : the r th unavailable period of machine k ;
- su_{kr} : starting time of unavailable period U_{kr} ;
- cu_{kr} : completion time of unavailable period U_{kr} ;
- λ : an weight coefficient;
- L : an arbitrary large positive number;

Decision variables

- C_{max} : maximal completion time of the machines;
- W_{max} : maximal workload of the machines ($\max_k\{W_k\}$);
- s_{ijk} : starting time of operation O_{ij} on machine k ;
- c_{ijk} : completion time of the operation O_{ij} ;
- $v_{ijk} :$
$$\begin{cases} 1 & \text{if } O_{ij} \text{ is performed on machine } k \\ 0 & \text{otherwise;} \end{cases}$$
- $u_{ijkr} :$
$$\begin{cases} 1 & \text{if } s_{ijk} < su_{kr} < c_{ijk} \quad \forall i, j, r \quad \forall k \in M_{ij} \\ 0 & \text{otherwise;} \end{cases}$$
- $y_{ijkr} :$
$$\begin{cases} 1 & \text{if } U_{kr} \text{ precedes operation } O_{ij} \text{ on machine } k \\ 0 & \text{otherwise;} \end{cases}$$
- $z_{ijhgk} :$
$$\begin{cases} 1 & \text{if } O_{ij} \text{ precedes operation } O_{hg} \text{ on machine } k \\ 0 & \text{otherwise.} \end{cases}$$

The mixed integer programming model for the FJSP-FCR can be given as follows:

$$\text{minimize } \lambda_1 C_{max} + \lambda_2 W_{max} + \lambda_3 \sum_{k=1}^m \sum_{i=1}^n \sum_{j=1}^{J_i} p_{ijk} v_{ijk} \quad (1)$$

$$\text{s.t. } C_{max} \geq \sum_{k \in M_{ij}} c_{ijk}, \quad \forall i, j = J_i \quad (2)$$

$$W_{max} \geq \sum_{i=1}^n \sum_{j=1}^{J_i} p_{ijk} v_{ijk}, \quad \forall k \quad (3)$$

$$\begin{aligned} c_{ijk} &\geq s_{ijk} + p_{ijk} \\ &+ \sum_{\forall r} (cu_{rk} - su_{rk}) u_{ijk} \\ &- (1 - v_{ijk}) L, \end{aligned} \quad \forall i, j, k \in M_{ij} \quad (4)$$

$$s_{ijk} \geq c_{hik} - z_{ijhgk} L, \quad \forall i < h, j, g, k \in M_{ij} \cap M_{hg} \quad (5)$$

$$s_{hik} \geq c_{ijk} - (1 - z_{ijhgk}) L, \quad \forall i < h, j, g, k \in M_{ij} \cap M_{hg} \quad (6)$$

$$c_{ijk} \leq su_{kr} + u_{ijk} L + y_{ijk} L \quad \forall i, j, r, k \in M_{ij} \quad (7)$$

$$s_{ijk} \geq cu_{kr} - (1 - y_{ijk}) L, \quad \forall i, j, r, k \in M_{ij} \quad (8)$$

$$\sum_{k \in M_{ij}} s_{ijk} \geq \sum_{k \in M_{ij}} c_{ij-1k}, \quad \forall i, j = 2, \dots, J_i \quad (9)$$

$$\sum_{k \in M_{ij}} v_{ijk} = 1, \quad \forall i, j \quad (10)$$

$$s_{ijk} \leq v_{ijk} L, \quad \forall i, j, k \in M_{ij} \quad (11)$$

$$c_{ijk} \leq v_{ijk} L, \quad \forall i, j, k \in M_{ij} \quad (12)$$

Objective function (1) ensures the minimization of maximal completion time, maximal workload, and total workload of the machines and is supported by constraints (2) and (3). Constraints (11) and (12) ensures that the start and the completion time of operation on a specific machine is zero if it is not performed on this machine. The duration of the operation, considering its processing time and all the unavailabilities it passes through, is ensured by Constraints (4). Constraints (5) and (6) guarantee that two operations do not overlap on the same machine. Constraints (7) and (8) certify that the operations do not overlap the unavailabilities and, if it occurs, it is accounted to increase the operation time (performed by Constraints (4)). The precedence of each job operations is established by Constraints (9). Constraints (10) states that one machine can be selected from the set of available machines for each operation. The parameter L is an upper bound to the maximum processing time and unavailable time and is calculated as $\sum_i^n \sum_j^{J_i} \max_{k \in M_{ij}} p_{ijk} + \max_{k=1, \dots, m} \left(\sum_{r=1}^{M_k} cu_{rk} - su_{rk} \right)$.

3 Firefly Algorithm

The firefly algorithm is a nature-inspired meta-heuristic for solving continuous problems and has been motivated by the simulation of the social behavior of

Algorithm 1. Firefly Algorithm

```

1: Objective function  $f(x)$ ,  $x = (x_1, \dots, x_d)^T$ 
2: Generate initial pop.  $P$  of fireflies  $x_i (i = 1, 2, \dots, c)$ 
3: Light intensity  $I_i = f(x_i)$ 
4: Define light absorption coefficient  $\gamma$ 
5: while ( $t < MaxGeneration$ ) do
6:   for each  $x_i \in P$  do
7:     for each  $x_j \in P$  do
8:       if ( $I_i < I_j$ ) then Move  $x_i$  towards  $x_j$  end if
9:       Vary  $\beta$  with distance  $r$  via  $\exp[-\gamma r]$ 
10:      Evaluate solutions and update light intensity
11:    end for  $j$ 
12:  end for  $i$ 
13:  Rank fireflies and find the current global best
14: end while

```

fireflies. The two fundamental functions of its flashing lights are to attract mating partners (communication), and to attract potential prey.

In essence, FA uses the three following idealized rules: all fireflies are unisex; attractiveness β is proportional to their brightness, in this way for any two flashing fireflies, the less bright one will move towards the brighter one; the brightness of a firefly is affected or determined by the landscape of the objective function. The pseudo code shown in Algorithm 1 summarizes the basic steps of the FA.

3.1 Variations of Light Intensity and Attractiveness

The variation of light intensity and formulation of the attractiveness are two important issues. For simplicity, we can always assume the attractiveness of a firefly is determined by its brightness, which in turn is associated with the encoded objective function.

The attractiveness function $\beta(r)$ can be any monotonically decreasing functions such as the following generalized form

$$\beta(r) = \beta_0 e^{-\gamma r^m}, m \geq 1, \quad (13)$$

where β_0 is the attractiveness at $r = 0$, and r is the distance between two fireflies. The Eq. (13) can be approximated as

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}. \quad (14)$$

The distance between any two fireflies i and j , at position x_i and x_j , can be defined as a Cartesian distance:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^d (x_{ik} - x_{jk})^2}, \quad (15)$$

where x_{ik} is the k th component of the spatial coordinate x_i of i th firefly.

OS	2	1	3	2	3	1	1	2
MS	2	4	3	1	3	4	2	1
	O_{21}	O_{11}	O_{31}	O_{22}	O_{32}	O_{12}	O_{13}	O_{23}
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}

Fig. 1. Example of OS string and MS string of a firefly.

The random movement of a firefly i towards another more brighter firefly j is determined by

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} (x_i - x_j) + \alpha \epsilon_i, \quad (16)$$

where the second term considers a firefly's attractiveness, the third term is randomization with α being the randomization parameter, and ϵ_i is a vector of random numbers drawn from a Gaussian distribution or uniform distribution. For most applications we can take $\beta_0 = 1$, $\alpha \in [0, 1]$. The parameter γ is crucially important in determining the speed of the convergence and how the FA algorithm behaves. For most applications, it typically varies from 0.001 to 1000. In this implementation of the algorithm, we used $\beta_0 = 1.0$, $\alpha \in [0, 1]$ and $\gamma = 0.1$.

3.2 Firefly Representation for the FJSP

In our proposed algorithm, each firefly represents an FJSP solution, i.e. operation sequence and machine assignment. The algorithm starts with an initial population of fireflies. Each firefly is attracted by other fireflies to varying degrees, on the basis of the objective value of those solutions and the distance between them, i.e. how different they are. The population of fireflies evolves by each firefly randomly (not directly) moving toward the most attractive solution.

The FJSP contains two sub-problems, in this way, our representation contains two strings. The MS string denotes the selected machine for the corresponding operations of each job. The h th part of the MS string can assume any value $k \in M_v$ and represents the assigned machine for operation v .

The OS string represents the order in which the operations will be processed in their respective machines. This representation uses an unpartitioned permutation with J_i repetitions of the job numbers, i.e. each job number appears J_i times in the OS string. By scanning the OS string from left to right, the f th appearance of a job number refers to the f th operation of this job. In this way, any permutation of the OS string can be decoded into a feasible solution and avoid the use of a repair mechanism. When a firefly is decoded, the OS string is translated into a sequence of operation at first. Figure 1 presents an example of OS and the MS strings.

The computation of the makespan can be obtained using graph traversal algorithms, commonly used in temporal planning. During the computation of the makespan for the FJSP-FCR, due to the advent of the unavailable periods, before

Table 1. Update of the movement of firefly i towards a brighter firefly j .

	MS string	OS string
Firefly j	1 4 1 3 2 2 3 4	1 2 3 2 1 1 2 3
Firefly i	2 4 3 1 3 4 2 1	2 1 3 2 3 1 1 2
H_{ij} and S_{ij}	$\{(1, 1), (3, 1), (4, 3), (5, 2), (6, 2), (7, 3), (8, 4)\}$	$\{(1, 2), (5, 6), (6, 7), (7, 8)\}$
$ H_{ij} $ and $ S_{ij} $	7	4
Attractiveness $\beta(r)$	0.17	0.38
$rand \in [0, 1]$	{0.35, 0.1, 0.09, 0.14, 0.33, 0.49, 0.32}	{0.52, 0.05, 0.12, 0.69}
Movement β -step	$\{(3, 1), (4, 3), (5, 2)\}$	$\{(5, 6), (6, 7)\}$
Position after β -step	2 4 1 3 2 4 2 1	2 1 3 2 1 1 3 2
Position after α -step	2 4 1 3 2 2 4 1	2 1 3 2 1 1 2 3

updating the outcome edges and vertices, is necessary to check whether there is an overlap of the operation with an unavailable period in the machine route. Thus, if the starting of the operation is overlapping the unavailable interval, the starting of the operation must be delayed to the end of the unavailable period; if the starting of the unavailable period is overlapping the operation, the processing time of the operation must be increased by the extension of the unavailable period.

3.3 Discrete Firefly Algorithm for the FJSP

The FA has been originally developed for solving continuous optimization problems and cannot be directly applied to solve discrete optimization problems. The main challenges for using the FA to solve FJSP are computing the discrete distance between two fireflies, and how they move in the coordination. In this work, the discretization is done for the following issues.

3.4 Distance

The discrete distance between two fireflies is defined by the distance between the permutation of its strings. There are two possible ways to measure the distance between two permutations: (a) Swapping distance (S_{ij}), i.e. the number of minimal required swaps in a permutation i in order to obtain j ; and (b) Hamming distance (H_{ij}), i.e. the number of non-corresponding elements in the sequence of i compared with sequence j .

The distance between two MS strings can be measured by using Hamming distance. The minimal number of swaps cannot be used for the MS string since two different strings can contain different elements. Given two MS strings, $MS_i = \{2 4 3 1 3 4 2 1\}$ and $MS_j = \{1 4 1 3 2 2 3 4\}$, every bit is compared and

Table 2. The experimental results on Fattahi instances.

Instance	n	o	m	OOY		WLH		DFA	
				C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
MFJS01	5	3	6	468	0.20	468	0.21	468	0.11
MFJS02	5	3	7	446	0.32	446	0.32	446	0.18
MFJS03	6	3	7	466	0.90	466	0.91	466	0.36
MFJS04	7	3	7	554	2.54	554	2.56	554	1.99
MFJS05	7	3	7	514	1.64	514	1.78	514	1.28
MFJS06	8	3	7	634	3.80	634	3.88	634	4.46
MFJS07	8	4	7	879	43.33	879	44.54	879	9.34
MFJS08	9	4	8	884	977	884	1050.55	884	15.23
MFJS09	11	4	8	[877.9; 1111] 20.98%	3600	[861; 116] 22.85%	3600	1055	31.22
MFJS10	12	4	8	[1012; 1208] 16.23%	3600	[1008.2; 1220] 17.36%	3600	1196	39.32

the number of bits whose are not equal are recorded, the Hamming distance is $H_{ij} = 7$. The distance between two OS strings of two fireflies can be measured with the so-called swapping distance. Given two OS strings, $OS_i = \{2\ 1\ 3\ 2\ 3\ 1\ 1\ 2\}$ and $OS_j = \{1\ 2\ 3\ 2\ 1\ 1\ 2\ 3\}$, the swapping distance is $S_{ij} = 4$.

3.5 Attraction and Movement

In this study we break up the movement given in Eq. (16) into two sub-steps: β -step and α -step. The attraction steps β and α are not interchangeable, thereby, β -step must be computed before α -step while finding the new position. Both steps are illustrated in details on Table 1, where the firefly i updates its position towards the a best firefly j . The parameters used in this illustration are as follows: $\beta_0 = 1$, $\gamma = 0.1$, $\alpha = 1$.

Moving Towards Another Firefly: β -Step. The β -step brings the iterated firefly closer to another firefly. An insertion mechanism and a pair-wise exchange mechanism are used to advance the MS string and OS string of a firefly towards the brighter firefly position. At first, all necessary insertions in the MS string and all pair-wise exchanges in the OS string, to make the elements of the current firefly equal to the best firefly, are computed and store in H_{ij} and S_{ij} . The Hamming distance and swap distance are respectively defined by $|H_{is}|$ and $|S_{ij}|$. The β probability is computed using Eq. (14). Secondly, it is defined which elements of H_{ij} and S_{ij} will be used to change the current solution. A random number $rand \in [0, 1]$ is generated for each element, and if $rand \leq \beta$, then the corresponding insertion/pair-wise exchange is performed on the elements of the current firefly.

Table 3. The experimental results (computational time in terms of seconds) on the proposed instances with fixed available periods.

Instance	n	o	m	u	WLH		GA		DFA	
					C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
FCR01	5	3	6	6	513	0.20	513	3.57	513	0.13
FCR02	5	3	7	9	548	0.56	552	7.18	548	0.16
FCR03	6	3	7	14	620	2.50	620	5.80	620	0.44
FCR04	7	3	7	17	746	27.46	748	5.86	746	2.33
FCR05	7	3	7	20	693	20.94	709	11.23	693	4.28
FCR06	8	3	7	20	774	4.83	777	11.31	774	6.17
FCR07	8	4	7	12	[1000; 1024] 2.34%	3600	1044	28.46	1024	10.34
FCR08	9	4	8	35	[1414; 1467] 3.61%	3600	1478	35.22	1418	16.78
FCR09	11	4	8	49	[1410.96; 2051] 31.21%	3600	1976	65.65	1944	34.70
FCR10	12	4	8	52	[1815.26; 2631] 31.00%	3600	2337	70.64	2320	43.01

Random Movement: α -Step. The α -step is much simpler than the β -step. The random movement of firefly $\alpha(rand - 1/2)$ is approximated as $\alpha(rand_{int})$ given Eq. 17.

$$x_i = x_i + \alpha(rand_{int}). \quad (17)$$

It allows us to shift the permutation into one of the neighbouring permutations, by choosing an element position using $\alpha(rand_{int})$ and swap with another position in the string which also chosen at random, where $rand_{int}$ is a positive integer generated between the minimum and maximum number of elements in the string.

4 Numerical Results

To solve the MILP models, we used the IBM ILOG CPLEX 12.7 solver with default parameters and a time limit of 3600 seconds. The DFA proposed in this work, and the Genetic Algorithm (GA) proposed in a previous work [7], were coded in C++. The MILP models, the DFA and the GA were run on an Intel Core i7 2.70 GHz, with 8 GB of RAM memory. The best and average results from 50 different runs were collected for performance comparison. Observations among the MILP model, the proposed DFA and GA, and other state-of-the-art reported algorithms are also provided to determine their performance. To demonstrate the efficiency of the proposed methods, the computational time is further compared.

The instances used in the experiments can be characterized by number of jobs n , number of machines m , number of operations o , and number of unavailable periods u . The DFA parameters consist of the population size P , a maximum number of generations G , firefly's attractiveness β_0 , light absorption γ , and randomization α . We kept fixed the following parameters: $\beta_0 = 1.0$, $\alpha \in [0, 1]$, and

Table 4. Comparison of the DFA with other algorithms on Brandimarte instances.

Instance	<i>n</i>	<i>m</i>	<i>o</i>	TABC		MA		DFA	
				<i>C_{max}</i>	CPU	<i>C_{max}</i>	CPU	<i>C_{max}</i>	CPU
Mk01	10	6	7	40	3	40	20	40	5
Mk02	10	6	7	26	3	26	28	26	16
Mk03	15	8	10	204	1	204	53	204	3
Mk04	15	8	10	60	66	60	30	60	11
Mk05	15	4	10	173	78	172	36	172	19
Mk06	10	15	15	60	173	59	80	59	63
Mk07	20	5	5	139	66	139	37	139	43
Mk08	20	10	15	523	2	523	77	523	4
Mk09	20	10	15	307	304	307	75	307	34
Mk10	20	15	15	202	418	202	90	202	94

$\gamma = 0.1$. The variation of P and G was based on the size of each instance. We used $P = 125$ and $G = 100$ for small instances, i.e. less or equal to 6 jobs and 5 machines; $P = 250$ and $G = 200$ for medium instances i.e. less or equal to 10 jobs and 8 machines; $P = 500$ and $G = 300$ for instances that does not belong to another group.

4.1 Fattahi Instances

We compare the proposed model and the DFA experimentally with [8] (OOY), a concise MILP model for the FJSP and has proven to be effective when compared to other state-of-the-art MILP models, as shown in [2]. Both models were implemented in the same platform and experiments were conducted in the same computer, mentioned in Sect. 4. The weight coefficients employed in this experiment are: $\lambda_1 = 1.0$, $\lambda_2 = 0.0$, and $\lambda_3 = 0.0$. Table 2 shows the numerical results of the experiments involving the Fattahi instances.

The proposed model contains additional constraints (compared to OOV model) to address the FJSP-FCR. Even with the additional constraints to address the available periods, our model can achieve similar results solving the standard FJSP. CPLEX found the optimal solution for the instances MFJS01-08. The DFA obtained the best solutions for all instances.

4.2 FJSP-FCR Instances

Due to the lack of literature, in order to compare the DFA with other algorithms using the FJSP-FCR instances, we implemented a GA for this experiment. The results of this experiment are presented in the Table 3. This set of instances can be obtained in JSON format through the following URL: https://github.com/snt-robotics/fjsp_fcr.

Table 5. The experimental results on Kacem instances of a multi-objective optimization experiment.

Algorithm	4 × 5				8 × 8				10 × 7				10 × 10				15 × 10			
	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU	f_1	f_2	f_3	CPU
AL + CGA	16	10	34	—	15	13	79	—	—	—	—	—	7	5	45	—	23	11	93	—
PSO + SA	—	—	—	—	15	12	75	—	—	—	—	—	7	6	44	—	12	11	91	—
AIA	—	—	—	—	14	12	77	0.76	—	—	—	—	7	5	43	8.97	11	11	93	109.22
P-DABC	11	10	32	—	14	12	77	—	12	11	61	—	8	7	41	—	12	11	91	—
SMF	12	8	32	2.6	14	12	77	39.5	11	10	62	109.5	7	6	42	39.1	11	10	93	864.6
PSO + TS	12	8	32	0.34	14	12	77	1.67	—	—	—	—	7	6	43	2.05	11	11	93	10.88
WLH	12	8	32	0.06	14	12	77	0.54	11	10	62	0.36	7	5	43	1.07	11	11	93	211.74
GA	11	10	32	0.29	14	12	77	0.90	11	10	62	1.57	7	6	42	2.02	12	12	93	18.76
DFA	12	8	32	0.11	14	12	77	0.64	11	10	62	0.84	7	5	43	1.27	11	11	93	6.86

$n \times m$ total number of jobs and machines. — equals not available. f_1 , f_2 and f_3 are respectively the C_{max} , W_{max} , and total workload of the machines.

In this experiment, we can see that the DFA achieve better results, and is more efficient and effective than the GA. The MILP model has found best solutions for the FCR01, …, FCR06, and for the other instances, bounds were provided.

4.3 Brandimarte Instances

To better demonstrate the effectiveness of the DFA we compare results with other state-of-the-art algorithms for the FJSP using the Brandimarte instances. We compare the DFA with an artificial bee colony algorithm (TABC) [4] and a memetic algorithm (MA) [12]. The TABC was implemented on an Intel 2.4 GHz Core 2 Duo processor with 4.0 GB of RAM memory in C++. The MA was implemented on an Intel Core i7-3520M 2.9 GHz processor with 8.0 GB of RAM memory in Java. The weight coefficients employed in this experiment are: $\lambda_1 = 1.0$, $\lambda_2 = 0.0$, and $\lambda_3 = 0.0$. Table 4 shows the comparison on the 10 Brandimarte instances. In this experiment, we can see that the DFA can achieve similar results to state-of-the-art algorithms.

4.4 Kacem Instances

Kacem et al. proposed five multi-objective FJSP instances. Using these instances our MILP model (WLH), and the DFA are compared with the hybrid particle swarm optimization and tabu search (PSO + TS) [13], implemented on a Pentium IV 1.8 GHz in C++; the discrete artificial bee colony (DABC) [6], implemented on a Pentium IV 1.8 GHz with MB of RAM memory in C++; the artificial immune algorithm (AIA) [1] implemented on a 2.0 GHz processor with 256 MB of RAM memory in C++; the simulation modeling (SMF) [11], implemented on a Pentium IV 2.4 GHz personal with 512 MB RAM memory in Matlab; the hybrid evolutionary and fuzzy logic (AL + CGA) [5]; the GA

proposed in [7], and the hybrid particle swarm optimization and simulating annealing (PSO + SA) [10]. The weight coefficients used in this experiment are: $\lambda_1 = 0.5$, $\lambda_2 = 0.3$, and $\lambda_3 = 0.2$. Table 5 shows the comparison of the results on the five Kacem instances.

5 Conclusion

Planning and scheduling with machine availability constraint become increasingly more important as a better understanding of their importance in various applications. We put forward a new MILP model and an FA for the FJSP-FCR. New instances are provided. We further presented computational experiments on classical instances in order to provide comparisons with other state-of-the-art algorithms. The numerical results make clear that the MILP model is important for comparisons with non-exact methods, providing good bounds to many small and medium size instances. The experiments among the DFA and others recently published algorithms shows that it is a feasible approach for the considered problem.

References

1. Bagheri, A., Zandieh, M., Mahdavi, I., Yazdani, M.: An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Gener. Comput. Syst.* **26**(4), 533–541 (2010)
2. Demir, Y., İşleyen, S.K.: Evaluation of mathematical models for flexible job-shop scheduling problems. *Appl. Math. Model.* **37**(3), 977–988 (2013)
3. Gao, J., Gen, M., Sun, L.: Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *J. Intell. Manuf.* **17**(4), 493–507 (2006)
4. Gao, K.Z., Suganthan, P.N., Chua, T.J., Chong, C.S., Cai, T.X., Pan, Q.K.: A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Syst. Appl.* **42**(21), 7652–7663 (2015)
5. Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Math. Comput. Simul.* **60**(3), 245–276 (2002)
6. Li, J.Q., Pan, Q.K., Gao, K.Z.: Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. *Int. J. Adv. Manuf. Technol.* **55**(9), 1159–1169 (2011)
7. Lunardi, W.T., Voos, H.: Comparative study of genetic and discrete firefly algorithm for combinatorial optimization. In: 33rd ACM/SIGAPP Symposium on Applied Computing, Pau, France, 9–13 April 2018 (2018)
8. Özgüven, C., Özbakir, L., Yavuz, Y.: Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Appl. Math. Model.* **34**(6), 1539–1548 (2010)
9. Wang, S., Yu, J.: An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Comput. Ind. Eng.* **59**(3), 436–447 (2010)
10. Xia, W., Wu, Z.: An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Comput. Ind. Eng.* **48**(2), 409–425 (2005)

11. Xing, L.N., Chen, Y.W., Yang, K.W.: Multi-objective flexible job shop schedule: design and evaluation by simulation modeling. *Appl. Soft Comput.* **9**(1), 362–376 (2009)
12. Yuan, Y., Xu, H.: Multiobjective flexible job shop scheduling using memetic algorithms. *IEEE Trans. Autom. Sci. Eng.* **12**(1), 336–353 (2015)
13. Zhang, G., Shao, X., Li, P., Gao, L.: An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Comput. Ind. Eng.* **56**(4), 1309–1318 (2009)
14. Zribi, N., El Kamel, A., Borne, P.: Minimizing the makespan for the MPM job-shop with availability constraints. *Int. J. Prod. Econ.* **112**(1), 151–160 (2008)