A Discrete Firefly Algorithm for an Extended Flexible Job Shop Problem With Fixed Availability Constraints

Abstract

The introduction of computing concepts such as the Internet of Things (IoT), Fog and Edge Computing, and Digital twin in the industry shifts the traditional manufacturing systems into smart and autonomous decentralized flexible manufacturing systems. Recent works show that traditional scheduling techniques and methods still are going to play essential roles in smart and distributed scheduling systems, being combined and reused. Models and techniques have to take into account essential and realistic characteristics present in the modern manufacturing environments such as flexibility towards mass customization, job route flexibility, transportation, operation overlapping, setup-time, fixed and non-fixed preventive maintenance and others. The considered problem in this work is issued from modern printing and boarding industry. Each job must be processed in a particular order and resources are machines that can be used for processing different types of operations. Non-continuous machines availability is essential for preventive maintenance and shift patterns. The considered problem is an extension of flexible job shop scheduling problem (FJSP). In this paper, we extend and compare two mixed integer linear models (MILP) and propose a discrete firefly algorithm (DFA) for solving the FJSP with fixed availability constraints with *crossable* unavailable periods and *resumable* operations. Simulated and real-world instances are used for the experiments. A genetic algorithm alongside the bounds provided by the MILP models is used in the experiments with the DFA. Computational results show that our algorithm gives results comparable with the best algorithms known so far for the standard FJSP.

1 Introduction

Production scheduling is one of the most important issues in the planning and scheduling of modern manufacturing systems. Most production scheduling problems, including the standard flexible job shop scheduling problem (FJSP), assume that machines are continuously available. In a real-world situation, continuous availability of machines is not feasible [Saidy *et al.*, 2008]. Machine unavailable periods might be consequent of pre-scheduling, machine maintenance, shift pattern, or the overlap of two consecutive time horizons in the rolling time horizon planning algorithm. Predictive capabilities such as preventive maintenance are one of the pillars of the Industry 4.0 [Zhang *et al.*, 2017]. Production scheduling with preventive maintenance activities is also treated as availability constraint of machines in the literature. Therefore, a more realistic scheduling model should consider machine availability constraints. This helps to reduce the frequency of rescheduling and improving the robustness.

Availability constraints can be categorized as *fixed* and *non-fixed*. The unavailable period of a fixed availability constraint starts at a fixed time point [Rajkumar *et al.*, 2011]. Unavailable periods can also be caregorized 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 [Wang and Yu, 2010].

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). Gao *et al.* considered the FJSP with non-resumable operations. 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. Zribi *et al.* proposed a Genetic Algorithm (GA) to solve the multi-purpose machine (MPM) scheduling problem with fixed non-crossable unavailable periods in a job shop environment with non-resumable operations. Wang and Yu proposed a filtered beam search (FBS) for the FJSP with non-fixed and fixed non-crossable unavailable periods and non-resumable operations.

In this paper, we propose a new mixed integer linear model (MILP) and a discrete firefly algorithm (DFA) for solving the FJSP with fixed availability constraints with *crossable* unavailable periods and *resumable* operations, so-called FJSP-FCR. Due to the lack of literature, we propose a new set of open-source instances with fixed availability data for the experiments. The optimal solutions and bounds given by the

MILP model are used alongisde a GA for the experiments with the DFA.

The remainder of this paper is structured as follows. The problem formulation and the MILP model is presented in Section 2. The DFA and solution representation are discussed in Section 3. Numerical results and discussions are presented in Section 4. Finally, conclusions and future work are presented at the end of this work.

2 MILP Model

This section presents the proposed MILP model, which extends the MILP model proposed in [Birgin *et al.*, 2014].

2.1 Notation

Let (V, A) be a directed acyclic graph (DAG). The vertices represents the *operations* and the *unavailable periods*, where $\overline{V}$ is a subset of V that contains all vertices corresponding to operations, and $\underline{V}$ all vertices corresponding to unavailable periods. The arcs represent *precedence constraints*. We denote M as a set of *machines* and a function F that associates a non-empty subset $F(v)$ of M with each operation $v \in \overline{V}$. The machines in $F(v)$ are capable of processing the operation v. Additionally, for each operation v and each machine $k \in F(v)$, we detone a positive rational number $p_{v,k}$ representing the *processing time* of operation v on machine k.

For each machine k, let V_k be the set of operations that can be processed on machine k, that is, $V_k = \{v \in \overline{V} : k \in F(v)\}$. Let B_k be the set of all ordered pairs of distinct elements of V_k. The pairs (v, w) in B_k are designed to prevent v and w from using machine k at the same time. Let B denote the union of all B_k. Hence, $(v, w) \in B$ if and only if $v \neq w$ and $F(v) \cap F(w) \neq \emptyset$.

A *machine assignment* is a function f that assigns a machine $f(v) \in F(v)$ with each operation v. Given a machine assignment f, let B^f be the set of all ordered pairs of distinct operations to be processed on the same machine, that is, $B^f = \{(v, w) \in B : f(v) = f(w)\}$. A *selection* is any subset Y of B^f such that, for each $(v, w) \in B^f$, exactly one of (v, w) and (w, v) is in Y. A selection corresponds to an ordering of the operations to be processed on the same machine.

Let the function $g(u)$ returns the respective machine of unavailable period u. For each machine k, let U_k be the set of unavailable periods on machine k, that is, $U_k = \{u \in \underline{V} : k = g(u)\}$. Let C_k be the set of ordered pairs of each unavailable period $u \in U_k$ and each operation $v \in V_k$. The pairs (u, v) in C_k are designed to prevent v from using machine k during the unavailable period u. Let C denote the union of all C_k. Given a machine assignment f, let C^f be the set of pairs (u, v) on the same machine, that is $C^f = \{(u, v) \in C : g(u) = f(v)\}$. An *unavailability selection* is a subset Z of C^f with the relations between u and $v \in V_f$.

An operation can cross an unavailable period in case its processing cannot be finished before the starting of the unavailable period. In this case, its processing time is increase by the extension of the unavailability period. Hence, given a machine assignment f, let $p_v^f := p_{v,f(v)+\sum_{u \in h(f(v),v)} \overline{c}_u - \overline{s}_u}$, where $h(k, v)$ is a function that assigns a subset of C_k containing unavailable periods that overlap the operation v, and $\overline{s}_u$ and $\overline{c}_u$ being respectively the starting and completion time of unavailable period u.

Given a machine assignment f and a admissible selection Y and Z, a *schedule* for $(V, A \cup Y \cup Z, p^f)$ is a function s from V to the set of non-negative rational number such that $s_v + p_v^f \leq s_w$ for each $(v, w) \in A \cup Y$ and $\overline{c}_u \leq s_v$ for each $(u, v) \in A \cup Z$. A schedule exists since $(V, A \cup Y \cup Z)$ is a DAG. The number s_v is the *starting time* of operation v. The *makespan* of a schedule s is the number $mks(s) := max_{v \in V}(s_v + p_v^f)$.

2.2 A New Model for the FJSP-FCR

In order to formulate the FJSP-FCR as a MILP, we use a binary array x to represent the machine assignments, a binary array y to represent operation selections, a binary array z to represent unavailable period selections and a binary array η to represent the crossed unavailable periods. We also use two rational arrays s and $p^{'}$, and a rational number C_{max}, where s represents the starting times, $p^{'}$ represents the processing times corresponding to the machine assignment given by x and crossed unavailable periods given by z, and C_{max} represents the makespan. The upper bound L on the makespan can be a global bound like $\sum_{v \in \overline{V}} \max_{\forall k \in M_v} p_{vk} + \max_{\forall k \in M} \left(\sum_{u \in U_k} \overline{c}_u - \overline{s}_u \right)$.

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

$$\text{Minimize } C_{max}$$
$$\text{subject to}$$
$$s_v + p_v^{'} \leq C_{max} \qquad \forall v \in \overline{V}, \quad (1)$$
$$\sum_{k \in M_v} x_{vk} = 1 \qquad \forall v \in \overline{V}, \quad (2)$$
$$p_v^{'} = \sum_{k \in M_v} p_{vk} x_{vk} + \sum_{u \in U_k} \eta_{vu} (\overline{c}_u - \overline{s}_u) \qquad \forall v \in \overline{V}, \quad (3)$$
$$y_{vw} + y_{wv} \geq x_{vk} + x_{wk} - 1 \qquad \forall k \in M, (v, w) \in B_k, \quad (4)$$
$$s_v + p_v^{'} \leq s_w \qquad \forall (v, w) \in A, \quad (5)$$
$$s_v + p_v^{'} - (1 - y_{vw}) L \leq s_w \qquad \forall (v, w) \in B, \quad (6)$$
$$s_v \geq \overline{c}_u (1 - z_{uv}) - (1 - x_{vk}) L \qquad \forall k \in M, (u, v) \in C_k, \quad (7)$$
$$s_v + p_v^{'} \leq \overline{s}_u + \eta_{vu} L + (1 - z_{uv}) L \qquad \forall k \in M, (u, v) \in C_k, \quad (8)$$
$$s_v \geq 0 \qquad \forall v \in \overline{V}. \quad (9)$$

Constraint (2) ensures x is a machine assignment. The constraint (3) makes array $p^{'}$ represent the processing time of operation extending it based on the overlapped unavailable periods. Constraint (6) makes sure y_{vw} and y_{wv} are not both equal to 1. Constraint (4) implies that y represents a selection. Constraint (7) makes sure s starts after u in case it precedes v, and in the opposite case, constraint (8) makes v finish before u based on z. If v is crossing u based on η, this constraint is trivially satisfied. Constraints (5), (6), and (9) make s represent a schedule. Finally, the objective function and con-

Job	$Operation$	M_1	M_2	M_3	M_4
J_1	O_{11}	1	3	4	1
	O_{12}	3	8	2	1
	O_{13}	3	5	4	7
J_2	O_{21}	4	1	1	4
	O_{22}	2	3	9	3
	O_{23}	9	1	2	2
J_3	O_{31}	8	6	3	5
	O_{32}	4	5	8	1

Table 1: An instance of the FJSP with three jobs and four machines.

straint (1) make sure C_{max} is the makespan of the schedule and is as small as possible.

3 Firefly Algorithm

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. These concepts are presented in details below.

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 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 following three idealized rules [Yang, 2010]:

- All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex;
- 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 attractiveness is proportional to the brightness and both decrease as their distance increases. If there is no brighter one than a particular firefly, it will move randomly;
- 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 which consists of the three rules discussed above.

3.1 Distance, Attraction and Movement

In the firefly algorithm, there are two important issues: the variation of light intensity and formulation of the attractiveness. 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

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \ \ m \geqslant 1, \tag{10}$$

Algorithm 1 Firefly Algorithm

```
 1: Objective function $f(x),\ x = (x_1, ..., x_d)^T$
 2: Generate initial pop. $P$ of fireflies $x_i (i = 1, 2, ..., c)$
 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
```

where β_0 is the attractiveness at $r = 0$, and r is the distance between two fireflies. As it is often faster to calculate $1/(1 + r^2)$ than an exponential function [Yang, 2010], the Equation (10) can be approximated as

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}. \tag{11}$$

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

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum\nolimits_{k=1}^{d} (x_{ik} - x_{jk})^2}, \tag{12}$$

where x_{ik} is the kth component of the spatial cordinate x_i of ith firefly.

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

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} (x_i - x_j) + \alpha \ \epsilon_i, \tag{13}$$

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. In a simplest form, ϵ_i can be replaced by **rand** $- 1/2$, where **rand** is a random number generator uniformly distributed in [0,1]. For most applications we can take $\beta_0 = 1$, $\alpha \in [0, 1]$.

3.2 Firefly Representation for the FJSP

The FJSP contains two sub-problems, in this way, our representation contains two strings. The MS string represents the machine assignment and the OS string represents the operation sequence.

The MS string denotes the selected machine for the corresponding operations of each job. The length of this string is equal to $\sum_{i=1}^{n} J_i$, where J_i is the number of operations of job i and n is the number of jobs. The hth part of the MS string can assume any value $k \in M_v$ and represents the assigned machine for operation v. The operation number does not change throughout the whole searching process.

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

	MS string	OS string
Firefly (P_{best})	1 4 1 3 2 2 3 4	1 2 3 2 1 1 2 3
Firefly (P)	2 4 3 1 3 4 2 1	2 1 3 2 3 1 1 2
d_{ms} and d_{os}	{(1,1), (3,1), (4,3), (5,2), (6,2), (7,3), (8,4)}	{(1,2), (5,6), (6,7), (7,8)}
r_{ms} and r_{os}	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

Table 2: Update of the movement of firefly P towards the global best P_{best}.

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 fth appearance of a job number refers to the fth 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 converted into a sequence of operation at first. Figure 1 presents an example of OS and the MS strings for the FJSP instance shown at at Table 1. The operations presented on Table 1 are denoted as O_{ij}, where j is the jth operation of job i. In this way, given the OS string presented at Figure 1, its the resultant translation into a list of ordered operations is: O_{21}, O_{11}, O_{31}, O_{22}, O_{32}, O_{12}, O_{13}, O_{23}. After the translation of the OS string each operation is assigned to a machine according to the MS string as follows: $[(O_{21}, M_1)$, (O_{11}, M_2), (O_{31}, M_2), (O_{22}, M_3), (O_{32}, M_1), (O_{12}, M_4), (O_{13}, M_3), $(O_{23}, M_4)]$.

The computation of the makespan can be obtained using a DAG and graph traversal algorithms, commonly used in temporal planning. It is necessary to highlight that due the unavailable periods, before updating the outcome edges and vertices, it must be checked whether there is an overlap of the operation with any unavailability in the machine route. Thus, the processing of the operation on this machine should be increased by the extension of the overlapped unavailable periods.

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.

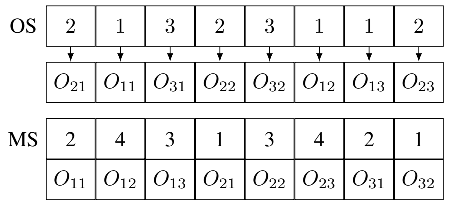
OS
2 1 3 2 3 1 1 2
O_{21} O_{11} O_{31} O_{22} O_{32} O_{12} O_{13} O_{23}
MS
2 4 3 1 3 4 2 1
O_{11} O_{12} O_{13} O_{21} O_{22} O_{23} O_{31} O_{32}

Figure 1: Example of an OS string and MS string of a firefly.

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 (D_s), i.e. the number of minimal required swaps of one permutation in order to obtain the other one [Karthikeyan *et al.*, 2015]; and (*b*) Hamming distance (D_h), i.e. the number of non-corresponding elements in the sequence [Kuo *et al.*, 2009].

The distance between two MS strings of two fireflies 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 $P^{ms} = \{2\ 4\ 3\ 1\ 3\ 4\ 2\ 1\}$ and $P^{ms}_{best} = \{1\ 4\ 1\ 3\ 2\ 2\ 3\ 4\}$, every bit is compared and the number of bits whose are not equal are recorded, the Hamming distance is $D_h(P^{ms}, P^{ms}_{best}) = 7$. The distance between two OS strings of two fireflies can be measured with the so-called swapping distance. Given two OS strings $P^{os} = \{2\ 1\ 3\ 2\ 3\ 1\ 1\ 2\}$ and $P^{os}_{best} = \{1\ 2\ 3\ 2\ 1\ 1\ 2\ 3\}$, the swapping distance is $D_s(P^{os}, P^{os}_{best}) = 4$.

3.5 Attraction and Movement

In this study we break up the movement given in equation (13) into two sub-steps: β-step and α-step as given in equation (14) and equation (15) respectively

$$x_i = \beta(r)(x_j - x_i), \tag{14}$$
$$x_i = x_i + \alpha(rand - 1/2). \tag{15}$$

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 2, where a firefly P update its position towards the global best firefly P_{best}. 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 global best 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 found and store in d_{ms} and d_{os} respectively. The hamming distance and swap distance are respectively defined by $r_{ms} = |d_{ms}|$ and $r_{os} = |d_{os}|$. The

Instance	n	o	m	OOY		WLH		DFA	
				C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
MFJS01	5	3	6	468	0.20	468	0.10	468	0.11
MFJS02	5	3	7	446	0.32	446	0.15	446	0.18
MFJS03	6	3	7	466	0.90	466	0.25	466	0.36
MFJS04	7	3	7	554	2.54	554	0.69	554	1.99
MFJS05	7	3	7	514	1.64	514	0.44	514	1.28
MFJS06	8	3	7	634	3.80	634	2.06	634	4.46
MFJS07	8	4	7	879	43.33	879	7.98	879	9.34
MFJS08	9	4	8	884	977	884	170.55	884	15.23
MFJS09	11	4	8	[877.91;1111] 20.98%	3600	[954;1070] 10.84%	3600	1055	31.22
MFJS10	12	4	8	[1008.20;1220] 17.36%	3600	[1012;1208] 16.23%	3600	1196	39.32

Table 3: The experimental results on Fattahi *et al.* instances. **Note this table will be increased to two columns in the final version of the paper.**

β probability is computed using equation (11). Secondly, we have to define which elements of d_{ms} and d_{os} will be used to change the current solution. A random number $rand \in [0, 1]$ is generated for each element of d_{ms} and d_{os}. In this way, if $rand \leqslant \beta$, then the corresponding insertion/pair-wise exchange is performed on the elements of the current firefly.

Random movement: α-step

The α-step is much simpler than the β-step. In equation 15 the random movement of firefly $\alpha(rand - 1/2)$ is approximated as $\alpha(rand_{int})$ given Equation 16.

$$x_i = x_i + \alpha(rand_{int}). \qquad (16)$$

It allows us to shift the permutation into one of the neighbouring permutation, by chosing 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 Results and Discussion

This section presents the results of computational experiments involving the MILP model and the DFA. We used as benchmarks the 10 instances proposed in [Fattahi *et al.*, 2007], 10 instances propose in [Brandimarte, 1993], and a new set of 18 instances where each instance contains the availability data of each machine. The experiments involving the MILP model are presented in the Section 4.1. The experiments involving the DFA are presented in the Section 4.2.

4.1 MILP Model Experiments

We compare the proposed model experimentally with [Birgin *et al.*, 2014] (ERN), which extends the usual definition of the FJSP allowing non-linear job routing, and with [Özgüven *et al.*, 2010] (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 [Demir and İşleyen, 2013]. 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 three models were implemented in the same platform and experiments involving the models, the DFA and the GA were conducted in the same computer, an Intel Core i7 2.70GHz, with 8 GB of RAM memory. Before the experiments with the MILP model, we comment on differences between the three models.

Inst.	TABC		MA		DFA	
	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
Mk01	40	3	40	20	40	5
Mk02	26	3	26	28	26	16
Mk03	204	1	204	53	204	3
Mk04	60	66	60	30	60	11
Mk05	173	78	172	36	172	19
Mk06	60	173	59	80	59	63
Mk07	139	66	139	37	139	43
Mk08	523	2	523	77	523	4
Mk09	307	304	307	75	307	34
Mk10	202	418	202	90	202	94

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

We used the ERN model for the extension due to the fact that it is more compact than the OOY model and allow non-linear route of jobs. The array variables that describe the selection and starting time have one more dimension in OOY since they are also indexed by the machines. Moreover, OOY has variables for the completion times that depend not only on the operations but also on the machines. Therefore, the OOY model uses significantly more variables. In addition, the ERN model consider non-linear route of jobs. In a non-linear route, an operation can have more than one predecessor and/or more than one successor. This is an essential constraint for the modern manufacturing, such in the modern printing industry as shown in [Vilcot and Billaut, 2011]. The difference of the proposed model with the ERN model is that our model has additional decision variables z, η, and constraints (Equations 3, 8, and 9) to deal with the availability constraint.

The first experiment compares the proposed model with the OOY and ERN for FJSP instances, which is a type of FJSP-FCR with zero unavailable periods. Table 3 shows the results for the 10 instances of Fattahi, where WLH is the proposed model shown in Section 2, n is the number of jobs, o is the number of operations and m the number of machines.

The second experiment involving the MILP model is focused on the new set of FJSP-FCR instances, available at (*https://url.contains.names.add.after.review*). In order to

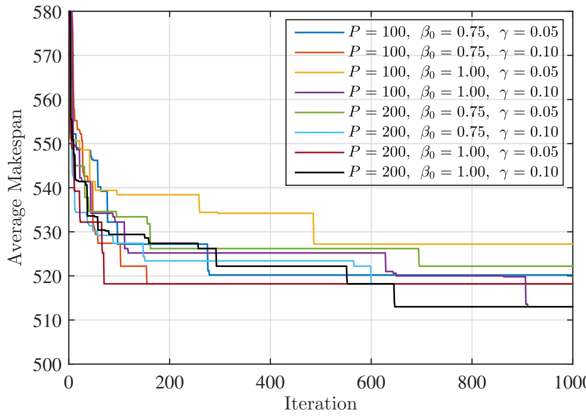

Figure 2: Convergence of the makespan for the instance FCR01 based on different parameter configurations.

Instance	n	o	m	u	Model 1		Model 2		GA		DFA	
					C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU
FCR01	5	3	6	6	513	0.20	513	0.15	513	3.57	513	0.13
FCR02	5	3	7	9	548	0.56	548	0.34	552	7.18	548	0.16
FCR03	6	3	7	14	620	2.50	620	1.96	620	5.80	620	0.44
FCR04	7	3	7	17	746	27.46	746	7.11	748	5.86	746	2.33
FCR05	7	3	7	20	693	20.94	693	5.83	709	11.23	693	4.28
FCR06	8	3	7	20	774	4.83	774	3.45	777	11.31	774	6.17
FCR07	8	4	7	12	[1000;1024] 2.34%	3600	1024	599.52	1044	28.46	1024	10.34
FCR08	9	4	8	35	[1414;1467] 3.61%	3600	1418	2905	1478	35.22	1418	16.78
FCR09	11	4	8	49	[1410.96;2051] 31.21%	3600	[1394.65;1991] 29.95%	3600	1976	65.65	1944	34.70
FCR10	12	4	8	52	[1815.26;2631] 31.00%	3600	[1875;2492] 24.76%	3600	2337	70.64	2320	43.01
FCR11	10	6	6	6	327	2.43	327	1.82	335	7.06	327	4.53
FCR12	10	6	6	8	[27;30] 10.00%	3600	[27;28] 3.57%	3600	30	35.87	28	20.65
FCR13	15	10	8	17	[85;554] 84.65%	3600	[95;464] 79.52%	3600	464	56.51	464	38.11
FCR14	15	9	8	20	[52.66;74] 28.82%	3600	[56.67;74] 23.41%	3600	75	35.95	74	21.75
FCR15	15	9	4	11	[85;520] 83.65%	3600	[83;386] 78.49%	3600	307	40.74	304	33.87
FCR16	10	15	15	42	[67.33;142] 67.33%	3600	[93;116] 19.82%	3600	120	62.16	116	40.42
FCR17	20	5	5	8	[123;213] 42.25%	3600	[123;213] 42.25%	3600	210	49.18	184	32.34
FCR18	20	14	10	42	[923;1344] 31.32%	3600	[923;1145] 19.38%	3600	1125	64.23	1117	45.67

Table 5: The experimental results on the new FJSP-FCR instances.

compare the MILP model, we expanded OOY model and added the availability constraints to deal with our problem. The additional constraints added to the OOY model are similar to the Equations 3, 7, and 8. Additional details of the OOY model are omitted to conserve space. Table 5 shows the results of the experiments of the models with the FJSP-FCR instances, where *Model 1* is the extended OOY model, *Model 2* is our proposed model shown in Section 2, n is the number of jobs, o is the number of operations, m is the number of machines and u is the number of unavailable periods.

4.2 DFA Experiments

We implemented the proposed DFA algorithm in C++ and the experiments were run on a Intel Core i7 2.70GHz, with 8 GB of RAM memory. We used the 10 largest Fattahi *et al.* instances for the first experiment. The bounds provided by the MILP models are used for comparison. Table 3 shows the results for the 10 Fattahi *et al.* instances. In this experiment, we can see that DFA is efficient and can obtain the best solutions for all instances.

To better demonstrate the effectiveness of the DFA we compare results with other state-of-the-art algorithms for the FJSP using the Brandimate instances. In order to compare efficiency, we additionally describe the programming language, CPU, RAM memory and the original computational time for the each algorithms. We compare the DFA with an artificial bee colony algorithm (TABC) [Gao *et al.*, 2015] and a memetic algorithm (MA) [Yuan and Xu, 2015]. 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 DFA parameters used in this experiment are $P = 150, G = 1000, \beta_0 = 1.0, \gamma = 0.1, \alpha = 1.0$, where P is the number of fireflies and G the number of iterations. Table 4 shows the comparison on the 10 Brandimarte instances.

Due the lack of literature, in order to compare the DFA with other algorithms using the FJSP-FCR instances, we implemented a GA proposed in [Lunardi and Voos, 2018]. The parameters used in this experiment are $P = 100, G = 1000, \beta_0 = 1.0, \gamma = 0.1, \alpha = 1.0$, where P is the number of firefly and G is the number of iterations. The results of this experiment are presented in the Table 5.

We use the FCR01 as problem instance we draw in the Figure 2 the convergence of the makespan of the global best firefly (average over 10 runs) using 8 different parameter configuration. Each configuration is composed of P, β_0, and γ, where the iterations $G = 1000$ and $\alpha = 1$. The best parameter configuration for the particular instance is $\beta_0 = 1.0$ and $\gamma = 0.1$. The best makespan, equal to 513, is reached after 646 generations with $P = 200$ and after 911 generations with $P = 100$.

5 Conclusion

In the present paper, we extended the definition of the FJSP taking into account availability constraint with *crossable* unavailable periods and *resumable* operations. We put forward a new MILP model and a discrete firefly algorithm for the FJSP-FCR. In order to evaluate the performance of the solution methods 18 new instances, so-called FCR01,...,FCR18, have been defined and their access is public. 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 very important for comparisons with non-exact methods, and achieved better results when compared with other models from the literature. The experiments among the DFA and others recently published algorithms shows that it is a feasible approach for the considered problem.

References

[Birgin *et al.*, 2014] Ernesto G Birgin, Paulo Feofiloff, Cristina G Fernandes, Everton L De Melo, Marcio TI Oshiro, and Débora P Ronconi. A milp model for an extended version of the flexible job shop problem. *Optimization Letters*, 8(4):1417–1431, 2014.

[Brandimarte, 1993] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3):157–183, 1993.

[Demir and İşleyen, 2013] Yunus Demir and S Kürşat İşleyen. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3):977–988, 2013.

[Fattahi *et al.*, 2007] Parviz Fattahi, Mohammad Saidi Mehrabad, and Fariborz Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of intelligent manufacturing*, 18(3):331, 2007.

[Gao *et al.*, 2006] Jie Gao, Mitsuo Gen, and Linyan Sun. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, 17(4):493–507, 2006.

[Gao *et al.*, 2015] Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Tay Jin Chua, Chin Soon Chong, Tian Xiang Cai, and Qan Ke Pan. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert systems with applications*, 42(21):7652–7663, 2015.

[Karthikeyan *et al.*, 2015] S Karthikeyan, P Asokan, S Nickkolas, and Tom Page. A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. *International Journal of Bio-Inspired Computation*, 7(6):386–401, 2015.

[Kuo *et al.*, 2009] I-Hong Kuo, Shi-Jinn Horng, Tzong-Wann Kao, Tsung-Lieh Lin, Cheng-Ling Lee, Takao Terano, and Yi Pan. An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert systems with applications*, 36(3):7027–7032, 2009.

[Lunardi and Voos, 2018] Willian Tessaro Lunardi and Holger Voos. Comparative study of genetic and discrete firefly algorithm for combinatorial optimization. *33rd ACM/SIGAPP Symposium On Applied Computing, Pau, France, April 9-13, 2018*, 2018.

[Özgüven *et al.*, 2010] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, 2010.

[Rajkumar *et al.*, 2011] M Rajkumar, P Asokan, N Anilkumar, and T Page. A grasp algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, 49(8):2409–2423, 2011.

[Saidy *et al.*, 2008] Dehnar Saidy, Hamid Reza, and Mohammad Taghi Taghavi-Fard. Study of scheduling problems with machine availability constraint. *Journal of Industrial and Systems Engineering*, 1(4):360–383, 2008.

[Vilcot and Billaut, 2011] Geoffrey Vilcot and Jean-Charles Billaut. A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. *International Journal of Production Research*, 49(23):6963–6980, 2011.

[Wang and Yu, 2010] Shijin Wang and Jianbo Yu. An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Computers & Industrial Engineering*, 59(3):436–447, 2010.

[Yang, 2010] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[Yuan and Xu, 2015] Yuan Yuan and Hua Xu. Multiobjective flexible job shop scheduling using memetic algorithms. *IEEE Transactions on Automation Science and Engineering*, 12(1):336–353, 2015.

[Zhang *et al.*, 2017] Jian Zhang, Guofu Ding, Yisheng Zou, Shengfeng Qin, and Jianlin Fu. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, pages 1–22, 2017.

[Zribi *et al.*, 2008] Nozha Zribi, Abdelkader El Kamel, and Pierre Borne. Minimizing the makespan for the mpm job-shop with availability constraints. *International Journal of Production Economics*, 112(1):151–160, 2008.