Local Search and Initialization in the Firefly Algorithm: Performance Analysis in Solving the Flexible Job-Shop Scheduling Problem

N. Alvarez-Gil, R. Rosillo, D. De la Fuente, and R. Pino

Abstract Hybrid metaheuristics are becoming a widely used alternative to solve some combinatorial optimization problems such as the Flexible Job-shop Scheduling Problem (FJSP). The inherent complexity of this type of problem requires methods that can find near optimal solutions in a reasonable computational time, since exact methods may be impractical in the real industry because of their exhaustive nature. Here is where metaheuristics, which have been proved to be very time-efficient in providing *quality* solutions, play a key role. Nevertheless, they also present some shortcomings like premature convergence and local optima stagnation. Hybrid versions are commonly used to avoid these issues and increase its search capability. In this paper, we conduct a comparative study of the performance of the Firefly Algorithm and two variants, one improved with an initialization phase and another that integrates both this initialization and multiple local search structures, in solving state-of-the-art FJSP instances. The study demonstrates how local search and initialization can notably enhance the performance of the algorithm.

Keywords Firefly algorithm · Local search · Initialization

N. Alvarez-Gil (✉) · R. Rosillo · D. De la Fuente · R. Pino
Departamento de Administración de Empresas, Escuela Politécnica de Ingeniería de Gijón, Universidad de Oviedo, Oviedo, Spain
e-mail: uo226901@uniovi.es

R. Rosillo
e-mail: rosillo@uniovi.es

D. De la Fuente
e-mail: david@uniovi.es

R. Pino
e-mail: pino@uniovi.es

D. De la Fuente et al. (eds.), *Organizational Engineering in Industry 4.0*, Lecture Notes in Management and Industrial Engineering,
https://doi.org/10.1007/978-3-030-67708-4_9

1 Introduction

Combinatorial optimization problems have been the focus of many scientific research because of their complexity and practical importance in a wide variety of fields, and different algorithms have been developed for their resolution. These algorithms can be classified as complete and approximate algorithms [1], the former being algorithms that ensure to find the optimal solutions.

Nevertheless, when the problem is NP-hard [2], complete algorithms require an impractical amount of time, and thus approximate algorithms (i.e. metaheuristics) are the alternative commonly used in the real world. Approximate algorithms cannot guarantee to find the optimal solution but they notably reduce the amount of time required to provide good solutions [1].

The Job-shop Scheduling Problem (JSP) is "not only NP-hard, but it also has the well-earned reputation of being one of the most computationally stubborn combinatorial problems (…)" [3], and hence it is a problem that elicits interest in areas like planning and managing of manufacturing processes or operations research. In the JSP, there is a set of jobs that has to be performed and each job consists in a set of operations that must be processed in exactly one machine and the operations are subjected to precedence constraints. The goal of the JSP is to find a sequence of the operations that optimizes certain criteria, e.g. minimize the completion time, minimize the total machines' workload, etc.

The Flexible Job-shop Scheduling Problem (FJSP) is an extension of the JSP in which, in addition to the sequencing, a further decision level is required, the assignment: the operations that make up each job can be processed by any machine from a given set of compatible machines, and thus it is required to assign each operation to one machine. This fact makes the FJSP even more complex than the JSP.

Many approaches have been proposed for the resolution of the FJSP, especially (meta)heuristics algorithms. To name a few within the most relevant ones, Brandimarte [4] described a hierarchical approach for the FJSP, solving separately the sequencing and the assignment problems, both tackled by a tabu search (TS) algorithm. TS was also used by Mastrolilli and Gambardella [5] where, additionally, two neighborhood functions are introduced. Kacem et al. [6] presented a hybrid approach combining evolutionary algorithms and fuzzy logic for solving the FJSP with multiple objectives. In Pezzella et al. [7], a genetic algorithm (GA) with different strategies for initializing the population and for the selection and reproduction of the individuals is presented. Regarding constructive metaheuristics, particle swarm optimization and ant colony optimization has also been applied for this problem (see [8, 9], respectively).

In this paper, we present a comparative study of the performance of different versions of the firefly algorithm (FA) in optimizing the total completion time (makespan) in the FJSP. With this purpose, we have developed a standard discrete version of the FA, another version in which some strategies to generate the initial population are used, and one hybrid version in which, together with this initialization, multiple local search procedures are integrated to increase the performance of the algorithm. These three versions have been tested with multiple state-of-the-art FJSP instances, providing a detailed analysis of the results.

The rest of the paper is organized as follows. In Sect. 2, a general description of the FJSP is provided. In Sect. 3, we describe the discrete version of the FA, the initialization procedure, and the local search strategies. The computational results of the tests and the comparative analysis are summarized in Sect. 4. Finally, Sect. 5 closes the paper with the conclusions obtained in the study.

2 Problem Definition

In the FJSP, there are n jobs, consisting each job J_i $(1 \leq i \leq n)$ in a sequence of n_i operations and m machines. An operation O_{ij} $(i = 1,2, \ldots, n; j = 1,2, \ldots, n_i)$ needs to be performed on one machine m_{ij} from the set of available machines M_{ij}. Each machine $k \in M_{ij}$ requires a certain processing time (P_{ijk}) to perform an operation O_{ij}.

Some assumptions are made: an operation cannot be interrupted (a_1); there are precedence constraints defined for any pair of operations within a job (a_2); machines are independent of each other (a_3); there are no precedence relations between jobs (a_4); transport and set-up times are already considered in P_{ijk} (a_5); and each machine can process at most one operation at any time (a_6).

The mathematical model could be given as follows:

$$\min f = \max_{1 \leq k \leq m} (C_k) \tag{1}$$

subject to

$$C_{ij} - C_{i(j-1)} \geq P_{ijk} X_{ijk},\ j = 2, \ldots, n_i;\ \forall i, j \tag{2}$$

$$[(C_{hg} - C_{ij} - P_{hgk}) X_{ghk} X_{ijk} \geq 0] \vee [(C_{ij} - C_{hg} - P_{ijk}) X_{hgk} X_{ijk} \geq 0], \forall i, j, h, g, k \tag{3}$$

$$\sum_{k \in M_{ij}} X_{ijk} = 1, \forall i, j \tag{4}$$

$$X_{ijk} \in \{0, 1\}, \forall i, j, k \tag{5}$$

Equation (1) ensures the minimization of the objective, the *makespan* (i.e. the maximal completion time of all the jobs). Constraint a_2 is ensured by Inequality (2) and constraint a_6 by Inequality (3). Equation (4) indicates that, for each operation, only one machine can be selected. Equation (5) is the binary decision variable (“1” if operation O_{ij} is assigned to machine k, “0” otherwise).

3 Firefly Algorithm

The firefly algorithm (FA) is a swarm intelligence algorithm inspired by the social behavior of the fireflies. The fireflies use their flashing lights to attract others with predation or mating purposes. It was originally developed for solving continuous optimization problems by Yang [10].

The main aspect of the FA is the association of the fireflies' light intensity with the objective function of the optimization problem. It is assumed that the firefly brightness (I) determines its attractiveness (β), both being in turn linked with the encoded objective function [11], the makespan in this study. If the aim is to minimize an objective, the objective function value of a firefly at a position x can be inversely associated with its brightness $I(x) \propto 1/f(x)$.

Fireflies are initially spread over the search space and each position stands for a solution to the optimization problem. At each iteration of the algorithm, the fireflies will move toward the brighter ones (i.e. for minimization problems, solutions with lower objective function value) within a certain region of the search space, depending on their distance and visibility. The fireflies' movement driven by its brightness, plus a random movement component, allows efficiently exploring the search space. A more detailed explanation of the original FA can be found in [10, 12].

This section is focused on the discrete version of the FA, which is required for the FJSP. The main aspects of the FA that need to be adapted are the representation of the solutions, the calculation of the distance, and how the movement is performed. The discrete approach presented in this paper is based on [11].

Solution Representation and Decoding. Each solution of the problem is obtained from two different vectors, one for each FJSP subproblem. The Sequencing Vector (SV) indicates the sequence of the operations and each job number J_i appears n_i times. The Assignment Vector (AV) denotes the machine assigned to each operation, an item AV[i] being a machine index. Both vectors have a length equal to the total number of operations, $\sum_{i=1}^{n} n_i$. Figure 1 shows the decoding of a solution from its AV and SV.

First, SV gives the sequence of the operations. The jth operation of J_i (O_{ij}) corresponds to the jth time a job index J_i appears. Then the machines m_{ij} assigned for each operation O_{ij} are obtained from the AV. The combination of these two vectors provides the final solution of a firefly: $\{(O_{11}, M_1), (O_{12}, M_1), (O_{31}, M_3), (O_{21}, M_1), (O_{32}, M_2), (O_{33}, M_3), (O_{13}, M_4), (O_{22}, M_2), (O_{23}, M_3), (O_{24}, M_1)\}$.

Measurement of the Distance Between Two Fireflies. The distance between the SVs of two fireflies can be carried out as the minimum number of swaps required

AV	**1**	**1**	**4**	**1**	**2**	**3**	**1**	**3**	**2**	**3**
	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}

SV	**1**	**1**	**3**	**2**	**3**	**3**	**1**	**2**	**2**	**2**
	O_{11}	O_{12}	O_{31}	O_{21}	O_{32}	O_{33}	O_{13}	O_{22}	O_{23}	O_{24}

Solution:

(O_{11},M_1) (O_{12},M_1) (O_{31},M_3) (O_{21},M_1) (O_{32},M_2) (O_{33},M_3) (O_{13},M_4) (O_{22},M_2) (O_{23},M_3) (O_{24},M_1)

Fig. 1 Solution representation and decoding

Table 1 Distances and movement

P	AV = [2 4 3 1 3 1 4 4 1 2]	SV = [1 2 2 3 1 2 1 3 3 2]
P_{best}	AV = [2 1 3 4 1 1 4 4 1 2]	SV = [1 3 2 2 1 2 1 2 3 3]
d_{av} and d_{sv}	{(2,1), (4,4), (5,1)}	{(2,4), (8,10)
$\lvert d_{av}\rvert$ and $\lvert d_{sv}\rvert$	3	2
$\beta(r)$	0.53	0.71
rand $\in$ [0,1]	{0.34, 0.17, 0.76}	{0.09, 0.82}
Mov. β-step	{(2,1), (4,4)}	{(2,4)}
Position after β-step	2 $\underline{1}$ 3 $\underline{4}$ 3 1 4 4 1 2	1 $\underline{3}$ 2 $\underline{2}$ 1 2 1 3 3 2
Position after α-step	2 1 3 4 3 1 4 4 $\underline{2\ 1}$	1 3 2 2 1 $\underline{1\ 2}$ 3 3 2

to bring one SV closer to the most attractive one ($\lvert d_{sv}\rvert$). For AV components, the distance is the number of non-corresponding items in the sequence, known as the Hamming distance ($\lvert d_{av}\rvert$). Table 1 shows how both distances between the AVs and the SVs of two fireflies (P and P_{best}) are calculated.

Fireflies' Movement. The movement of the original FA is divided into two not interchangeable steps: First, the β-step and then the α-step. The β-step is an insertion and pair-wise exchange mechanism used to bring the AV and SV of a firefly closer to the global best firefly. The β-step consist in the following sub-steps: all necessary pair-wise exchanges in SV and insertions in AV are found and stored in d_{sv} and d_{av}, respectively; the distances $\lvert d_{sv}\rvert$ and $\lvert d_{av}\rvert$ are stored in R; the probability $\beta(r) = \beta_o/(1 + \gamma r^2)$ is computed; a random number rand $\in$ [0,1] is generated for each element of d_{av} and d_{sv}; and then the corresponding insertion/pair-wise exchange is performed if $Rand \leq \beta$. The α-step is a swapping mechanism in which two non-equal element positions are chosen at random and swapped. Table 1 shows how the different steps of a firefly movement are performed, with $\beta_o = 1$, $\gamma = 0.1$, and $\alpha = 1$.

3.1 Initialization Module

In order to study how the initial population can impact the performance of the FA, we have implemented an initialization module. This module aims to initially locate the fireflies in promising areas of the search space instead of doing it randomly. The initialization rules explained below are the same used in the GA of Pezzella [8] who, in turn, follow the *approach by location* of Kacem et al. [13].

For AVs, two different rules are used: Ar_1 and Ar_2, both considering the processing times and the machines' workload. Ar_1 works as follows: the minimum processing time P_{ijk} is selected, and the machine k is assigned to the operation O_{ij}. Then all the columns corresponding to machine k are updated with P_{ijk}, and the process is repeated until all the operations have been assigned to a machine. Ar_2 works similarly but, before starting, all rows (i.e. operations) and columns (i.e. machines) are randomly shuffled. Then, instead of selecting the minimum of the table, the minimum P_{ijk} of the new first row is selected, then the minimum of the second row, and so on, updating the columns as it was explained for $Ar_{1.}$ The advantage of Ar_2 is that different assignments can be obtained in different runs of the algorithm.

For SVs, the well-known sequencing dispatching rules, the Most Work Remaining (MWR), and the Most number of Operations Remaining (MOR) are applied. In addition, to enhance diversity, some of the initial SVs are randomly generated.

3.2 Local Search Module

To enhance the search performance of the FA and to avoid common problems of the metaheuristics such as premature convergence or local optima stagnation, we have introduced several local search procedures into the algorithm. These local search (LS) strategies are based on the neighborhood structures used in the variable neighborhood search (VNS) algorithm presented in [14].

LS Strategy 1. Two element positions of the SV are selected at random and swapped, ensuring that no precedence constraint is violated. This is repeated multiple times depending on the total time of operations.

LS Strategy 2. A random operation is selected from the AV, and a different machine of the set of available machines for that operation is assigned, randomly as well. This step is repeated depending on the total time of operations. It can be noticed that LS strategies 1 and 2 are very similar to the α-step of the discrete FA.

LS Strategy 3. Two jobs are selected at random, and the positions of all its operations are swapped maintaining the precedence relations within each job, while the machine assignments remain the same.

LS Strategy 4. One operation assigned to the machine with the maximal workload (e.g. the sum of the processing times of the operations assigned to that machine) is randomly selected, and then it is assigned to the machine with the least workload, if possible. If not, it is assigned to any random available machine.

LS Strategy 5. One random operation assigned to the machine spending the maximum time to complete its assigned operations (equal to makespan) is selected, and then it is assigned to the machine spending the minimum time, if possible. If not, it is assigned to any random available machine.

4 Results and Comparative Study

This section describes the computational study conducted to compare how differently the FA performs with the initialization and the local search. We implemented three different versions of the FA: a standard discrete FA (V_S), another version integrated with the initialization module (V_I), and one more version with both the initialization and local search modules (V_{LS}). The algorithms were implemented in Python 3 and the tests were run on an Intel Core 7 2.1 GHz PC with 8 GB RAM memory.

Table 2 shows the average results (*Cmax*-Av.), the best results (*Cmax*-Best), and the average and best relative time of 30 runs of the three FA versions (V_S, V_I, and V_{LS}) for six different FJSP instances. $N\text{x}m\text{x}n_i$ stands for the number of jobs (n), the numbers of machines (m), and the total number of operations (n_i). Large instances were selected for the study because it is where more differences in the performance of algorithm's versions exist. Behnke and Geiger [15] provide a detailed explanation of the instances, from where we took the best-known results (BKR). *Av.Rel* and *Best.Rel* were calculated as $(tVx-tVs)/tVs$, tVx being the time spent by version x ($x = V_I$, V_{LS}) in reaching the maximal numbers of generations allowed (Max_{Gen}), which is the termination criteria. The parameters of the FA are number of fireflies $nf = 200$,

Table 2 Comparative study

				Cmax		Time	
Instance	*n* x *m* x *ni*	BKR	Ver	Av	Best	Av.Rel	Best.Rel
Mk1	10 x 6 x 55	40	V_S	50.4	48	–	–
			V_I	47.7	46	0.007	0.118
			V_{LS}	43	42	−0.008	−0.050
Mk3	15 x 8 x 150	204	V_S	246.5	231	–	–
			V_I	243.6	237	−0.127	−0.227
			V_{LS}	219.7	**204**	−0.255	−0.264
MFJS2	5 x 7 x 15	446	V_S	474.43	456	–	–
			V_I	460.96	**446**	−0.002	0.014
			V_{LS}	453.33	**446**	−0.026	−0.017
MFJS3	6 x 7 x 18	466	V_S	549.4	507	–	–
			V_I	530.4	497	−0.003	−0.005
			V_{LS}	497	**466**	−0.030	−0.035
Kacem 2	10 x 7 x 29	11	V_S	24.9	***21***	–	–
			V_I	14	13	−0.002	−0.004
			V_{LS}	11.96	**11**	−0.012	−0.008
Kacem 3	10 x 10 x 30	7	V_S	20.6	18	–	–
			V_I	8.03	**7**	−0.006	−0.004
			V_{LS}	8	**7**	−0.009	−0.008

$\text{Max}_{\text{Gen}} = 50$, $\beta_o = 1$, $\gamma = 0.1$, and $\alpha = 1$. For the initialization module: $Ar_1 = 20\%$, $Ar_2 = 80\%$, MWR $= 40\%$, MOR $= 40\%$, and random $= 20\%$.

It can be noticed how V_{LS} notably and consistently outperforms V_S and V_I (Table 2), both in average and best values, reaching the BKR for almost all the tested instances. It was expected since V_{LS} is the most complete version, using both the initialization and local search modules. Nevertheless, what catches our attention is that the time spent by V_{LS} is almost the same or lower than for V_S and V_I. As a preliminary hypothesis, we believe a possible reason is that the more different the fireflies are, the more computational time for the distance calculation and movement is required. Hence, when applying local search, many solutions are neighborhood solutions of others, and the distance between them is very low, requiring less time to compute it and perform the movement. Another explanation may be how we apply the local search to the firefly's population. To each solution from the first $nf/2$ set (better fireflies), we randomly apply one of the LS strategies. But, at each iteration, the second $nf/2$ (worse fireflies) is renewed with solutions obtained after applying one of the LS strategies to the best solution obtained so far, and thus half of the population are neighborhood solutions of the best one (more focused in the exploitation), while the other half is used for the exploration of promising areas.

5 Conclusions

It is common to look for improvements in the efficiency of metaheuristics algorithms by adding some kind of problem-specific strategies or knowledge. This allows to better explore the search space obtaining quality solutions to the optimization problem in a shorter time. In this work, the original Firefly Algorithm was enhanced with an initialization phase and some different local search procedures for solving the FJSP, aiming to analyze how these two upgrades affect its performance. With this purpose, we have explained and implemented three different versions of the FA—the original discrete version, another version with an initialization phase, and one more version with both the initialization and the local search modules—and compared them in the resolution of some middle- and large-sized state-of-the-art FJSP instances.

Computational results confirmed that the most complete version, the one that starts the search from solutions obtained from the initialization phase and that uses the different local search strategies during the search, consistently outperforms the other two versions, reaching the best known results in most of the tested cases. Future work will be focused on expanding this study to more FJSP instances and on studying further techniques to speed up the algorithm.

References

1. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv 35(3):268–308
2. Garey MR, Johnson DS (1979) Computers and intractability. Freeman, A Guide to the Theory of NPCompleteness. W.H
3. Applegate D, Cook W (1991) A computational study of the Job-shop scheduling problem. J Comput 3(2):149–156
4. Brandimarte P (1993) Routing and scheduling in a flexible job shop by taboo search. Ann Oper Res 41:157–183
5. Mastrolilli M, Gambardella LM (1996) Effective neighborhood functions for the flexible job shop problem. J Sched 3:3–20
6. Kacem I, Hammadi S, Borne P (2002) Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60:245–276
7. Pezzela F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the Flexible Job-shop scheduling problem. J Comput Oper Res 35:3202–3212
8. Zhang G, Shao X, Li P, Gao L (2009) An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 56:1309–1318
9. Wang L, Cai J, Li M, Liu Z (2017) Flexible job shop scheduling problem using an improved ant colony optimization. Sci Program. https://doi.org/10.1155/2017/9016303
10. Yang X (2008) Nature-inspired metaheuristic algorithm. Luniver Press, Bristol
11. Karthikeyan S, Asokan P, Nickolas S (2014) A hybrid discrete firefly algorithm for multi-objective flexible job shop scheduling problem with limited resource constraint. Int J Adv Manuf Technol 72:1567–1579
12. Yang XS (2009) Firefly algorithm for multimodal optimization. Stochastic Algorithms: Found Appl 5792:169–178
13. Kacem I, Hammadi S, Borne P (2002) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybernet Part C 32(1):1–13
14. Yazdani M, Amiri M, Li P, Zandieh M (2009) Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Exp Syst Appl 37:678–687
15. Behnke D, Geiger MJ (2012) Test instances for the flexible job shop scheduling problem with work centers. Research report, Helmut-Schmidt-University. ISSN 2192–0826