Parallelism and Hybridization in Differential Evolution to solve the Flexible Job Shop Scheduling Problem

Paralelismo e hibridización en un algoritmo de evolución diferencial para resolver el problema de planificación job shop flexible

Franco Morero[1] , Carlos Bermúdez[1] , and Carolina Salto[1,2]

[1]*LISI - Facultad de Ingeniería, Universidad Nacional de La Pampa, General Pico, Argentina*
{bermudezc,saltoc}@ing.unlpam.edu.ar
[2]*CONICET, Argentina*

Abstract

The Flexible Job Shop Scheduling Problem (FJSSP) is one of the most challenging combinatorial optimization problems, with practical applicability in a real production environment. In this work, we propose a simple Differential Evolution (DE) algorithm to tackle this problem. To represent a FJSSP solution, a real value representation is adopted, which requires a very simple conversion mechanism to obtain a feasible schedule. Consequently, the DE algorithm still works on the continuous domain to explore the problem search space of the discrete FJSSP. Moreover, to enhance the local search ability and to balance the exploration and exploitation capabilities, a simple local search algorithm is embedded in the DE framework. Also, the parallelism of the DE operations is included to improve the efficiency of the whole algorithm. Experimental results confirm the significant improvement achieved by integrating the modifications introduced in this study. Additionally, test results show that our algorithm is competitive when compared with most existing approaches for FJSSP.

Keywords: differential evolution, flexible job shop scheduling, parallelism

Resumen

El problema de planificación *job shop* flexible (FJSSP, en inglés) es uno de los problemas de optimización más desafiantes, con aplicabilidad práctica en ambientes de producción real. En este trabajo, se propone un algoritmo de evolución diferencial (DE, en inglés) simple para resolver el problema en cuestión. Para representar una solución al FJSSP, se adopta una representación de vectores reales, la cual requiere un mecanismo de conversión muy simple para obtener una planifición factible. Consecuentemente, el algoritmo DE continúa trabajando en el dominio continuo para explorar el espacio de búsqueda del problema que es de caracter discreto. Además, para mejorar la habilidad de búsqueda local y lograr un equilibrio entre la exploración y explotación, se incorpora un algoritmo de búsqueda local simple. También, se incluye paralelismo a las operaciones del DE para mejorar la eficiencia del algoritmo. Los resultados obtenidos confirman que el rendimiento del DE mejora en forma significativa al incorporar las propuestas incluidas en este estudio. Además, los resultados de las pruebas muestran que nuestro algoritmo es competitivo en comparación con la mayoría de los existentes.

Palabras claves: evolución diferencial, paralelismo, planificación de trabajos

1 Introduction

One of the main goals in modern manufacturing systems is efficiency. In this sense, the production scheduling can be considered an important issue. In this field, the job shop scheduling problem (JSSP) [1] is one of the most important and difficult problems. JSSP consists of jobs and machines, where each job consists of operations to be processed in a given order, and each operation is processed by a specific machine. The objective is to find an operation sequence on the machines (schedule) so that the time needed to complete all jobs is minimized.

The supposition that one machine only processes one type of operation does not reflect the reality of the modern manufacturing systems. The possibility of selecting alternative routes among the machines is useful in production environments where multiple machines, possibly not identical, can perform the same operation (perhaps with different processing times). This flexibility allows the system to absorb changes in work demands or in machine performances. When this factor is included, the problem is known as the Flexible Job Shop Scheduling Problem (FJSSP), becoming in a more realistic production environment and with practical applicability.

The solution of FJSSP involves two decisions: to sequence the operations on the machines and to assign each operation to the appropriate set of machines to minimize the elapsed time to complete all the jobs

(makespan or C_{max}). These decisions suggest that FJSSP is a complex optimization problem (NP-hard problem [2]), consequently, the adoption of meta-heuristic [3, 4] has led to better results than classical dispatching or greedy heuristic algorithms [5, 6, 7]. Since introduced in 1997 by Storn and Price [8], the Differential Evolution (DE) metaheuristic became very popular among computer scientists and practitioners almost immediately after its original definition.

DE is a population-based evolutionary algorithm, utilizing real-valued vectors as a population for each generation. DE employs simple mutation and crossover operators to generate new candidate solutions and applies a one-to-one competition scheme to greedily determine whether the new candidate or its parent will survive in the next generation. Besides, the canonical DE requires very few control parameters, a feature that makes it easy to use for the practitioners. Consequently, their success is due to its simplicity and ease implementation, and reliability and high performance. DE algorithms have been applied to many combinatorial optimization problems ([9, 10, 11, 12], among many others), but as far as we are aware, there is few published research work that describes the use of DE to deal with FJSSP [13].

In [14], we design a simple DE to solve FJSSP. As DE was originally devised for solving continuous optimization problems, we adopt a real value representation for FJSSP to make the continuous DE applicable for solving the discrete FJSSP. This implies that algorithmic operations (mutation and recombination) should not be modified or adapted to solve the problem at hand. Another important feature of DE is the little number of parameters to be set when compared to other evolutionary algorithms: with only three parameters, the algorithm behavior can be controlled. However, the success to find good solutions to a problem depends on discovering the correct values of these parameters [15]. Therefore, we analyze this line to determine the adequate values for these parameters for solving the FJSSP instances. Moreover, a simple local search procedure is embedded in the DE to improve their exploration capacities by solving the problem. Finally, parallelism at algorithmic level [3] is incorporated into the DE framework to improve its scalability and reduce the computation time. Resuming, the contributions of this work are:

- design of a simple DE to solve FJSSP in the real-value search space without affecting the efficacy

- improvement of the DE performance with a simple and efficient local search procedure

- proposal of a parallel DE to deal with efficiency issues

The experimental methodology we have followed consists of computing the makespan or C_{max} values for the proposed DE and their improvements to solve FJSSP. The obtained results are compared by considering different quality indicators. The current work is an extension of the previous article [14], where we extend and reorganize the problem and algorithm description. Moreover, another DE parameter is considered in the experimental study to determine its appropriate value to solve FJSSP. Consequently, both the experimentation and the result analysis are enlarged, also including other metrics to improve the comprehension of the relation between the solution quality and the DE computational effort.

The paper is organized as follows. In Section 2, we introduce the problem formulation and show an illustrative example of input data. In Section 3, we present the basic DE algorithm. In Section 4 we explain our proposal based on DE to solve FJSSP. In the following section, we introduce the experimental design and in Section 6, we evaluate the results. Further, in Section 7 we make a comparison between the proposed DE and other solvers present in the literature. Some final remarks and future research directions are given in Section 8.

2 The Flexible Job Shop Scheduling Problem

FJSSP is an extension of the classic JSSP where a set of machines, not necessarily identical, can process an operation. Consequently, a set of available machines is provided for each operation. The goal is to decide on which machine each operation will be assigned and in what order the operations will be sequenced on each machine so that the makespan is minimized.

FJSSP can be formally described as follows. A set $J = \{J_1, J_2, \ldots, J_n\}$ of independent jobs and a set $U = \{M_1, M_2, \ldots, M_m\}$ of machines are given. A job J_i is broken down by a sequence of $O_{i1}, O_{i2}, \ldots, O_{in_i}$ operations to be performed one after the other according to the given sequence. Each operation O_{ij} can be executed on any among a subset $U_{ij} \subseteq U$ of compatible machines. For this reason, the FJSSP can be classified into two categories, partial (P-FJSSP) and total (T-FJSSP) [16]. We have partial flexibility whether exists a proper subset $U_{ij} \subset U$, for at least one operation O_{ij}, while we have $U_{ij} = U$ for each operation O_{ij} in the case of total flexibility. The processing time of each operation is machine-dependent. Pre-emption is not allowed, i.e., each operation must be completed without interruption once started. Furthermore, the machines cannot perform more than one operation at a time. All jobs and machines are available at time 0.

The problem is to assign each operation to an appropriate machine (routing problem), and sequence the operations on the machines (sequencing problem) to minimize the makespan (C_{max}). This measure is the time needed to complete all the jobs, which is defined as $C_{max} = max_i\{C_i\}$, where C_i is the completion time of the job J_i. Table 1 shows an instance of FJSSP with 3 jobs, 4 machines and 8 operations. The rows

Table 1: Instance Example for FJSSP

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

and columns correspond to machines and operations, respectively, and the entries of the table are the processing times.

3 DE: Background

Storn and Price [8, 17] proposed the Differential Evolution (DE) algorithm to solve global optimization problems over continuous search spaces. Due to its simple algorithmic framework and inexpensive computation in terms of CPU time but performing well in convergence, DE has emerged as one of the most competitive and versatile families of evolutionary algorithms. Moreover, DE has been widely applied and shown its strengths to solve a wide variety of very complex problems from diverse domains of science and technology [18, 19].

DE is a population-based optimization method, having a very simple algorithmic structure, whose implementation requires only a few lines of code in any standard programming language. The optimal or near-optimal solution is obtained by an iterative process that is applied to a set of solutions (population) to achieve a new one. At each step of the process, known as iteration, new solutions arise as a result of perturbations to the current ones.

The general DE structure shares similar features with other evolutionary algorithms (EAs), such as genetic algorithms (GAs) [20]. However, DE differs markedly from the well-known EAs because the first one reinforces the mutation as the principal perturbation operation. DE mutates the solutions with a scaled difference(s) of distinct members from the current population. Consequently, DE is different in handling distance and direction information to move the population at the current iteration toward the next one, in virtue of constructive cooperation between individuals. The framework of DE is described in Algorithm 1. After the initialization step, an iterative process begins that includes the application of mutation, recombination, and selection operators. The iterations continue until a termination criterion is satisfied. The next sections explain each of the operations performed during the iterative process.

3.1 Initialization

The first step (Line 1 of Algorithm 1) consists in the initialization of the population P^0 of N_P target vectors of D real values ($x_i = (x_{i,1}, x_{i,2}, ..., x_{i,D}) \in \mathbb{R}^D (1 \leq i \leq N_P)$). Each vector forms a candidate solution to the multi-dimensional optimization problem. Usually,

---

Algorithm 1 DE Algorithm

Require: F, Cr, N_p
Ensure: x_{best}
1: initialize(P^0, N_p)
2: $g \leftarrow 0$
3: **while** not meet stop criterion **do**
4:     **for** each vector x_i^g from P^g **do**
5:         $v_i^g \leftarrow$ mutate(x_i^g, P^g, F)
6:         $u_i^g \leftarrow$ recombinate(x_i^g, v_i^g, Cr)
7:         $x_i^{g+1} \leftarrow$ select(x_i^g, u_i^g)
8:         add(P^{g+1}, x_i^{g+1})
9:     **end for**
10:     $g \leftarrow g + 1$
11: **end while**
12: $x_{best} \leftarrow$ best_solution(P^g)

---

each $x_{i,j}$ is bounded to a value in the range $[lo_j, lu_j]$, where lo_j, $lu_j \in \mathbb{R}$ are the lower and upper bound, respectively. The N_P target vectors are initialized randomly by applying Equation 1.

$$x_{i,j} = lo_j + U(0,1) \times (lu_j - lo_j) \qquad (1)$$

where $U(0,1)$ is a random number with uniform distribution in the range $[0,1]$.

3.2 Mutation

The mutation operator (Line 5 of Algorithm 1) obtains a donor vector $v_i^g = (v_{i,1}, v_{i,2}, ..., v_{i,D})$ for each target vector x_i^g from the current population P^g ($0 \leq g \leq max_{iter}$) following Equation 2. To obtain v_i^g, a base vector x_{r0}^g and other two vectors x_{r1}^g y x_{r2}^g are randomly selected from P^g, with $r0, r1$ and $r2$ chosen from the set $\{1, 2, ..., N_P\}$ and all of them are mutually exclusive as well as different from i.

$$v_i^g = x_{r0}^g + F \times (x_{r1}^g - x_{r2}^g) \qquad (2)$$

The $F \in [0...1)$ factor, known as scale factor, controls the rate at which the population evolves, in order to avoid their stagnation during the search process. The mutation operator is important to the DE's behaviour because it focuses the search on the most promising areas of the solution space.

3.3 Recombination

The donor vector is modified by the recombination operator (Line 6), to increase the population diversity. This operator creates a trial vector u_i^g through mixing components of the donor vector v_i^g and the target vector x_i^g. The most frequently referred crossover operator is the binomial crossover, which is shown in Equation 3:

$$u_{i,j}^g = \begin{cases} v_{i,j}^g & \text{si } r_j < Cr \vee j = j_r \\ x_{i,j}^g & \text{otherwise} \end{cases} \qquad (3)$$

where $r_j = U(0,1)$ is a random value, j_r is also a random value in the set $\{1, 2, ..., D\}$, and finally, Cr is a parameter known as recombination probability, which controls the fraction of parameter values that are copied from the donor.

3.4 Selection

The last step in the DE's iterative process is the selection operation (Line 7). The trial vector u_i^g competes against the target vector x_i^g regarding their objective values (obtained applying the objective function to each vector). The best vector is selected to be part of the population P^{g+1} of the next generation (Line 8) (see Equation 4). This competition creates a new population with performance equal or superior to the current one. Consequently, DE is an elitist EA.

$$x_i^{g+1} = \begin{cases} u_i^g & \text{if } f(u_i^g) \leq f(x_i^g) \\ x_i^g & \text{otherwise} \end{cases} \tag{4}$$

3.5 Stopping criterion

The stopping criterion can be set to a preset maximum number of iterations (max_{iter}) or some other problem-dependent criterion. Whichever stop of the iterative process, the choice has a direct influence on the best solution x_{best} obtained by the algorithm (Line 12).

3.6 DE Parameters

DE performance mainly depends on three parameters: scaling factor of the difference vector (F), recombination control parameter (Cr), and population size (N_P). Some guidelines are available to choose the control parameters [19]. In this work, N_P is chosen based on previous knowledge and keep it constant during all runs. The factor F usually takes a value that ranges from 0.4 to 1.0 [21]. On the other hand, a good value for Cr is 0.1. However, greater values can be used to speed up convergence. In consequence, a good parameter setting can enhance the algorithm's ability to search for the global optimum or near-optimum with a high convergence rate. This assertion is the driving force behind one of the objectives of this work.
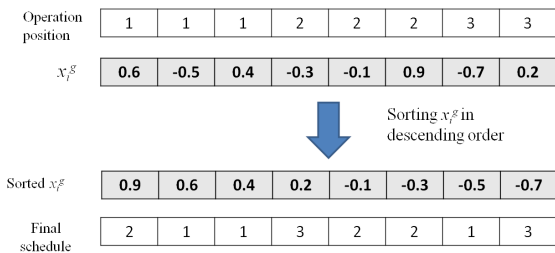

Figure 1: Example of the decoding process used by the DE to solve FJSSP.

4 Our proposal: DE for FJSSP

This section details our proposal to solve FJSSP. To apply the DE algorithm, it is crucial to design a suitable encoding scheme that map real-valued vectors to a feasible solution for FJSSP (see Section 4.1). Moreover,

our proposal is enhanced by a simple local search (Section 4.2). Finally, a parallel version of our proposal is introduced in Section 4.3.

4.1 Representation

In this work, we adopt a random key encoding scheme [22], which was used early with genetic algorithms for sequencing and optimization problems. It is based on random real numbers in a continuous space to encode solutions in a combinatorial space. In this way, the DE algorithm manipulates a real-valued vector to maintain the simplicity and properties in their natural configuration.

For a n-job m-machine scheduling problem, each vector's position (a random key) is a real number in U(-1,1). After a descending order of the random keys, the vector can be translated to a unique list of ordered operations. This procedure always obtains a feasible schedule, which is a permutation with repetitions [23].

Figure 1 presents an example of the decoding process considering the instance shown in Table 1. A fixed ID for each operation is first given following the job number and operation order within the job (represented as an operation position). The order of occurrence for each operation in the final schedule indicates its scheduling priority. Given the vector $x_i^g = [0.6, -0.5, 0.4, -0.3, -0.1, 0.9, -0.7, 0.2]$. The last one is converted to the schedule $[2, 1, 1, 3, 2, 2, 1, 3]$, which is a permutation of the set of operations that represents a tentative ordering to schedule them, each one being represented by its ID number. This valid schedule corresponds to the operation sequence $O_{21}, O_{11}, O_{12}, O_{31}, O_{22}, O_{23}, O_{13}$, and O_{32}.

In order to evaluate x_i^g, the objective value is the makespan or C_{max} value. To compute it, each operation O_{ij} in x_i^g is assigned to a feasible machine M_k in U_{ij} with the shortest completion time, and then the load of M_k must be updated. The initial solution is generated by a random procedure (Equation 1), mainly because high performing construction heuristics for FJSSP are unknown.

4.2 DE and Local Search Method

DE is enhanced with a simple local search technique to improve the exploitation of promising regions of the search space. This new algorithm is called DE$_{LS}$. In this work, the local search method is a simple swap mechanism, which randomly selects two positions of the target vector and interchange its values. After that, a greedy selection takes place. If the modified solution presents an improvement in its C_{max} value, then the swap is accepted, otherwise, it is discarded. The frequency of the local search is controlled by the probability p_{LS}. The pseudo-code of the local search procedure is given in Algorithm 2.

This local search procedure is applied to the target vectors x_i of the next population (just before Line 11

Algorithm 2 Local Search Procedure

```
1: for each x_i^g from P^g do
2:     if random() < p_LS then
3:         j , k ← random(1,D)
4:         u_i ← swap(x_i, j,k)
5:         if f(u_i) ≤ f(x_i) then          ▷ for a minimization problem
6:             x_i ← u_i
7:         end if
8:     end if
9: end for
```

Table 2: Parameter Values

Parameter	Values
NP	50
F	0.5, 0,7, and 0,9
Cr	0.1 and 0.9
p_{LS}	0.5, 0,7, and 0,9

of Algorithm 1) but not to the trial vector u_i, which is beneficial to avoid both cycling search and getting trapped in a local optimum. Another important feature of this local search procedure is that it does not need a backward conversion because it is applied over the real-valued vector.

4.3 DE and Parallelism

In terms of designing parallel metaheuristics, the DE can be parallelized in different ways [3]. In this work, the aim of the parallelization is not to change the behavior of the metaheuristic but to speed up the search. For that purpose, we focus on the parallelization of each iteration of the DE [24]. The population is decomposed and handled in parallel, using the well-known global parallelization model. The main process performs the selection operation, which is generally sequential. The rest processes (workers) perform the mutation, the recombination, and the evaluation of the solutions in parallel. Consequently, this model maintains the sequence of the original algorithm, and hence the behavior of the metaheuristic is not altered.

The local search method follows the same parallelization approach as previously mentioned. The solutions are assigned to different partitions, which are improved in parallel.

5 Experimental Design

In this section, we describe the experimental design followed in this approach. We have selected a wide range of FJSSP instances used in the literature taking into account their complexity, which is given by the number of jobs and machines, and the wide variation of flexibility in the number of available machines per operation. In this sense, we considered the data set proposed by Brandimarte [25] as a representative one. The number of jobs ranges from 10 to 20, the number of machines belongs to the set {4,15} and the number of operations for each job varies from 5 to 15. Consequently, the total number of operations ranges from 55 to 240. The flexibility is between 1.43 and 4.10.

Concerning the methodology followed to analyze the results, first, we studied the behavior of these algorithms with different F and Cr values, considering the best C_{max} found and the number of iterations to reach them for each instance. This analysis allows us to determine the best values for the control parameters. Secondly, we determined the impact of incorporating

a local search procedure at different rates of p_{LS}. For this purpose, we take into account the best C_{max} found, and the number of iterations to reach the best solution for each instance. Finally, we study DE's behaviour including parallelism regarding the execution time of each approach. The analyses are principally validated by the data in the tables and figures shown in the experimental research section.

The parametric configuration considered for DE's experimentation is the following. The population size N_P is set to 50. The F factor and the Cr probability are tested with three different values (0.1, 0.5, and 0.9). For the remaining parameter, p_{LS}, three values are also analysed (0.1, 0.5 and 0.7). Table 2 shows a summary of the parameters.

Because of the stochastic nature of the algorithms, we performed 30 independent runs of each test to gather meaningful experimental data and apply statistical confidence metrics to validate our conclusions. Before performing the statistical tests, we first checked whether the data followed a normal distribution by applying the Shapiro-Wilks test. Where the data was distributed normally, we later applied an ANOVA test. Otherwise, we used the Kruskal-Wallis (KW) test to assess whether or not there were meaningful differences between the compared algorithms with a confidence level of 99%.

The considered algorithms were programmed in C++. Consequently, their runtimes are directly comparable. All algorithms were compiled on the same computer with the same compilation flags, and run on homogeneous hardware. All are positive attributes of a comparison. The experimentation is carried out on a cluster of 4 INTEL I7 3770K quad-core processors, 8 GB RAM, and the Slackware Linux with a 2.6.27 kernel version. To implement the parallel version of DE, a portable programming interface for shared memory parallel computers such us OpenMP [26] is used.

6 Experimental Results

In this subsection, we analyze the result quality considering the C_{max} values obtained from the DE algorithm to solve FJSSP, described in previous sections. First, the influence of Cr and F parameters in the DE performance to solve FJSSP is studied (Section 6.1). Next, the incorporation of the local search process to the DE using different p_{LS} values is analyzed in Section 6.3. Finally, the parallel implementation of DE_{LS} is considered in Section 6.3.

Table 3: Best values of C_{max} found by the DE algorithm with different F and Cr values for all FJSSP instances

Inst.	Opt.	F=0.1			F=0.5			F=0.9		
		Cr=0.1	Cr=0.5	Cr=0.9	Cr=0.1	Cr=0.5	Cr=0.9	Cr=0.1	Cr=0.5	Cr=0.9
Mk01	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
Mk02	26	**26**	27	27	**26**	27	**26**	27	27	27
Mk03	204	**204**	**204**	**204**	**204**	**204**	**204**	**204**	**204**	**204**
Mk04	60	**60**	**60**	65	61	**60**	**60**	62	63	62
Mk05	172	175	173	173	175	179	173	175	179	173
Mk06	58	65	59	63	66	69	59	67	70	61
Mk07	139	143	140	142	143	148	140	144	146	140
Mk08	523	**523**	**523**	**523**	**523**	**523**	**523**	**523**	**523**	**523**
Mk09	307	318	**307**	310	321	333	**307**	321	338	**307**
Mk10	197	237	210	221	238	245	206	240	246	215
		5/10	5/10	3/10	4/10	4/10	**6/10**	3/10	3/10	4/10

6.1 Influence of DE Parameters

The first analysis focuses on the effect of using different combinations of F and Cr values on the performance of DE to solve FJSSP. In particular, both parameters take values in $\{0.1, 0.5, 0.9\}$, from low to high values. For this purpose, we analyze the result quality taking into account the best C_{max} values obtained for the simple DE algorithm when solving FJSSP instances, and also the number of evaluations needed to reach their best values.

Table 3 shows the best C_{max} values obtained for the DE algorithm using the different combinations of F and Cr values for each FJSSP instance. Column 2 presents the best-known value of C_{max} (optimal) for each instance. The last row displays the ratio of the number of instances that an algorithm can find its optimum to the total number of instances.

The DE finds the optimum in 3 instances (MK01, MK03, and MK08) independently from the combinations of F and Cr values considered. The highest value of F offers the less chance of finding the best solutions for the DE regardless of the Cr value used. This combination presents the lowest number of optimal values for the different instances (no more than 4 of the 10 instances). The DE algorithm with the combination of $F = 0.5$ and $Cr = 0.9$ finds more times the optimal C_{max} value than the others (in 6 of the 10 instances).

An important metric is the relative error or gap of the different best C_{max} values obtained for each DE regarding the optimum of each instance. This metric allows us to normalized the data of the different instances and, in this way, we focus the attention on how different combinations of F and Cr values affect the performance of the DE to solve FJSSP. On the one hand, KW statistical test indicates that there are significant statistical differences between the DE combinations (p-value$= 2.2e - 16$ is lower to the significance level $\alpha = 0.01$). On the other hand, the gap distribution for each combination (see Figure 2) suggests that the DE algorithm with $F = 0.5$ and $Cr = 0.9$ presents the lowest median relative error, and also, its box is
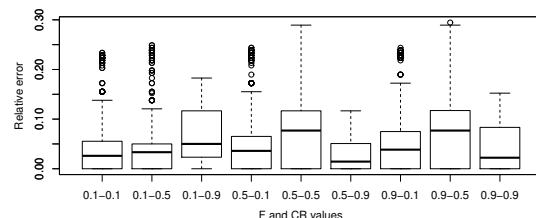

Figure 2: Relative error of the DE algorithm for different combinations of F and Cr values.

comparatively short suggesting that overall gap values have a high level of similarity with each other. These observations indicate a superiority of the combination over the other possible ones.

Finally, we proceed to study the distribution of the number of evaluations to find the best C_{max} value performed by DE with different combinations of different F and Cr values. For that purpose, Figure 3 illustrates these results employing ten box plot graphs (one for each instance). In general, we observe that the DE algorithm with fewer evaluations is the one with $F \in \{0.1, 0.5\}$ and $Cr = 0.9$, but the combination $F = 0.5$ and $Cr = 0.9$ outperforms the other ones from the result quality point of view (see Table 3). Consequently, we will adopt $F = 0.5$ and $Cr = 0.9$ as the parameter settings for the DE in the following experimentation.

6.2 Results of the DE with Local Search

The following analysis goes into detail of what happened when the DE algorithm is enhanced with a local search procedure to solve FJSSP. The resulting algorithm is named DE$_{LS}$. For this study, we consider three different p_{LS} values in the set $\{0.1, 0.5, 0.7\}$, i.e. we study how the frequency of the LS impacts the DE performance by using low, medium, and high probability values.
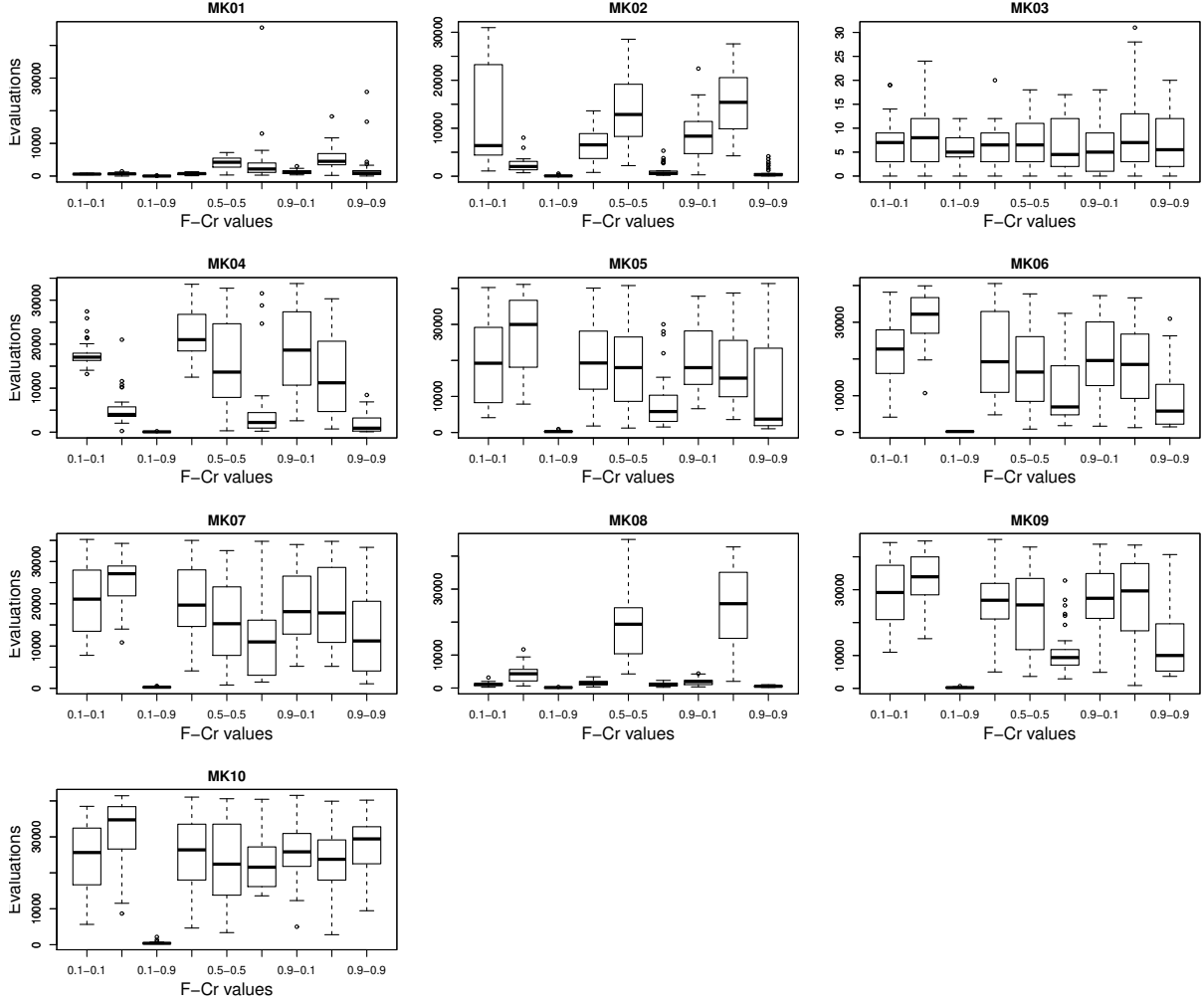
Figure 3: Number of evaluations to find the best C_{max} value of the DE for different combinations of F and Cr values considering all FJSSP instances.

Table 4: Best and mean$_{\pm sd}$ C_{max} values found by the DE$_{LS}$ with different values of p_{LS} for all FJSSP instances.

Inst.	Opt	Best C_{max}			Mean$_{\pm sd}$ C_{max}		
		$p_{LS} = 0.1$	$p_{LS} = 0.5$	$p_{LS} = 0.7$	$p_{LS} = 0.1$	$p_{LS} = 0.5$	$p_{LS} = 0.7$
Mk01	40	**40**	**40**	**40**	$40.00_{\pm 0.00}$	$40.00_{\pm 0.00}$	$40.00_{\pm 0.00}$
Mk02	26	27	**26**	**26**	$27.03_{\pm 0.18}$	$26.96_{\pm 0.18}$	$26.76_{\pm 0.43}$
Mk03	204	**204**	**204**	**204**	$204.00_{\pm 0.00}$	$204.00_{\pm 0.00}$	$204.00_{\pm 0.00}$
Mk04	60	**60**	**60**	**60**	$62.00_{\pm 1.31}$	$61.30_{\pm 0.70}$	$61.03_{\pm 0.66}$
Mk05	172	173	173	173	$174.30_{\pm 0.75}$	$173.00_{\pm 0.00}$	$173.00_{\pm 0.00}$
Mk06	58	63	62	61	$64.40_{\pm 0.56}$	$62.83_{\pm 0.53}$	$62.43_{\pm 0.62}$
Mk07	139	142	140	140	$143.20_{\pm 0.69}$	$142.00_{\pm 0.90}$	$141.80_{\pm 0.92}$
Mk08	523	**523**	**523**	**523**	$523.00_{\pm 0.00}$	$523.00_{\pm 0.00}$	$523.00_{\pm 0.00}$
Mk09	307	309	**307**	**307**	$313.10_{\pm 2.20}$	$310.00_{\pm 1.33}$	$309.00_{\pm 1.72}$
Mk10	197	226	225	224	$231.10_{\pm 2.41}$	$228.00_{\pm 1.36}$	$227.20_{\pm 1.38}$
		4/10	6/10	6/10			

Table 4 shows the best and the mean C_{max} values, together the standard deviation, found by the DE$_{LS}$ with the different p_{LS} values. The DE$_{LS}$ obtains more times the optimum when $p_{LS} = 0.5$ or $p_{LS} = 0.7$. Moreover, the KW test indicates that there are statistically significant differences among the algorithms (p-values

are lower than the level of significance). The DE$_{LS}$ using the highest frequency ($p_{LS} = 0.7$) has the lowest mean C_{max} values for all instances, suggesting that the algorithm can find the optimum or near-optimal C_{max} values in the majority of the runs. This assumption is reflected in the boxplot of the relative error presented
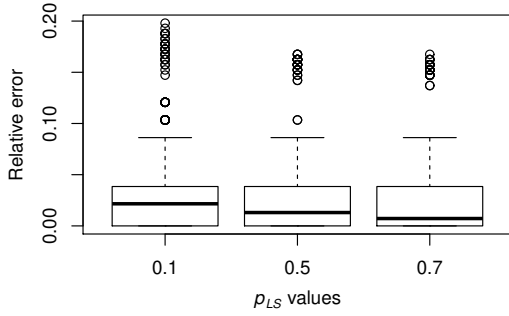
Figure 4: Relative error of the DE_{LS} for different values of p_{LS}.
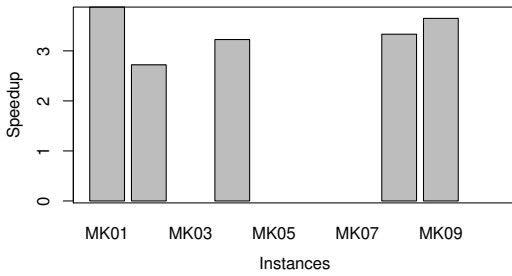

Figure 5: Speedup by FJSSP instances.

in Figure 4, which shows that the median error value is lower with the highest local search frequency. Now, to determine if the DE_{LS} can improve the C_{max} values found by the DE, we perform a comparison of relative error values shown in Figure 2 and the ones from Figure 4. We can observe that the DE_{LS} algorithm with $p_{LS} = 0.7$ obtains lower relative errors than those presented by DE, which indicates the advantage of incorporating local search into the DE framework.

6.3 Results of the DE_{LS} and Parallelism

In this section, we study the behavior of the parallel DE described in Section 4.3. The most important measure of a parallel algorithm is speedup. This measure is defined as the ratio of the sequential execution time (DE_{LS} execution time, in this case) to the parallel execution time. For this analysis, we consider the weak speedup [27]. For that reason and following the best practice by Luque and Alba [4], the stopping criterion is based on the quality of the final solution achieved by the algorithms, which is set to the optimum for each FJSSP instance (see column Opt. of Table 3). Consequently, the speedup values are only reported for those instances for which the DE_{LS} algorithm obtains the optimum value.

Once we established the execution times of the DE_{LS} and the parallel DE_{LS}, we calculate the speedup values. Figure 5 shows that the use of parallelization is worthwhile, as we expected. The parallel DE_{LS} allows to reduce the search time and obtains a very good speed up, nearby linear (the ideal speedup value is 4, the number of available cores per machine).

7 Comparison of DE_{LS} with the Literature

Finally, we present a comparative assessment of the C_{max} values obtained by the DE_{LS} with the ones of several competitive algorithms present in the literature to solve FJSSP. This allows us to determine the goodness of the metaheuristics considered in this work. In this comparison different population-based metaheuristics to solve FJSSP are considered:

i) hGA [5]: a hybrid algorithm combining chaos particle swarm optimization with genetic algorithm

ii) BEDA [6]: a bi-population based estimation of distribution algorithm

iii) IACO [7]: an ant colony optimization

iv) HDE [13]: a hybrid differential evolutionary algorithm

Table 5 shows that the C_{max} values obtained by DE_{LS} are similar to the ones of remaining algorithms, for the majority of the ten instances. This observation suggests that the DE_{LS} proposed in this work is a competitive algorithm to solve FJSSP. Comparisons regarding computational effort are hard to be carried out because the majority of the works do not report the number of evaluations. Consequently, the relative efficiency of referred algorithms is difficult to contrast to obtain meaningful comparisons.

8 Conclusions

This paper presented a simple DE algorithm to solve FJSSP, which has significant application value in modern manufacturing environments. In this study, we considered a real-valued representation to code a valid schedule for FJSSP. Consequently, we maintained the good properties of the DE as a problem optimizer because it still works on the continuous domain. The DE was enhanced with a very simple local search procedure (DE_{LS}) to balance the exploration and exploitation of the search space. Furthermore, each iteration of the DE_{LS} was parallelized to speed up the computation using the global parallelization model.

Computational results indicated that the DE with a combination of medium F and high Cr values obtained good quality of C_{max} values in a less number of

Table 5: Comparison between DE$_{LS}$ and population-based metaheuristics from the literature

	MK01	MK02	MK03	MK04	MK05	MK06	MK07	MK08	MK09	MK10
DE$_{LS}$	**40**	**26**	**204**	**60**	173	61	140	**523**	**307**	224
hGA	**40**	**26**	**204**	62	**172**	65	140	**523**	310	214
BEDA	**40**	**26**	**204**	**60**	172	60	**139**	**523**	**307**	206
IACO	**40**	**26**	**204**	**60**	173	60	140	**523**	**307**	208
HDE	**40**	**26**	**204**	**60**	172	57	**139**	**523**	**307**	**198**

evaluations. Moreover, the DE$_{LS}$ with a high probability of application of the local search procedure was able to improve the best solutions found by the DE for FJSSP in a major number of instances. We also obtained a decrease in execution times by introducing parallelization to the DE framework without changing the algorithm behavior. Finally, the comparison of our proposal with several algorithms in the literature indicated that we develop a competitive approach. As a consequence, our simple DE proposal is an efficient and competitive optimizer for this NP-hard problem.

As future research activities, we will plan to extend the study by including another set of instances with high dimensionality. Furthermore, FJSSP variants with more constraints will be evaluated considering the approaches developed in this article.

Competing interests

The authors have declared that no competing interests exist.

Funding

Authors' contribution

All authors conceived the idea, wrote the program, conducted the experiments, analyzed the results, and finally, wrote and revised the manuscript. Moreover, all authors read and approved the final manuscript.

References

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Springer Publishing Company, Incorporated, 3rd ed., 2008.

[2] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, pp. 117–129, May 1976.

[3] E.-G. Talbi, *Metaheuristics: From Design to Implementation.* Wiley, 2009.

[4] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications.* Springer Publishing Company, Incorporated, 2013.

[5] J. Tang, G. Zhang, B. Lin, and B. Zhang, "A hybrid algorithm for flexible job-shop scheduling problem," *Procedia Engineering*, vol. 15, pp. 3678 – 3683, 2011.

[6] L. Wang, S. Wang, Y. Xu, G. Zhou, and M. Liu, "A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem," *Computers & Industrial Engineering*, vol. 62, no. 4, pp. 917 – 926, 2012.

[7] L. Wang, J. Cai, M. Li, and Z. Liu, "Flexible job shop scheduling problem using an improved ant colony optimization," *Scientific Programming*, pp. 1–11, 2017.

[8] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[9] B. Teoh, S. Ponnambalam, and G. Kanagaraj, "Differential evolution algorithm with local search for capacitated vehicle routing problem," *Int. J. Bio-Inspired Comput.*, vol. 7, no. 5, pp. 321–342, 2015.

[10] R. Greco and I. Vanzi, "New few parameters differential evolution algorithm with application to structural identification," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 6, no. 1, pp. 1 – 14, 2019.

[11] P. Hull, M. Tinker, and G. Dozier, "Evolutionary optimization of a geometrically refined truss," *Structural and Multidisciplinary Opt.*, vol. 31.

[12] R. M. K. Rout, "Simultaneous selection of optimal parameters and tolerance of manipulator using evolutionary optimization techniques," *Structural and Multidisciplinary Optimization*, vol. 40, no. 1-6, pp. 513–528, 2010.

[13] Y. Yuan and H. Xu, "Flexible job shop scheduling using hybrid differential evolution algorithms," *Computers & Industrial Eng.*, vol. 65, no. 2, pp. 246–260, 2013.

[14] F. Morero, C. Bermudez, and C. Salto, "A simple differential evolution algorithm to solve the flexible job shop scheduling problem," *XXV Congreso Argentino de Ciencias de la Computación*, pp. 1–10, 2019.

[15] T. Eltaeib and A. Mahmood, "Differential evolution: A survey and analysis," *Applied Sciences*, vol. 8, no. 10, 2018.

[16] I.Kacem, S.Hammadi, and P. Borne, "Approach by localization and multi objective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Trans. Syst. Man Cybern. C, Appl.*, vol. 1, pp. 1–, 2002.

[17] R. Storn and K. Price, "Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces," tech. rep., ICSI,USA,Tech. Rep. TR-95-012,, 1995.

[18] S. Das, S. S. Mullick, and P. Suganthan, "Recent advances in differential evolution – an updated survey," *Swarm and Evolutionary Computation*, vol. 27, pp. 1 – 30, 2016.

[19] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series).* Berlin, Heidelberg: Springer-Verlag, 2005.

[20] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, Massachusetts: The MIT Press, first ed., 1975.

[21] R. Gämperle, S. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," in *WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pp. 293–298, Press, 2002.

[22] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[23] C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *Operations-Research-Spektrum*, vol. 17, pp. 87–92, 1995.

[24] N. Nedjah, E. Alba, and L. de Macedo Mourelle, *Parallel Evolutionary Computations*. Springer-Verla, 2006.

[25] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, pp. 157–183, 1993.

[26] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007.

[27] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.