A Simple Differential Evolution Algorithm to Solve the Flexible Job Shop Scheduling Problem

Franco Morero, Carlos Bermudez, and Carolina Salto

Facultad de Ingeniería, UNLPam
General Pico, La Pampa, Argentina, CONICET

Abstract. This paper addresses the Flexible Job Shop Scheduling Problem (FJSSP) where the objective is to minimize the makespan. We develop a parallel hybrid Differential Evolution (DE) algorithm to tackle this problem. A random key representation of the FJSSP is adopted, which requires a very simple conversion mechanism to obtain a feasible schedule. This allows the DE algorithm to work on the continuous domain to explore the problem space of the discrete FJSSP. Moreover, a simple local search algorithm is embedded in the DE framework to balance the exploration and exploitation by enhancing the local searching ability. In addition, parallelism of the DE operations is included to improve the efficiency of whole algorithm. Experiments confirm the significant improvement achieved by integrating the propositions introduced in this study. Additional, test results show that our algorithm is competitive when compared with most existing approaches for the FJSSP.

Keywords: flexible job shop scheduling, differential evolution algorithm, parallelism

1 Introduction

A realistic production environment, and with practical applicability, is the Flexible Job Shop Scheduling Problem. Each job has to undergo multiple operations on the various machines. The decision concerns how to sequence the operations on the machines, so that the time needed to complete all the jobs (C_{max}) is minimized. Moreover, an additional decision consists in to assign each operation to the appropriate set of machines. These decisions suggest that the FJSSP is a complex optimization problem (NP-hard problem [1]), consequently, the adoption of metaheuristic [2, 3] has led to better results than classical dispatching or greedy heuristic algorithms [4–6]. Since introduced in 1997 by Storn and Price [7], the Differential Evolution (DE) metaheurisitc became very popular among computer scientists and practitioners almost immediately after its original definition.

DE is a stochastic real-parameter global optimizer. It employs simple mutation and crossover operators to generate new candidate solutions, and applies one-to-one competition scheme to greedily determine whether the new candidate or its parent will survive in the next generation. Their successful is due to its simplicity and ease implementation, and reliability and high performance. DE algorithms have been applied to many combinatorial optimization problems

([8–11], 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 the FJSSP [12].

In this work, a simple DE to solve the FJSSP is design. As DE was originally devised for solving continuous optimization problems, we adopt a real value representation for the FJSSP to make the continuous DE applicable for solving the discrete FJSSP, which implies that algorithm operations should not be modified or adapted to resolve the problem. Another important feature of DE is the little number of parameters to be set, when compared to other evolutionary algorithms. However, the success to find good solutions to a problem depends on discovering the correct values of those parameters [13]. Therefore, we make an analysis in this line to determine the adequate values for these parameters for solving the instances of the FJSSP. Moreover, a simple local search procedure is embedded to the DE to improve their exploration capacities by solving the problem. Finally, parallelism at algorithmic level [2] is incorporated to the design of the DE, with the aim of improving the scalability and reducing the computation time. The experimental methodology we have followed consists of computing the C_{max} values for the different DE proposed to solve the FJSSP and then comparing the obtained results by considering different quality indicators. We find that our simple DE shows a promising behavior to solve the FJSSP.

The paper is organized as follows. In Section 2, we introduce the problem formulation . In Section 3, we present the basic DE algorithm. In Section 4 we explain the adaptations of the DE to solve the FJSSP. In the following section, we introduce the experimental design and in Section 6, we evaluate the results. Some final remarks and future research directions are given in Section 7.

2 The Flexible Job Shop Scheduling Problem

The FJSSP can be formally described as follows. A set $J = \{J_1, J_2, ..., J_n\}$ of independent jobs and a set $U = \{M_1, M_2, ..., M_m\}$ of machines are given. A job J_i is broken down by a sequence of $O_{i1}, O_{i2}, ..., O_{in_i}$ 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. We have partial flexibility if there 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. We denote with d_{ijk} the processing time of operation O_{ij} when executed on machine M_k. 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 to sequence the operations on the machines (sequencing problem) in order 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 job J_i. Table 1 shows an instance of the FJSSP with 3 jobs, 4 machines and 8 operations. The rows and columns correspond to machines and operations, respectively, and the entries of the table are the processing times.

Table 1. Instance Example for the 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

Algorithm 1 Differential Evolution Algorithm (DE)

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: $x_i^g \leftarrow$ 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)

3 Differential Evolution Algorithm: Background

The DE algorithm was proposed by Storn and Price [7] to solve optimization problems with real-valued parameters. DE is a stochastic, population-based optimization method. Despite having a very simple algorithmic structure, DE has demonstrated a high level of performance when solving a wide variety of very complex problems [14]. The optimal or near-optimal solution is obtained by an iterative process which is applied to a set of solutions (population) to achieve a new one. At each step of the process, new solutions arise as a result of perturbations to the current solutions, caused by mutation and recombination operators.

The algorithmic framework of DE is described in Algorithm 1. The first step (Line 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 component $x_{i,j} \in \mathbb{R} (1 \leq j \leq D)$ represents a variable or a parameter of the optimization problem. Usually, each $x_{i,j}$ is bounded to a value in the range $[li_j, ls_j]$, where li_j, $ls_j \in \mathbb{R}$ are the lower and upper bound, respectively. The N_P target vectors are initialized randomly by applying Equation 1:

$$x_{i,j} = li_j + U(0,1) \times (ls_j - li_j) \quad (1)$$

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

After the initialization step, an iterative process begins. The mutation operation (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 $(0 \leq g \leq max_{gen})$ 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. 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.

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

The donor vector is modified by the recombination operator (Line 6), with the aim of increasing 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:

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

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.

The last step 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. Clearly, this competition creates a new population with a performance equal or superior to the current one (Line 8). Consequently, DE is an elitist evolutionary algorithm.

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

DE performance mainly depends on three parameters: scaling factor of the difference vector (F), crossover control parameter (Cr) and population size (N_P). Some guidelines are available to choose the control parameters [14]. In this work, N_P and F are chosen based on previous knowledge and keep it constant during all runs. On the other hand, a good value for Cr is 0.1 however, to speed up convergence a greater value can be used.

4 Our proposal: Hybrid DE for the FJSSP

In this section, our proposal to solve the FJSSP is detailed. In order to apply the DE algorithm, it is crucial to design a suitable encoding scheme that maps the floating point vectors to the feasible solution for the 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 (Section 4.3).

4.1 Representation

In this work, the DE algorithm still manipulates real-valued vector in order to maintain the simplicity and properties of the DE in their natural configuration. Consequently, the schedule is generated following the random keys encoding scheme [15]. This representation deals with real point vectors, where these

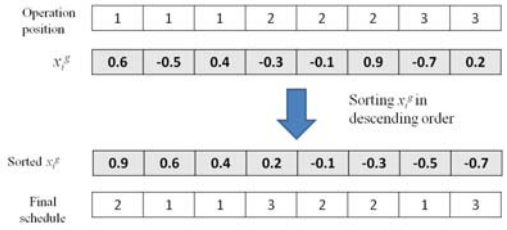

Fig. 1. Example of the decoding process used by the DE to solve the FJSSP.

points are used as sort keys to decode the solution. For an n-job m-machine scheduling problem, each vector's position (a random key) consists of a floating number in U(-1,1) which can be translated to an unique list of ordered operations, after a descending order of the random keys. These steps always obtain a feasible schedule from a real-valued vector. The schedule is a permutation with repetitions [16]. See Figure 1 for an example considering the instance shown in Table 1. Given the vector x_i^g=[0.6,-0.5,0.4,-0.3,-0.1,0.9,-0.7,0.2], it is converted to a 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 job 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 (C_{max}). 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 the FJSSP are unknown.

4.2 DE and Local Search Method

DE is enhanced with a local search technique, yielding a hybrid DE (HDE) for exploration and exploitation among the solutions to obtain a near optimal solution. In this work, a simple interchange mechanism is implemented in which two positions of the target vector are randomly selected and interchanged. If there is an improvement in the objective function the swap is accepted, otherwise, it is not considered. This local search procedure is applied to the target vectors x_i of the next population (just before Line 11 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. Moreover, the frequency of the local search is controlled by the probability p_{LS}. Another important characteristic 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 paralleled in different ways [2]. In this work, the aim of the parallelization is not to change the behaviour of the metaheuristic but to speed up the search. For that purposes,

we focus on the parallelization of each iteration of the DE [17]. The population of individuals is decomposed and handled in parallel, using the well-known global parallelization model. A principal process performs the selection operations and the replacement, which are generally sequential. The rest processes (workers) perform the mutation, 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.

5 Experimental Design

In this section, we describe the experimental design used 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 [18] as a representative one, since 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 ranges from 5 to 15, consequently the total number of operations ranges from 55 to 240. The flexibility varies between 1.43 and 4.10.

The parametric configuration considered for the DE's experimentation is the following. The population size, N_P is set to 50. The F factor is equal to 0.9. These values were adopted from previous works. Regarding Cr probability, three different values were considered (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).

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. As the data do not follow a normal distribution, 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 experimentation is carried out on a cluster of 4 INTEL I7 3770K quad-core processors, 8 GB RAM, and the Slackware Linux with 2.6.27 kernel version. To implement the parallel version of DE, a portable programming interface for shared memory parallel computers such us OpenMP [19] is used.

6 Experimental Results

The first analysis is focused on the effect of using different Cr values in the DE performance, from low to high values (0.1, 0.5 and 0.9). Table 2 shows the best C_{max} values obtained for the DE algorithm using the different Cr values for each instance (columns 3 to 5). Also, the mean C_{max} values together with the mean standard deviation (sd) are presented (columns 6 to 8). Column 2 displays the best known C_{max} value for each instance. Last column shows the results of the KW test, where the symbol "+" indicates significant differences between the algorithms (p-value is inferior to the significance level$\alpha = 0.01$).

Table 2. C_{max} values found by the DE algorithm with different Cr values.

Instance	Opt.	Best C_{max} Values			Mean C_{max} Values $\pm$ sd			KW
		Cr=0.1	Cr=0.5	Cr=0.9	Cr=0.1	Cr=0.5	Cr=0.91	
Mk01	**40**	**40**	**40**	**40**	40.00 $\pm$0.00	40.00 $\pm$0.00	40.73 $\pm$0.78	-
Mk02	26	27	27	27	27.00 $\pm$0.00	27.20 $\pm$0.42	27.33 $\pm$0.47	+
Mk03	204	**204**	**204**	**204**	204.00 $\pm$0.00	204.00 $\pm$0.00	204.00 $\pm$0.00	-
Mk04	60	**60**	61	**60**	61.70 $\pm$0.60	65.20 $\pm$2.53	64.80 $\pm$2.19	+
Mk05	172	174	177	173	175.87 $\pm$0.73	181.10 $\pm$1.79	174.03 $\pm$1.82	+
Mk06	58	66	70	60	67.17 $\pm$0.59	71.10 $\pm$0.74	63.15 $\pm$1.12	+
Mk07	139	144	149	140	144.07 $\pm$0.25	150.10 $\pm$0.99	142.18 $\pm$1.30	+
Mk08	523	**523**	**523**	**523**	523.00 $\pm$0.00	523.30 $\pm$0.95	523.00 $\pm$0.00	-
Mk09	307	321	338	**307**	325.13 $\pm$1.48	343.00 $\pm$2.40	310.05 $\pm$2.48	+
Mk10	197	237	249	213	241.03 $\pm$1.25	252.00 $\pm$1.25	218.00 $\pm$2.71	+

The DE algorithm with Cr=0.9 finds lowest C_{max} values than the rest. Moreover, this configuration reaches the best known C_{max} values in 5 of the 10 instances. Now, analyzing what is happening with the mean C_{max} values, the DE using Cr=0.1 presents the lowest values for the four first instances (MK01-04). Furthermore, sd values are equal to zero for these instances, indicating that the algorithm is able to find the optimum value for all the runs. In the remaining instances (MK05-10), the DE with Cr=0.9 presents the lowest C_{max} values. Finally, the DE using Cr=0.5 shows the worst performance. Given that the p-values of KW test is lower than the level of significance considered, we can state that there are significant differences among the DE with Cr values (except for instances MK01, MK03, and MK08).

From previous analysis, we can conclude that for instances with relatively few operations, the fraction of parameter values that are copied from the target vector in the recombination operation should be small to allow the algorithm to find the best solutions to the problem. On the other hand, when the complexity of the instances grows up, it is necessary to increase the amount of disturbances in the solutions, and in this way the algorithm could converge to near-optimal solutions. Consequently, we will adopt two different values of Cr for the remaining experimentation: $Cr = 0.1$ for MK01-04 instances and $Cr = 0.9$ for the remaining ones.

Following analysis goes into detail of what happened with the introduction of a local search procedure to the DE algorithm (HDE) to solve the FJSSP. For that purpose, we considered three different p_{LS} values: 0.1, 0.5 and 0.7 (low, medium, and high probability values), i.e. we study how the frequency of the LS application impacts on the DE performance.

Table 3 shows the best and mean C_{max} values obtained for the HDE with the different p_{LS} values. The HDE algorithm applying the local search procedure with high frequency (p_{LS}=0.7) obtains low best C_{max} values for the majority of the FJSSP instances. Moreover, this algorithm exhibits the lowest mean C_{max} values for all instances. These indicate that the algorithm is able to find the optimum or the near-optimum values in the majority of the runs. Moreover, the KW test indicates that there are statistical significant differences among the algorithms (p-values are lower than the level of significance). Comparing C_{max}

Table 3. C_{max} values found by the HDE algorithm with different p_{LS} values.

Inst	Opt.	Best C_{max}			Mean C_{max} values $\pm$ sd			KW
		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**	27.00 $\pm$0.00	26.80 $\pm$0.41	26.77 $\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**	61.20 $\pm$0.66	60.33 $\pm$0.48	60.17 $\pm$0.38	+
MK05	172	173	173	173	1.73 $\pm$0.18	173.00 $\pm$0.00	173.00 $\pm$0.00	+
MK06	58	62	61	60	63.27 $\pm$0.69	62.13 $\pm$0.73	61.90 $\pm$0.48	+
MK07	139	140	**139**	140	142.37 $\pm$0.96	140.83 $\pm$0.79	140.63 $\pm$0.61	+
MK08	523	**523**	**523**	**523**	523.00 $\pm$0.00	523.00 $\pm$0.00	523.00 $\pm$0.00	-
MK09	307	**307**	**307**	**307**	309.93 $\pm$1.53	307.67 $\pm$1.06	307.37 $\pm$0.89	+
MK10	197	225	223	219	228.20 $\pm$1.97	225.77 $\pm$1.36	224.73 $\pm$1.66	+

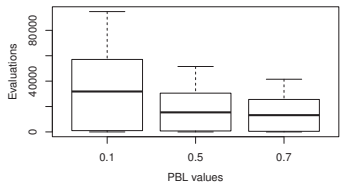

Fig. 2. Total number of evaluations for the HDE .

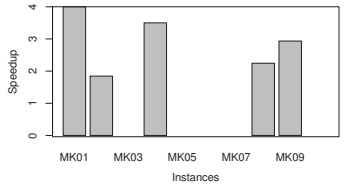

Fig. 3. Speedup per FJSSP instances.

values from Table 2 and the ones from Table 3, we can observe that the HDE algorithms obtain best

Figure 2 illustrates the distribution of the number of evaluations to find the best C_{max} values for the HDE algorithms with different P_{LS} values. We observe that the HDE algorithm with P_{LS}=0.7 needs less number of evaluations than the rest of the algorithms. If we also consider the C_{max} values obtained by each HDE, the one with P_{LS}=0.7 is the best approach to solve this problem.

Following analysis is devoted to compare the HDE and its parallel version as described in Section 4.3. The most important measure of a parallel algorithm is the speedup. The speedup is defined as the ratio of the sequential execution time (HDE execution time, in this case) to the parallel execution time. For this analysis, we consider the weak speedup [20]. For that reason and following the best practice by Luque and Alba [3], the stopping criterion is based on the quality of the final solution achieved by the algorithms, which is set to the best known C_{max} for each FJSSP instance (see column Opt of Table 2). Consequently, the speedup values is only reported for the instances for which the HDE algorithm obtains the optimum value.

Figure 3 shows that the use of parallelization is worth while, which allow us to speed up the execution time with respect to the sequential HDE near 3 times in average (the ideal speedup value is 4, the number of available cores per machine).

Finally, to determine the goodness of the metaheuristics considered in this work, we present a comparison of the results from the HDE with several competitive algorithms present in the literature. This allows a comparative assessment

Table 4. Comparison between HDE and population-based Metaheuristics from the literature

	MK01	MK02	MK03	MK04	MK05	MK06	MK07	MK08	MK09	MK10
HDE	**40**	**26**	**204**	**60**	173	60	140	**523**	**307**	219
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**

of the algorithms for the FJSSP. In this comparison different population-based metaheuristics to solve the FJSSP are considered: i) a hybrid algorithm combining chaos particle swarm optimization with genetic algorithm (hGA) [4], ii) a bi-population based estimation of distribution algorithm (BEDA) [5], iii) an ant colony optimization (IACO) [6], and finally, iv) a hybrid differential evolutionary algorithm [12]. From the comparison, the C_{max} values of HDE are similar with the values of remaining algorithms, for the majority of the ten instances (a comparative table is no included due to lack of space). This observation suggests that the HDE developed in this work is a competitive algorithm to solve the FJSSP. Comparisons regarding computational effort are hard to be carried out because the majority of the works do not report number of evaluations. Consequently, the relative efficiency of referred algorithms are difficult to contrast in order to obtain meaningful comparisons.

7 Conclusions

In this article, we have presented a simple DE algorithm to solve the FJSSP. In this study, the traditional real-parameter global optimizer is considered to maintain the properties of the DE in their natural configuration. The DE is enhanced with a very simple local search procedure, obtaining a hybrid DE (HDE). Moreover, each iteration of the DE is parallelised to speed up the computation. The results indicate that the HDE with a high probability, at which the local search procedure is applied, is able to find the best solutions for the FJSSP. Moreover, when HDE is contrasted with algorithms in the literature, it also becomes a competitive approach. As a consequence, HDE gives good solutions to this NP-hard problem in an efficient and competitive way.

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

Acknowledgements

This research is is supported by Universidad Nacional de La Pampa, and the Incentive Program from MINCyT. The last author acknowledges CONICET.

References

1. M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.
2. E.-G. Talbi, *Metaheuristics: From Design to Implementation.* Wiley, 2009.
3. G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications.* Springer Publishing Company, Incorporated, 2013.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. P. Hull, M. Tinker, and G. Dozier, "Evolutionary optimization of a geometrically refined truss," *Structural and Multidisciplinary Opt.*, vol. 31.
11. 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.
12. 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.
13. T. Eltaeib, Tarik, and A. Mahmood, "Differential evolution: A survey and analysis," *Applied Sciences*, vol. 8, no. 10, 2018.
14. 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.
15. J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.
16. C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *Operations-Research-Spektrum*, vol. 17, pp. 87–92, 1995.
17. N. Nedjah, E. Alba, and L. de Macedo Mourelle, *Parallel Evolutionary Computations.* Springer-Verla, 2006.
18. P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, pp. 157–183, 1993.
19. B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming.* MIT Press, 2007.
20. E. Alba, *Parallel Metaheuristics: A New Class of Algorithms.* Wiley, 2005.