2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Multi-Objective Flexible Job-Shop Scheduling Problem with DIPSO: More Diversity, Greater Efficiency

Luiz Carlos Felix Carvalho   Márcia Aparecida Fernandes

Abstract— **The Flexible Job Shop Problem is one of the most important NP-hard combinatorial optimization problems. Evolutionary computation has been widely used in research concerning this problem due to its ability for dealing with large search spaces and the possibility to optimize multiple objectives. Particle Swarm Optimization has shown good results, but algorithms based on this technique have premature convergence, therefore some proposals have introduced genetic operators or other local search methods in order to avoid the local minima. Therefore, this paper presents a hybrid and multi-objective algorithm, Particle Swarm Optimization with Diversity (DIPSO), based on Particle Swarm Optimization along with genetic operators and Fast Non-dominated Sorting. Thus, to maintain a high degree of diversity in order to guide the search for a better solution while ensuring convergence, a new crossover operator has been introduced. The efficiency of this operator was tested in relation to the proposed objectives by using typical examples from literature. The results were compared to other studies that have shown good results by means Evolutionary Computation technique, for instance MOEA-GLS, MOGA, PSO + SA and PSO + TS.**

I. Introduction

JOB-SHOP SCHEDULING (JSP) has motivated the interest of a significant number of researchers in different areas over the years since it has appeared and have gained prominence through diverse and important real world applications, for example, manufacturing systems, production planning, computer design and communication [1].

Since JSP is one of the hardest combinatorial problems and considered an important NP-Complete problem [2], many different approaches based on methods such as branch and bound, dynamic programming, integer programming, genetic algorithms and hybrid techniques have been proposed to approximate successful solutions to these problems.

The Flexible Job-Shop Scheduling Problem (FJSP) is an extension of JSP since it allows each operation to be processed on more than one machine. This introduces two additional difficulties to the classical JSP: the assignment of each operation to the appropriate machine and the determination of the operation sequencing (start times) on each machine [3]. Consequently, the FJSP is more powerful than JSP for approximating solutions to real world applications, where it is easy to observe features such as more than one available resource for performing one task. The FJSP is denominated partial, P-FJSP, if operations can be processed by subsets of a machine set, otherwise, it is total, T-FJSP, which means all operations can be processed by all machines.

Luiz Carlos Felix Carvalho and Márcia Aparecida Fernandes belong to the Graduate Program in Computer Science, Federal University of Uberlândia, Minas Gerais, Brazil (email: luizlcfc@gmail.com, marcia@ufu.br).

This article is organized as follows. Section 2 describes problem formulation. In section 3 studies related to the use of Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) in order to propose solutions for the FJSP are presented. Section 4 presents general GA and PSO algorithms. Section 5 details a hybrid and multi-objective algorithm based on GA, PSO and Fast Non-dominated Sorting procedure (FNS) which was developed in this study. Section 6 presents, discusses and compares the results and section 7 presents the conclusion along with further works.

II. FJSP - Flexible Job Shop Problem

The problem consists of n jobs and m machines, where each job is made up by order and not preemptive operations (in this article, only T-FJSP is considered). The goal is the assignment of the job operations to machines according to some performance criteria. Then, in this work, FJSP takes into account the precedence constraints and multi-objective performance. The main variables are given by j: job index, where $1 \leq j \leq n$, k: machine index, where $1 \leq k \leq m$ and i: job operation index, n_j: is the number of job operations j, and $N = \sum_{j=1}^{n} n_j$: is the number of operations.

The problem formulation is given by the minimization of three objectives. The first is the makespan (M), given by Equation 1, total workloads of machines (TW) given by Equation 2, the maximum workloads of machines (W), given by Equation 3, where t_k is final execution time of machine k and W_k is the workload of machine k. It is assumed that all machines are available in time zero, each machine can perform a maximum of one operation at a time and the fixed order of operations of jobs cannot be changed (precedence constraint).

$$M = max_{1 \leq k \leq m} t_k \tag{1}$$

$$TW = \sum_{1 \leq k \leq m} W_k \tag{2}$$

$$W = max_{1 \leq k \leq m} W_k \tag{3}$$

A solution to this problem is a job execution sequence through the observation of one or multiple optimization objectives. In order to cope with the multi-objective aspect, a weight can be established for each objective and the weighted sum of their values is taken as the fitness, as shown in Zhang et al. [4]. However, the individual consideration of each objective is more effective to deal with the multi-objective and can be done by means of the Fast Non-dominated Sorting [5], a procedure that finds the Pareto-optimal front for the classification of diverse candidate solutions.

III. Related Studies

Evolutionary computation techniques have contributed with research concerning optimization problems. Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) are two important examples of this technique, which are widely used in proposals for resolving FJSP. Zhang et al. [4], Jia et al. [6], Ling-li et al. [7] and Xiao-hong et al. [8] have used PSO and Wang et al. [10], Gen et al. [11] and Binh et al. [12] have used GA. Zhang et al. [4], Wang et al. [10] and Binh et al. [12] presented proposals that contribute to the multi-objective aspect of this problem.

It has been observed that PSO algorithms can lead to local minima due to their fast convergence. In [4] and [8], genetic operators such as crossover and mutation avoid local minima in the PSO algorithm. In addition, Zhang et al. [4] have proposed the use of Tabu search, a local search that has shown good results when dealing with scheduling problems and multiple objectives, where fitness is given by the weighted sum of these parameters. In [8], only makespan was considered.

Ling et al. [7] presented a mathematical model using HPSO (Hybrid Particle Swarm Optimization), which is composed of PSO and GA for makespan minimization. However, the individual generation and the crossover need an additional process of individual correction, since spurious individuals may have been created, especially in relation to job operation order. This implies in increasing the algorithm runtime. Ling et al. [7] demonstrated that hybrid algorithms are more efficient than using only one of them through the comparisons between HPSO and a simple PSO.

Binh et al. [12] developed an algorithm based on evolutionary algorithms and local search for resolving multi-objective FJSP. The main aspect of this proposal is the composition of an evolutionary algorithm cycle and a local search cycle, taking into account Pareto-optimal rank. At the end of each iteration, an elite population is saved.

Based on the above examples, it was observed that the composition of techniques has presented better results for FJSP. Thus, this article presents a proposal for resolving the multi-objective FJSP based on PSO algorithm, genetic operators and Pareto-optimal through the use of the Fast Non-dominated Sorting algorithm. The proposal is composed of a PSO algorithm where crossover and mutation operators replace velocity and position equations according to proposal in the original version of PSO. The fitness considers makespan, the total workloads and the maximum workloads of machines.

IV. Theoretical Background

As shown through research, the evolutionary computational techniques have allowed for the exploration of complexity surrounding problems such as FJSP. This section describes the main features of multi-objective GA and PSO algorithms used in the proposed hybrid algorithm.

A. Genetic Algorithms

Among the techniques within evolutionary computation, Genetic Algorithms are the most widely used. In general, GA evolves a randomly generated initial population of individuals, through the use of genetic operators such as, crossover, selection and mutation. The process ends when a certain number of iterations is executed.

The original proposal of GA consisted of the optimization of a single objective and the individuals were represented by bit strings. However, the successful application of these algorithms to different problems has expanded the research towards the treatment of problems with more complex representations and multiple objectives. FJSP is an example of a problem in which these advances allow obtaining significant results. A multi-objective GA was developed by [5] and denominated Non-dominated Sorting Genetic Algorithm (NSGA), where the Pareto-optimal is determined according to Definitions (1), (2) and (3).

Definition 1 (Relation of Dominance): Let S be a solution set, x and y two solutions of S. x dominates y, if and only if, $M_x \leq M_y$, $W_x \leq W_y$, and $TW_x \leq TW_y$, since x is different of y.

Definition 2 (Non-dominated Sets): Ordered Subsets from S set, which contain solutions that are not dominated by other solutions. Therefore, Front 1 ($F1$) is the set of every solution that are not dominated by any solution of S. Front 2 ($F2$) contains the solutions that are not dominated by any solution $SF1$ ($S - F1$), and so forth.

Definition 3 (Pareto-optimal Set): Subset of S, which contains solutions that are not dominated by any other. In other words Front 1.

B. PSO - Particle Swarm Optimization

Particle Swarm Optimization (PSO)[13] raises from artificial life and swarming theories; in addition, it is related to evolutionary algorithms, such as genetic algorithms and evolutionary programming. Particle swarm algorithms are defined by a particle (individual) population, where each particle has position and velocity. The evolutionary algorithm is determined by updates in these two parameters. In each evolution step the new positions and velocities of the particles are calculated taking into account the global and local best positions. Global best position (g_{best}) is related to all particles in the population and local best (p_{best}) is related to the particle itself. These two parameters are very important for the convergence algorithm. If global is greater than local, there is diversity, otherwise premature convergence.

A general PSO formulation can be seen in Martínez et al. [14], where b particles in the initial population have random positions (${x_a}^0$) and velocities (v_a), for $a = 1, \ldots, b$. In each step $c+1$ the new positions and velocities are calculated by (4).

$$\begin{aligned} {v_a}^{c+1} &= \omega {v_a}^c + \phi_1(g_{best} - {x_a}^c) + \phi_2(p_{best} - {x_a}^c) \\ {x_a}^{c+1} &= {x_a}^c + {v_a}^{c+1} \end{aligned} \quad (4)$$

where $\phi_1 = r_1 * a_g$, $\phi_2 = r_2 * a_l$, $\omega \in \Re$ means the inertia, a_g and $a_l \in \Re$ are, respectively, global and local acceleration constants, r_1 and r_2 are random numbers uniformly distributed in $(0, 1)$. ϕ_1 and ϕ_2 are random global and local accelerations. The position and velocity of each particle is updated taking into account the objective function.

The inertia, ω, plays a key role in the process of providing balance between exploration and exploitation processes. This parameter determines the contribution rate of a particle's previous velocity to its velocity at the current time step [15].

Moreover, Bansal et al. [15] have described fifteen different formula to calculate particle inertia during PSO iterations, but in this work the linear weight decreasing inertia was used, which has a small error. There are several other functions that can adjust this parameter, as can be seen in Bansal et al. [15], Nickabadi et al. [16] and Qin et al. [17].

V. DIPSO Algorithm

This work proposes a hybrid algorithm to solve multi-objective FJSP, whose foundations are PSO and GA algorithms. Since the multi-objective aspect was also considered, the Fast Non-dominated Sorting (FNS) algorithm is used to determine the fronts and Pareto-optimal rank, in order to optimize (minimize) three objectives. As the main features of each paradigm have already been considered, this proposal intends to obtain a fast convergence and avoid local minima. This aspect can be justified through the fact that related studies have shown that PSO-based algorithms have fast convergence. This should be taken into consideration mainly when dealing with complex problems such as FJSP, as the probability of reaching local minima is high. Thus, genetic operators such as crossover and mutation allow diversity to be maintained increased, and therefore to obtain convergence for best solutions [8].

A. Particle Representation

In Cheng et al. [1], individual representations for scheduling problems using GA are presented and discussed. For example, there are representations that are based on operation, others based on jobs, among others. In Gen et al. [11] there can be seen a representation of the first type. Due to the possibility of immediate recovery of scheduling and simplicity of calculating fitness, [7], [8] and [10] have also used this representation. In addition, this representation allows maintaining the viability of individuals obtained after crossover or mutation processes.

Then, in this study, it was decided to use the representation described in [11], where an individual (or particle) is identified by two vectors of size N, X_{op} and X_{mach}. The first one contains each operation of each job in accordance with the operation's execution sequence. The second one contains the machine in which each operation is executed. In X_{op}, each job is identified by an integer j, for $j = 1, \ldots, n$, and each job operation is represented by the integer j that corresponds to the job. Thus, the n_j operations of the j^{th} job are represented by n_j positions in X_{op} with value j. The first j in X_{op} indicates the first operation, and the last j indicates the last operation of job j. Each position of the X_{mach} vector contains a value k, $1 \leq k \leq m$, which indicates the machine in which the operation of position corresponding to X_{op} is executed.

X_{op}:	1	3	1	4	2	2	4	3
X_{mach}:	3	1	3	1	2	4	2	5

Fig. 1. Particle Representation

Figure 1 shows an example of this representation for a problem with $n = 4$ and $m = 5$, where each job contains two operations. It was observed that the operations of job $j = 1$ are represented in the first and third positions of X_{op} both are executed on machine three, as can be seen in X_{mach}.

B. Genetic Operators

As previously discussed about the fast convergence of PSO, some proposals had introduced genetic operators in order to avoid local minima. On the other hand, crossover operators are also able to accelerate the convergence. In this context, this work proposes the use of two crossover operators, one of them being responsible for the convergence while the other is responsible for population diversification. The first crossover operator is applied to X_{op} and X_{mach} in an analogue way, maintaining the resulting particles in the same machine association as in parent particles. In addition, the second operator is applied differently to each vector and is able to produce a greater diversity since there are more changes in relation to machine associations. This feature introduced in this work aims to maintain diversity, as the use of PSO is able to accelerate convergence.

In Figure 2 an example of the first crossover is shown. Consider two particles P_1 and P_2 with $n = 4$ jobs and the respective operation and machine vectors. This operator randomly selects $\frac{n}{2}$ jobs, given by $J_1 = \{2, 4\}$, the operations of these selected jobs will be positioned into one of the resulting particles, C_1, in the same positions in which they appear in P_1; also the machines associated with these operations will be placed in the same positions. All remaining positions from C_1 are occupied by the non-selected job operations in accordance with P_2 order.

The same procedure is carried out for obtaining the resulting C_2 particle, the difference is that the non-selected jobs, $J_2 = \{1, 3\}$, are considered. Then, the operations belonging to J_2 are positioned in C_2, in the same positions in which they appear in P_2 and the remaining positions are occupied by operations of J_1 in same order they appear in P_1. This operation is defined by [10] and is denominated improved precedence operation crossover (IPOX). Therefore, the machines associated with operations transferred to X_{op} vector of C_2 are also transferred to X_{mach} vector of C_2.

The second crossover operator is also applied to both vectors; however this occurs in two stages that are different in each vector. The first stage occurs in operation vectors, i.e. X_{op}. Let P_1 and P_2 be two particles and a randomly selected interval between the first and last positions of P_1

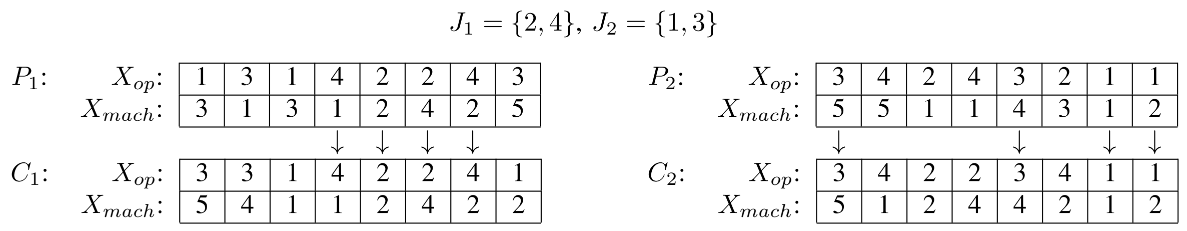
$J_1 = \{2,4\}, J_2 = \{1,3\}$
P_1: X_{op}: 1 3 1 4 2 2 4 3
X_{mach}: 3 1 3 1 2 4 2 5
C_1: X_{op}: 3 3 1 4 2 2 4 1
X_{mach}: 5 4 1 1 2 4 2 2
P_2: X_{op}: 3 4 2 4 3 2 1 1
X_{mach}: 5 5 1 1 4 3 1 2
C_2: X_{op}: 3 4 2 2 3 4 1 1
X_{mach}: 5 1 2 4 4 2 1 2

Fig. 2. Crossover operator: IPOX

or P_2. The resulting particle, C_1, receives all the values that belong to the interval in the same positions of P_1 and those that do not belong to the interval are filled with the remaining operations, according to the order they appear in P_2. The second resulting particle, C_2, is generated analogously: it receives all the values that do not belong to the interval in the same positions of P_1 and those that belong to the interval are filled with the remaining operations, according to the order they appear in P_2. In relation to vector X_{mach}, the second stage, it randomly selects a position and exchanges the machines from these positions between particles. This can be seen as a single point crossover.

The two-stage procedure allows the second crossover to generate new particles by changing the machine that will execute the operation, and consequently implies alterations at the start and end times of operation's execution. This fact increases the population diversity and in turn this operator is denominated Diversity Crossover (DX).

Figure 3 shows an example of the DX first-stage with $n = 4$ jobs, where each one contains two operations. Suppose $[0, 3]$ is the selected interval, the positions that belong to this interval are copied from P_1 to C_1. In this case, one operation of each job was added to C_1 in the following order 1, 3, 2 and 4. The second operation of each job is included in C_1 in the order they appear in P_2. Similarly, the C_2 particle is obtained. The $[4, 7]$ interval is copied from P_1 to C_2 and the remaining positions (from C_2) are completed with the remaining operations in accordance with the order they appear in P_2. Figure 4 shows an example of the DX second-stage, where position 4 was selected and intervals of P_1 and P_2 formed between the selected position and the last position will be exchanged between particles.

The mutation operator randomly selects two positions pos_1 and pos_2 of vector X_{op} and makes $X_{op}[pos_1] = X_{op}[pos_2]$ and for every position pos, so that $pos_1 < pos \leq pos_2$, makes $X_{op}[pos] = X_{op}[pos - 1]$. The mutation changes the values of two random positions of vector X_{mach} for randomly generated values, where the new values should be different from previous ones and lower than or equal to m. Figure 5 shows an example of this operator.

As shown above, all operators used, especially those applied to X_{op}, preserve the number and the order of job operations. Therefore, no check or correction of invalid solutions is carried out. This avoids increasing the execution time of the proposed algorithm.

C. Pseudo code

DIPSO

```
var
  gBestSet: Set;
  pBest: Individual;
begin
  Randomly start population
  Calculate objectives fitness
  Execute FNS algorithm, obtaining
    Front 1
  Determine the gbestSet and pbest
  while goal is not reached
    For each particle of the population
      Select_best(P1, P2, P3)
      Mutation according to probability
      Calculate objectives fitness
      Execute FNS algorithm, obtaining
        Front 1
      Update gbestSet and pbest
```

In the proposed hybrid algorithm the initial population is randomly generated and the fitness of each objective is calculated according to Equations 1, 2 and 3, for every particle. Then, the FNS algorithm is executed in order to determine Front 1 (F1), the best local (p_{best}) and the best global set (g_{best}). One observes that at the beginning, the particle has only one position, therefore in this case the p_{best} for each particle is the particle itself. In the loop, the same previous steps are executed in addition to the mutation operator before updating the particle's position.

In general, g_{best} is a particle, but in multi-objective algorithms it is usual for more than one non-dominated best position to occur. Then the DIPSO algorithm saves these positions in the g_{best} set during iterations. The current g_{best} is determined by the FNS algorithm taking into account the new particle and previous g_{best} set. This way, all the non-dominated solutions can influence the new particle position and thus increase diversity.

According to Equation 4, a PSO algorithm updates the particle position by means of p_{best} and g_{best} particles, which is one of the most important PSO features. However, Niu et al. [18] have replaced this equation by crossover and mutation operators. Thus, the new particle position is the best particle selected among three other particles, which are the resulting crossing over between the current particle and p_{best} or g_{best}, and the mutation of the current particle. In the DIPSO algorithm, this is the procedure $Select_{best}(P_1, P_2, P_3)$, which

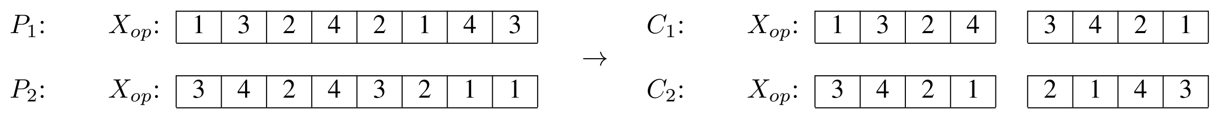
P_1: X_{op}: 1 3 2 4 2 1 4 3
P_2: X_{op}: 3 4 2 4 3 2 1 1
$\rightarrow$
C_1: X_{op}: 1 3 2 4 | 3 4 2 1
C_2: X_{op}: 3 4 2 1 | 2 1 4 3

Fig. 3. Crossover operator: DX - First Stage

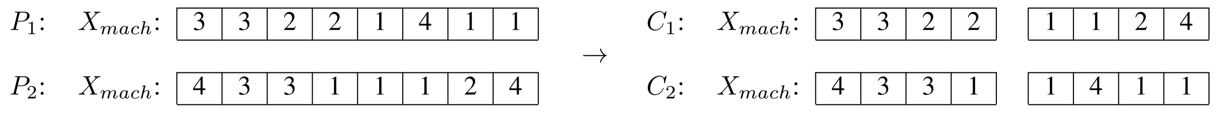
P_1: X_{mach}: 3 3 2 2 1 4 1 1
P_2: X_{mach}: 4 3 3 1 1 1 2 4
$\rightarrow$
C_1: X_{mach}: 3 3 2 2 | 1 1 2 4
C_2: X_{mach}: 4 3 3 1 | 1 4 1 1

Fig. 4. Crossover operator: DX - Second Stage

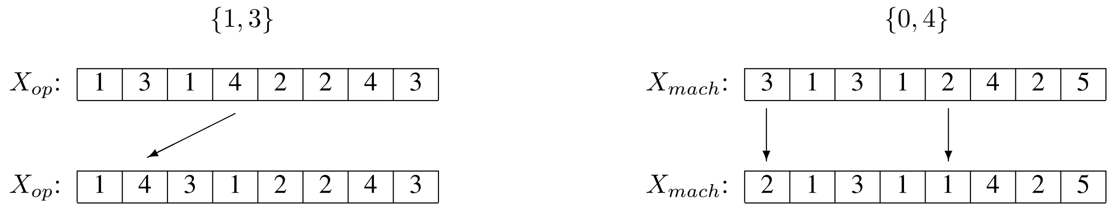
$\{1,3\}$
X_{op}: 1 3 1 4 2 2 4 3
X_{op}: 1 4 3 1 2 2 4 3
$\{0,4\}$
X_{mach}: 3 1 3 1 2 4 2 5
X_{mach}: 2 1 3 1 1 4 2 5

Fig. 5. Mutation

selects, by means of the FNS algorithm, the best particle among $P_1 = crossover(P_k, selectOne(g_{best}))$, $P_2 = crossover(P_k, p_{best})$ and $P_3 = mutation(P_k)$, where P_k is the current particle (position) and $selectOne$ is a function that randomly selects an element from g_{best} set. If the FNS algorithm determines that among P_1, P_2 and P_3, the set of non-dominated particles is formed by more than one element, the $Select_{best}$ procedure will randomly select one of this set.

In fact, the new P_k position is a particle, but if one considers that each particle in the search space is a position in the space, the $select_{best}(P_1, P_2, P_3)$ procedure can be seen as an equation that updates the position even if the particle itself is changed. In these terms, the velocity can be considered the proximity of the current particle in relation to p_{best} or g_{best} .

As two crossover operators are proposed in DIPSO, the $Select_{best}()$ procedure randomly choose one of them according to p_{cc}, the probability of choosing the convergence crossover (Figure 2) and p_{dx}, the probability of choosing the diversity crossover (Figure 3), where $p_{cc} + p_{dx} = 100\%$. It has been observed that the greater the p_{dx}, the greater the population diversity and the greater the p_{cc}, the faster the convergence. At each DIPSO iteration, both crossovers are used, guaranteeing that part of the population evolves and part diversifies.

VI. Results

FJSP instances have been obtained from Kacem et al. [19], which are represented by $n \times m$ and widely used by most studies, as seen in [4], [9], [10] and [12]. The DIPSO algorithm was coded in Java language and run on an Intel Core I7 (2.0 GHz) processor with 8 GB of RAM memory. Four of these instances, 4×5, 8×8, 10×7 and 10×10, were tested through the four experiments. In all experiments, the algorithm executions considered the objectives defined in Section II, makespan (M), total workloads of machines (TW) and the maximum workloads of machines (W).

The DIPSO algorithm can cope with some features, especially those related to the use of two crossover operators and the g_{best} set, an experiment planning was elaborated. Then, comparisons among results with and without these features are performed. Each experiment was executed thirty times and the results presented are the $F1$ elements for which the algorithm converged. In each experiment, the characteristics of the algorithm evolution such as diversity and convergence are analyzed. The convergence rates shown in experiments consider the sum of the number of repeated elements in the final result. In every execution the population size = 1000, mutation rate= 23%, maximum number of iterations = 1400, p_{cc} = 80% and p_{dx} = 20%. These values values were empirically determined according to the most complex problem (10×10). Multiple runs of the algorithm varying the parameter values were performed to check their behavior. Thus, based on the results obtained, the values of parameters were determined.

A. First Experiment

In order to test the two DIPSO features, this experiment did not considered any of these features. The algorithm executions have taken into account one of the crossover operators at each time and g_{best} was only a particle. Thus, only one particle is maintained as g_{best} rather than $F1$. To determine the particle, the $selectOne$ method is used to choose one $F1$ element. Table I presents the results using only the IPOX crossover (Figure 2).

The results in Table I are similar to those presented by [4], [9], [10] and [12]. As each experiment was executed thirty times, convergence rate could be observed. In this case, in 95% of executions the algorithm reached the convergence in only one solution and the convergence rate of the executions was around 30% and 40%.

TABLE I

RESULTS: FIRST EXPERIMENT

	4×5			8×8		10×7		10×10	
M	11	12	13	16	15	12	11	8	8
TW	32	32	33	73	75	60	61	41	42
W	10	8	7	13	12	12	11	7	6

In tests of the 8×8 and 10×10 problems, the DX crossover (Figure 3 and 4) was applied. It was observed that the number of different particles was 50% greater than when the IPOX crossover was executed. During the execution of the 4×5 problem, results similar to IPOX were obtained.

B. Second Experiment

The tests for the first experiment carried out only one of the crossovers at a time. Then, in the second experiment, both were tested together according to $p_{cc} = 80$ (IPOX probability) and $p_{dx} = 20$ (DX probability). The g_{best} remains with only one particle. Table II presents the results related to this experiment. In this experiment, the solutions were also similar to those shown in previous works. Through the solution number for the 10×10 problem, it could be inferred that the diversity rate increased.

TABLE II

RESULTS: SECOND EXPERIMENT

	4×5			8×8		10×7		10×10		
M	11	12	13	16	15	12	11	8	8	7
TW	32	32	33	73	75	60	61	41	42	42
W	10	8	7	13	12	12	11	7	5	6

The introduction of the diversity crossover (DX) increased the number of different particles at an average rate of 21%, when compared to the first experiment using only IPOX. Due to this increase in the diversity rate, the ability to avoid local minima also increased. In 70% of executions, the algorithm reached convergence during the first 250 iterations. In relation to the 4×5 problem, 80% of executions reached convergence in the 25^{th} iteration.

C. Third Experiment

In this case, the executions have considered only the IPOX crossover and g_best is a set. The results of this experiment are described in Table III, where similarities to other experiments can be observed. However, when compared to the second experiment, three new solutions were identified. The main observation is the convergence to more than one solution. In particular, the executions of the 4×5 problem showed that 80% of the population were solutions.

D. Fourth Experiment: DIPSO

In previous experiments, each DIPSO feature was tested separately. Now, the fourth experiment put together both features and g_{best} is a set, which demonstrated that these features introduced more diversity in the population evolution. A greater number of optima solutions was presented, as seen in Table IV. The number of different particle generated during the algorithm evolution is similar to that of the second experiment.

In 78% of executions, the DIPSO convergence occurred before the 250^{th} iteration and for all results the algorithm converged for more than one optimum solution. In particular, every execution of the 4×5 problem produced a convergence for every solution presented in Table IV. In addition, every execution of this problem converged in at least three solutions of the final result, at 25^{th} iteration. In relation to the 8×8 and 10×10 problems, 68% of executions converged before the 250^{th} iteration.

Comparisons among DIPSO and other works MOGA [10], MOEA-GLS [12], PSO + SA [9], PSO + TS [4], an Hybrid Tabu Search algorithm (HTS) [20] and a Multi-objective Evolutionary Algorithm (MOEA) [21] showed similar results, as observed in Tables V, VI, VII and VIII. When only PSO-based algorithms are considered, one observes that the number of solutions presented by DIPSO is the highest and there was convergence for more than one result in the same execution.

VII. Conclusion and further works

The DIPSO algorithm for solving multi-objective FJSP using PSO, genetic operators and the FNS procedure was described. Also, a new crossover operator, DX, was developed in order to cope with diversity while maintaining convergence. In addition to the DX operator, DIPSO used the IPOX and the proposal efficiency was demonstrated, since some results were improved and the best results were also reached.

The composition of PSO and genetic operators avoided local minima, as previously shown in similar studies. However, operators not only ensure convergence, but also introduce diversity. This was demonstrated by DIPSO through the simultaneous use of IPOX and DX. It was also proved that a set of best solution contributes to diversity and convergence rates.

Further studies should compare NSGA II and DIPSO in order to evaluate the diversity and evolution of the population along with the results of each algorithm.

References

[1] Cheng, R., Gen, M. and Tsujimura, Y., A Tutorial Survey of Job-Shop Scheduling Problems using Genetic Algorithms - I. Representation, *Computers & Industrial Engineering*, 30 (4) (1996) 983–997.

[2] Lenstra, J.K. and Rinnooy, K.A.H.G., Computational Complexity of Discrete Optimization Problems, *Annals of Discrete Mathematics*, 4 (1979) 121–140.

[3] Ho, N.B. and Tay, J.C., Genace: An Efficient Cultural Algorithm for Solving the Flexible Job-Shop Problem, *Proceedings ofIEEE* (2004) 1759–1766.

[4] Zhang, G., Shao, X., Li, P. and Gao, L. An Effective Hybrid Particle Swarm Optimization Algorithm for Multi-objective Flexible Job-shop Scheduling Problem, *Computers & Industrial Engineering*, 56, 1309–1318 (2009)

[5] Deb, K. Pratap, A., Agrawal, S. and Meyarivan, T., A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, V. 6, N. 2, pp. 182–197.

TABLE III

RESULTS: THIRD EXPERIMENT

	4×5				8×8			10×7			10×10		
M	11	12	13	11	16	15	14	12	11	11	8	8	7
TW	32	32	33	34	73	75	77	60	61	62	41	42	42
W	10	8	7	9	13	12	12	12	11	10	7	5	6

TABLE IV

RESULTS FOR THE FOURTH EXPERIMENT: DIPSO

	4×5				8×8				10×7			10×10			
M	11	12	13	11	16	15	14	16	12	11	11	8	8	7	7
TW	32	32	33	34	73	75	77	77	60	61	62	41	42	42	43
W	10	8	7	9	13	12	12	11	12	11	10	7	5	6	5

TABLE V

RESULT COMPARISON - 4×5

	DIPSO				PSO + TS [4]	MOGA [10]		
M	11	11	13	12	11	11	11	12
TW	32	34	33	32	32	32	34	32
W	10	9	7	8	10	10	9	8

TABLE VI

RESULT COMPARISON - 8×8

	DIPSO				PSO + TS [4]		PSO + SA [9]		MOGA [10]			MOEA-GLS [12]			
M	16	15	14	16	14	15	16	15	15	15	16	16	15	14	16
TW	73	75	77	77	77	75	73	75	81	75	73	73	75	77	77
W	13	12	12	11	12	12	13	12	11	12	13	13	12	12	11

TABLE VII

RESULT COMPARISON - 10×7

	DIPSO			HTS [20]		MOEA [21]		
M	12	11	11	11	11	12	11	11
TW	60	61	62	61	62	60	61	62
W	12	11	10	11	10	12	11	10

TABLE VIII

RESULT COMPARISON - 10×10

	DIPSO				PSO + TS [4]	PSO + SA [9]	MOGA [10]				MOEA-GLS [12]			
M	8	8	7	7	7	7	8	8	7	7	8	8	7	7
TW	41	42	43	42	43	44	41	42	45	42	41	42	43	42
W	7	5	5	6	6	6	7	5	5	6	7	5	5	6

[6] Jia, Z., Chen, H. and Tang, J. A New Multi-objective Fully-Informed Particle Swarm Algorithm for Flexible Job-Shop Scheduling Problems, *International Conference on Computational Intelligence and Security Workshops*, 191–194, 2007

[7] Ling-li, Z., Feng-Xing, Z. and Xiao-hong, X. Mathematical Model and Hybrid Particle Swarm Optimization for Flexible Job-Shop Scheduling Problem, *Genetic and Evolutionary Computation Conference*, 2009.

[8] Xiao-hong, X., Ling-li, Z. and Yue-wen, F., Hybrid Particle Swarm Optimization for Flexible Job-Shop Scheduling Problem and Its Implementation, *IEEE International Conference on Information and Automation*, 1155–1159, (2010).

[9] Xia, W. J. and Wu, Z. M. An effective hybrid optimization approach for multiobjective flexible job-shop scheduling problems, *Computers and Industrial Engineering*, 48(2), 409-425, 2005.

[10] Wang, X., Gao, L., Zhang, C. and Shao, X. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem, *The International Journal of Advanced Manufacturing Technology*, V. 51, N. 5–8, 757–767, 2010.

[11] Gen, M., Tsujimura, Y. and Kubota, E. Solving job-shop Scheduling Problems by Genetic Algorithm, *International Conference on Systems, Man, and Cybernetics*, V. 2, 1577–1582, 1994.

[12] Binh Ho, N., and Cing Tay, J. Using Evolutionary Computation and

Local Search for Solving Multi-objective Flexible Job Shop Problems, *Genetic and Evolutionary Computation Conference*, 2007.

[13] Kennedy, J. and Eberhart, R., Particle swarm optimization, *International Conference on Neural Networks*, pp. 1942–1948, 1995.

[14] Martínez, J. L. F. and Gonzalo, E. G., The PSO family: deduction, stochastic analysis and comparison, *Swarm Intelligence*, V. 3, pp. 245–273, 2009.

[15] Bansal, J. C., Singh, P. K., Saraswat, M., Verma, A., Jadon, S.S. and Abraham, A., Inertia Weight Strategies in Particle Swarm Optimization, *Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp.633–640, 2011.

[16] Nickabadi, A., Ebadzadeh, M. M. and Safabakhsh, R., A novel particle swarm optimization algorithm with adaptive inertia weight, *Applied Soft Computing* V. 11, N. 4, pp. 3658–3670, 2011.

[17] Qin, Z., Yu, F., Shi, Z. and Wang, Y., Adaptive Inertia Weight Particle Swarm Optimization, *Artificial Intelligence and Soft Computing*, V. 4029, pp. 450–459, 2006.

[18] Niu, Q., Jiao, B. and Gu, X., Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. *Applied Mathematics and Computation*, V. 205 (1), 148–158, 2008.

[19] Kacem, I., Hammadi, S. and Borne, P., Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, *Mathematics and Computers in Simulation*, 60, 245–276, 2002.

[20] Li, J., Pan, Q. and Liang, Y. An Effective Hybrid Tabu Search Algorithm for Multi-objective Flexible Job-shop Scheduling Problems, *Computers and Industrial Engineering*, 59, 647–662, 2010.

[21] Chiang, T. and Lin, H. A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling, *International Journal of Production Economics*, 141, 87–98, 2013.