A clustering search metaheuristic for the bi-objective flexible job shop scheduling problem

Wagner A. S. Altoé, Dayan de C. Bissoli, Geraldo R. Mauri, André R. S. Amaral
Graduate School of Computer Science (PPGI)
Federal University of Espírito Santo (UFES)
Vitória, Brazil
altoe.wagner@gmail.com, dayan.bissoli@ufes.br, geraldo.mauri@ufes.br, amaral@inf.ufes.br

Abstract—The Flexible Job Shop Scheduling Problem (FJSP) is an extension of Job Shop Scheduling (JSP), which is closer to reality, allowing an operation of a given job to be processed by alternative machines. Considering that for some industries it is relevant to consider for more than one objective, the FJSP is treated in this study in a multiobjective way with the following criteria: the last processing time of the last operation, called makespan, and the total tardiness. Therefore, it is proposed an algorithm based on the Clustering Search (CS) metaheuristic to generate solutions, and thus produce a set of non-dominated solutions in order to obtain the Pareto frontier, providing to decision maker a set of quality solutions. To evaluate the CS, we proposed a set of instances considering due dates for the jobs, to enable analysis of the bi-objective FJSP (BOFJSP). The CS results were competitive when compared to the literature, generating several non-dominated solutions.

Index Terms—flexible job shop scheduling, multiobjective optimization, metaheuristic, clustering search, Pareto frontier.

I. INTRODUCTION

In the current competitive environment, for an industry to keep up with market growth, it becomes necessary to efficiently schedule production. Industries must meet deadlines and minimize costs. So, organizations must plan their activities in such a way that resources are used efficiently [1].

The scheduling of activities in an industry consists of determining the sequence of tasks (jobs) in each machine in order to optimize some objective. In this category of problems we have a set of tasks, each one formed by a sequence of operations, and each operation is performed by a single specific machine. For a job to be processed, it must move through all machines in the pre-determined sequence. The purpose of this problem, called Job Shop Scheduling (JSP), comes down to designate the order of the operations for each job in the previously defined machines to optimize some criterion [2].

Considering the limitation of the JSP that each operation is processed by a specific machine, Brucker and Schlie [3] proposed the Flexible Job Shop Scheduling (FJSP). In this perspective, it is possible that some jobs can be processed by alternative machines, that is, the same job may have variations in the sequencing of their production [4].

The FJSP can have several different criteria as an objective function, the processing time of the last operation, called makespan, being the most studied. However other objectives can be evaluated, such as those related to the date of delivery, the sum of the delay in the delivery of products (total tardiness), among others [5], [6].

With the continuous need for quality improvement in the decision-making process, and the goal of making the model closer to the real operational context of the industries, comes the need to address the problem considering more than one objective. Such problems are called multiobjective problems [1], [7].

In multiobjective optimization problems, it is expected to satisfy all objectives. However, find this solution is not trivial, because the objectives tend to conflict with each other, that is, by improving one objective, it can lead to the worsening of the other (trade-off). Multiobjective problems approaches can be classified as follows [8]:

- Single objective: one objective is selected and the others are applied in the constraints of the problem;
- Linearization: the objective function is linearized, and for each objective is given a multiplier value, which is classified according to the importance of the objective. This approach uses strategic level knowledge to weight the objective function as needed;
- Pareto frontier: considers that the objectives of a multi-objective problem are usually conflicting, that is, by improving one objective, it can affect and / or worsen the other. There is no single solution that is optimal for all goals. In this situation, we seek for a set of non-dominated solutions that satisfy the problem constraints.

To better understand the Pareto frontier approach, Jozefowicz et al. [8] define the dominance concept as follows. A solution \( y = (y_1, y_2, ... y_n) \) dominates a solution \( z = (z_1, z_2, ... z_n) \) if and only if \( \forall i \in \{1...n\}, y_i \leq z_i \) and \( \exists i \in \{1...n\}, y_i < z_i \).

These classes of problems can be solved through exact methods and / or from mathematical programming models. However, such methods tend to be ineffective with instances that have a larger dimension, that is, they can not obtain an optimal solution, or even a feasible solution in reasonable computational time, which is common if we consider real FJSP applications [9]. Thus, it arises the motivation to use heuristic methods that are able to generate good quality solutions to these problems.
In order to extend the approaches made by Moreira et al. [10] and Lei [9], which addressed the JSP in a multiobjective way, minimizing simultaneously the objectives makespan and the total tardiness of jobs, in this paper a bi-objective version of the FJSP (BOFJSP) is treated, considering these same criteria.

Whereas BOFJSP is a problem that inherits the characteristics of the JSP, it is defined as NP-Hard [11]. Thus, as a solution method, we implemented the Clustering Search (CS) metaheuristic [12], considering that it has been showing good results for other combinatorial optimization problems, and that no application of CS to the FJSP has been found in literature so far [6], [13].

This paper is divided as follows: Section II presents some works related to the BOFJSP. The Section III details the methodology developed in this study. Computational experiments are described in Section IV. Section V presents the conclusions and then the references used in this work are listed.

II. Related Literature

For the classical FJSP, more recently, Cruz-Chávez et al. [14] presented an algorithm based on the Simulated Annealing (SA) metaheuristic with a mechanism of partial scheduling and a cooling mechanism, which is function of the standard deviation. It was found that SA with the proposed mechanism, called SA-Partial Controlled (SAPC), converges faster to good solutions than without the proposed mechanism. Another metaheuristic that combines the methods Scatter Search and Path-Relinking with Tabu Search (SSPR) was developed by González et al. [15] for the FJSP. The authors sought to combine the diversification characteristic provided by Scatter Search and Path-Relinking, with the intensifying nature of TS. Both studies presented efficient results, providing new best known values for classic instances.

There are a large number of works in the literature that propose methods to solve different versions of the FJSP [6], [16]. However, with the aim to approximate the FJSP to real-world cases, it is considered the model proposed by Özgüven et al. [25], with an adaptation in the respective model performed by Melo and Ronconi [5], to propose a new adaptation in the mixed integer linear programming model for the FJSP considering as objective function to simultaneously minimize the makespan and total tardiness of the jobs, as treated in [10] and [9], we did not find some application to the FJSP. So, it becomes interesting to study this approach.

III. Methodology

A. Problem Modeling

In order to approximate the FJSP to real-world cases, it is considered the model proposed by Özgüven et al. [25], with an adaptation in the respective model performed by Melo and Ronconi [5], to propose a new adaptation in the mixed integer linear programming model for the FJSP considering as objective function to simultaneously minimize the makespan and total tardiness of jobs, thus defining the bi-objective FJSP (BOFJSP):

Indices and sets
- \( J \): set of jobs;
- \( M \): set of machines;
- \( O \): set of operations;
- \( i \): jobs \( i, i \in J \);
- \( j \): operations \( j, j \in O \);
- \( k \): machines \( k \in M \);
- \( O_i \): ordered set of operations of job \( i \) \( (O_i \subseteq O) \);
- \( O_{ij} \): operation \( j \) of the job \( i \) \( (O_{ij} \subseteq O_i) \);
- \( M_j \): the set of alternative machines on which operation \( j \) can be processed, \( (M_j \subseteq M) \);
- \( M_j \cap M_{j'} \): the set of machines on which operations \( j \) and \( j' \) can be processed.
- \( P_{ij} \): set of operations that must precede the operation \( O_{ij} \) in job.

Parameters
- \( t_{ijk} \): the processing time of operation \( O_{ij} \) on machine \( k \);
- \( d_i \): due date of the job \( i \);
- \( L \): a large number.

Decision variables
- \( X_{ijk} \): 1, if machine \( k \) is selected for operation \( O_{ij} \); 0, otherwise;
S_{ijk}: the starting time of operation O_{ij} on machine k;
C_{ijk}: the completion time of operation O_{ij} on machine k;
Y_{ij'j''k}: 1, if operation O_{ij} immediately precedes operation O_{ij''} on machine k; 0, otherwise
C_i: the completion time of job i;
T_i: tardiness of job i;
C_{max}: maximum completion time over all jobs (makespan).

\[
\text{min } \sum_{i \in J} T_i \quad (1)
\]

\[
\text{min } C_{max} \quad (2)
\]

\[
\text{s.t. } \sum_{k \in M_j} S_{ijk} \leq X_{ijk} \times L, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (4)
\]

\[
C_{ijk} \leq X_{ijk} \times L, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (5)
\]

\[
C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) \times L, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (6)
\]

\[
S_{ijk} \geq C_{ij'j''k} - (Y_{ij'j''k}) \times L, \forall i < i', \forall j \in O_i, \forall j' \in O_{ij'}, \forall k \in M_j \quad (7)
\]

\[
S_{ij'j''k} \geq C_{ij'j''k} - (1 - Y_{ij'j''k}) \times L, \forall i < i', \forall j \in O_i, \forall j' \in O_{ij'}, \forall k \in M_j \cap M_{ij'} \quad (8)
\]

\[
\sum_{k \in M_j} S_{ijk} \geq \sum_{k' \in M_{ij'}} C_{ij'k'}, \forall i \in J, \forall j \in O_i, \forall O_{ij'} \in P_{ij} \quad (9)
\]

\[
C_i \geq \sum_{k \in M_j} C_{ijk}, \forall i \in J, \forall j \in O_i \quad (10)
\]

\[
T_i \geq C_i - d_i, \forall i \in J \quad (11)
\]

\[
X_{ijk} \in \{0, 1\}, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (12)
\]

\[
S_{ijk} \geq 0, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (13)
\]

\[
C_{ijk} \geq 0, \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (14)
\]

\[
Y_{ij'j''k} \in \{0, 1\}, \forall i < i', \forall j \in O_i, \forall j' \in O_{ij'}, \forall k \in M_j \cap M_{ij'} \quad (15)
\]

\[
C_i \geq 0, \forall i \in J \quad (16)
\]

\[
T_i \geq 0, \forall i \in J \quad (17)
\]

The objective function (1) aims to minimize the total tardiness and objective function (2) minimizes the makespan. The constraint (3) ensures that each operation will have only one machine to process it, while the constraints (4) and (5) certify that if an operation is not allocated to a machine, the start and end times of processing for that machine are equal to zero. The constraint (6) ensures that the processing end time of an operation assigned to a machine will be greater than or equal to the sum of its start time plus the processing time of this machine. Constraints (7) and (8) limit a machine from processing more than one operation simultaneously. Constraint (9) guarantees the precedence of operations, and the constraint (10) certifies that the completion time of the job is greater than or equal to the time of completion of the operations. The constraint (11) certifies that the job will not be completed before having completed all operations, and the constraints (12) to (17) define the domain of the variables. Finally, the constraints (12) to (15) treat the integrity of the variables.

Based on the mathematical model presented, the complexity of the problem is highlighted because it is a problem with more than one objective and, considering the conflicting characteristics of the objectives to with each other, a single solution tends to prioritize one objective over the other. Thus, obtaining the Pareto frontier becomes an interesting alternative to solve the problem, since it is possible for the decision maker to evaluate a set of quality solutions and thus to define which objective to prioritize, according to your convenience.

B. Proposed instances

Considering the characteristic of the proposed model, which considers the criterion of minimize the total tardiness besides the makespan, we not find in the literature a set of instances that allow perform computational experiments. Consequently, there is a need to propose new instances, considering due dates, to evaluate the method proposed in this work. Thus, the proposed instances are described in this section.

We adapted the instances proposed by Brandimarte [26], because he did not define due dates. The instances were adapted in a similar way to the dates generated by Armentano and Scrich [27] and Moreira et al. [10], as shown in Equations (18), (19) and (20).

\[
d_i = \left\lfloor (\beta \times \sum_{i=1}^{M} \bar{t}_{ij}) \right\rfloor \quad (18)
\]

\[
\beta_1 = ((J \times M)/1000) + 0.5 \quad (19)
\]

\[
\beta_2 = \beta_1 + 1 \quad (20)
\]

The parameters \(\beta_1\) and \(\beta_2\) are used to determine the due date of jobs, being \(\beta_2\) a more relaxed way to generate the due dates. The due date of the job \(i\) is referenced in \(d_i\), being \(\bar{t}_{ij}\) the average processing time of operations of the job \(i\). To avoid rounding or difference problems when considering decimal places, we have set truncate the due date values.

C. Clustering Search metaheuristic

The Clustering Search metaheuristic (CS) is a generic definition of Evolutionary Clustering Search (ECS) [28]. The main difference between the methods is that ECS uses an evolutionary algorithm to generate solutions, while CS can use any metaheuristic [12], [13].

In CS, a cluster is defined by a set of three elements \(C = \{c, v, r\}\): the best solution of the cluster is its center \(c\); the number of solutions associated with each cluster defines the volume \(v\); and the parameter \(r\) indicates the number of attempts without improvement after applying a local search [29].

When the volume \(v\) reaches the parameter \(\lambda\), it becomes promising and then local search is applied. The parameter \(r\)
is incremented when better solutions are not obtained. If the inefficiency index \( r \) reaches the value \( r_{\text{max}} \), a perturbation occurs in the center of the cluster \( c \), in order to diversify the search.

As a solution-generating metaheuristic for CS, we used Simulated Annealing (SA) [30]. The SA is based on an analogy with thermodynamic to optimize solutions to combinatorial optimization problems.

SA performs perturbations in the current solution in an iterative way, seeking better neighbor solutions. An important feature of SA is the fact that it allows worse solutions as a form of diversification of the search. In doing this, SA tends to escape from local optimum, seeking a global optimum in later iterations. The probability of accepting solutions that are worse than the current one is decreased by the temperature.

Algorithm 1 shows the pseudocode of the proposed CS, using SA as the solution generator. SA runs until the timeout \( T_{\text{max}} \) expires. At each temperature, \( SA_{\text{max}} \) iterations are done and then the current SA solution is sent to the CS (lines 4-21).

The method is initialized by creating the number \( \gamma \) of clusters, with their respective characteristics of maximum volume \( \lambda \) and index of inefficacy \( r_{\text{max}} \). Then a initial solution is created and defined as the best solution. Next, the SA is started, where at each iteration a neighborhood is chosen randomly between \( N^1 \) or \( N^2 \) to generate a neighbor solution of the current solution (lines 9-10). A neighbor solution is accepted if it is better than the current solution (lines 11-16) or with a given probability (lines 17-19), otherwise.

The SA temperature is decreased (line 21) and if the maximum number of clusters is not reached, the current SA solution is assigned to the center of a new cluster (lines 22-24). After all \( \gamma \) clusters have been initialized, the new solutions generated by the SA are now included in existing clusters (line 25 ahead).

Next, the assimilation process begins, which is defined based on the similarity between the current SA solution and the center of all clusters. The current SA solution is inserted in the cluster whose center is most similar to it. The algorithm then verifies if the cluster has become promising and if the local search or a perturbation will be applied or if a new best solution has been found (lines 26-45).

In order to perform the bi-objective analysis, initially the CS generates solutions considering the objective functions in a separated way. Thus, first the CS is applied to the BOFJSP considering the makespan criterion, and then, the CS is applied considering the criterion total tardiness. In both cases, in the search process, when a new best solution is found, it is stored in a list of solutions \( \text{listSol} \). After the CS is finished, the process of removing the dominated solutions from the list of solutions \( \text{listSol} \) is started, resulting on the set of non-dominated solutions.

The parameters used in the proposed CS algorithm are:

- \( \gamma \): Maximum number of clusters;
- \( \lambda \): Maximum volume of clusters;
- \( r_{\text{max}} \): Limit to the index of ineffectiveness of clusters;
- \( T_0 \): Initial temperature of the SA;
- \( \alpha \): SA cooling rate;
- \( T_f \): Final temperature of the SA;
- \( SA_{\text{max}} \): Maximum number of iterations per temperature of the SA; and
- \( T_{\text{max}} \): SA execution time limit.

1) Neighborhood Structures: In order to search for better solutions, SA has the characteristic of exploring the neighborhood in an iterative way, making exchanges between neighbors. In the FJSP context, moves are performed by changing the assignment (e.g. by moving an operation from one machine to another), or by changing a sequence (e.g. by shifting an operation or swapping two operations) [31], [32]. Thus, two ways of exploring the neighborhood were used: \( N^1 \) and \( N^2 \), which are described below.

a) Neighborhood Structure \( N^1 \): The strategy used in the neighborhood structure \( N^1 \) is based on exchanging positions between operations on a machine. For this, we randomly select a machine and then randomly select an operation among the ones allocated in the machine. Only the first operation on the selected machine does not enter the raffle. This is due to how the exchange is performed: when selecting an operation, the exchange move is always performed with its nearest neighbor on the left.

In the Figure 1 an \( N^1 \) exchange move is shown. Initially, Machine 2 (M2) was randomly selected. M2 processes, in the following order, the operations \( O_{1,1} \), \( O_{2,1} \), \( O_{3,1} \) and \( O_{4,1} \). Then, a new raffle is carried out, and the operation \( O_{2,1} \) is selected. Then the exchange is made with the operation located to its left. Therefore, the updated sequence of operations processed in M2 is as follows: \( O_{2,1}, O_{1,1}, O_{3,1} \) and \( O_{4,1} \).

![Fig. 1. Exchange move: \( N^1 \).](image)

b) Neighborhood Structure \( N^2 \): The strategy used in the neighborhood structure \( N^2 \) is based on exploiting the flexibility. The method seeks to exchange an operation that has flexibility from one machine to another. For this, a raffle is initially realized to select an operation among those with flexibility. The machine on which the operation is allocated is identified. Then, randomly select another machine that can process it and then, exchange the operation from the current machine to the selected machine. The solution is repaired in order to maintain its viability.

Figure 2 presents an \( N^2 \) exchange move. Initially the operation \( O_{2,2} \) was randomly selected among those that have flexibility. It was identified that the \( O_{2,2} \) operation is allocated on the M1 machine. It was found that the \( O_{2,2} \) operation can be processed alternately in several machines, among which the machine M2 was randomly selected. In this way the \( O_{2,2} \) operation was exchanged from machine M1 to machine M2.
Algorithm 1: CS Pseudocode for BOFJSP

```
input: γ, λ, T_{max}, T_f, α, SA_{max}, Time_{max}, s
cluster ← 0; r_i ← 0; v_i ← 0; ∀i = 1..γ; qtd ← 1;
listSol[qtd] ← s*; SA_{maxIni} ← SA_{max};
while Time < Time_{max} do
    T ← T_0;
    while T > T_f do
        iter ← 0;
        while iter < SA_{max} do
            iter ← iter + 1;
            k ← random[1,2];
            s' ← N^k(s);
            if f(s') < f(s) then
                s ← s';
                if f(s') < f(s*) then
                    s* ← s'; listSol[qtd] ← s*; qtd + +;
                end
            end
        end
        if cluster < α then
            cluster + +; v_{cluster} + +; c_{cluster} ← s;
        end
        else
            i ← armin\{i ∈ \{1..γ\}\}\{D_i\};
            v_i + +; c_i ← best(c_i, s);
            if v_i = λ then
                v_i ← 1;
                if r_i = r_{max} then
                    r_i ← 0;
                    c_i ← N(random[1,2])(c_i);
                end
                else
                    LocalSearch(c_i);
                    if c_i improved then
                        r_i ← 0;
                    end
                end
            end
            s* ← best(s*, c_i);
            listSol[qtd] ← s*; qtd + +;
        end
    end
    T ← T × T_f;
if cluster < γ then
    cluster + +; v_{cluster} + +; c_{cluster} ← s;
else
    i ← armin\{i ∈ \{1..γ\}\}\{D_i\};
    v_i + +; c_i ← best(c_i, s);
    if v_i = λ then
        v_i ← 1;
        if r_i = r_{max} then
            r_i ← 0;
            c_i ← N(random[1,2])(c_i);
        end
        else
            LocalSearch(c_i);
            if c_i improved then
                r_i ← 0;
            end
        end
    end
    s* ← best(s*, c_i);
    listSol[qtd] ← s*; qtd + +;
end
return listSol
```

2) Similarity: The objective of this step is to identify which cluster has its center most similar to the current solution (line 27 of the Algorithm 1). This similarity is obtained by calculating the Hamming Distance ($D_i$) [33] to count the number of positions that the corresponding entries are different. At the end of the comparison, the solution is assigned to the cluster that is the least distant and, then, the cluster volume $v_i$ is increased. Next, the solution of the cluster center is compared with the solution generated by the SA and if the solution generated by the SA is better, it becomes the center of the cluster.

Figure 3 presents two distinct solutions $s_1$ and $s_2$. This structure refers to the order of processing of jobs on the machines. The solutions differ in the machine that processes the last job ($O_{2,2}$). The Hamming distance between the two solutions is calculated and its value equals one.

3) Local Search: Soon after the CS identifies the promising areas, a local search (LS) is carried out, in order to refine the solution. This LS helps in quickly conversion to a local optimum. As LS was implemented an algorithm based on the best improvement method [34]. The pseudocode of the LS is presented in the Algorithm 2. The LS is divided into two stages. First takes place an intensification in the neighborhood of the current solution $s$ by changing the sequence of the operations (lines 3-13). Then the flexibility is explored, changing the assignment of the operations in the current solution $s$ (lines 14-34). At the end of the search, it returns the best solution found.

4) Selection of non-dominated solutions: After completion of the CS algorithm, begins the process of selecting the non-dominated solutions. This process consists of searching in the solutions vector $listSol$, with the objective of compare, at each iteration, the solution $s_i$ with the other solutions $s_j$, verifying if $s_i$ dominates $s_j$. If the solution $s_i$ is dominated by $s_j$, then
Algorithm 2: Local Search pseudocode.

```plaintext
input : s
1. s* ← s' ← s;
2. i ← 1;
while i <= m do
   3. j ← 2;
   4. while j <= n_{m_i} do
      5. exchange jobs s'(m_i[j]) and s'(m_i[j - 1])
      6. rebuilds s';
      7. if f(s') < f(s*) then
         8. s* ← s';
      end
   end
   9. s' ← s; j + +;
end
k ← 1;
while k <= n_{flex} do
   10. jflex ← flexJobs[k];
   11. i ← machine that will be allocated the jflex;
   12. destroy the solution s' from right to left until the
       jflex;
   13. insert the jflex in machine i;
   14. j ← index of jflex in machine i;
   15. rebuilds s';
   16. if f(s') < f(s*) then
      17. s* ← s'; s ← s';
   end
   18. while j <= n_{m_i} do
      19. exchange jobs s'(m_i[j]) and s'(m_i[j - 1])
      20. rebuilds s';
      21. if f(s') < f(s*) then
         22. s* ← s';
      end
      23. s' ← s; j + +;
   end
   24. k + +;
end
return s*
```

Algorithm 3: Pseudocode of the selection of the non-dominated solutions.

```plaintext
input : listSol, qtd
1. s* ← s' ← s;
2. i ← 1;
while i <= qtdm do
   3. j ← i + 1;
   4. while j <= qtd do
      5. if dominates(listSol[i], listSol[j]) then
         6. listSol[j] ← listSol[qtd];
         7. qtd ← qtd - 1; j ← j - 1;
      end
      8. else
         9. if dominates(listSol[j], listSol[i]) then
            10. listSol[i] ← listSol[qtd];
            11. qtd ← qtd - 1; j ← i;
         end
      end
   end
   12. j ← i + 1;
end
```

### TABLE I

**Combination of values used in parameter calibration.**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA_{max}</td>
<td>1000</td>
<td>3000</td>
<td>5000</td>
<td>8000</td>
<td>10000</td>
<td>120000</td>
</tr>
<tr>
<td>α</td>
<td>0.95</td>
<td>0.96</td>
<td>0.97</td>
<td>0.975</td>
<td>0.98</td>
<td>0.985</td>
</tr>
<tr>
<td>T_{0}</td>
<td>100</td>
<td>200</td>
<td>400</td>
<td>600</td>
<td>800</td>
<td>1000</td>
</tr>
<tr>
<td>T_{c}</td>
<td>0.001</td>
<td>0.0015</td>
<td>0.0018</td>
<td>0.0019</td>
<td>0.0011</td>
<td>0.0112</td>
</tr>
<tr>
<td>CS</td>
<td>γ</td>
<td>λ</td>
<td>r_{max}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>15</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>25</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>30</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

was set as 180 seconds without the algorithm presenting new improvement in the solution. At the end of the tests, the combination of parameters that obtained the best results was experiment number 3 of Table I.

**B. Computational results of the CS for FJSP**

To evaluate the CS, it was applied in the BRdata [26] instance set, and also to the set of proposed instances. For the BRdata set the CS was executed 5 times for each instance. To analyze the efficiency of the algorithm, in Table II, a comparison is made with the best values obtained by the Simulated Annealing Partial Controlled (SAPC) [14] and the Scatter Search with Path-Relinking (SSPR) [15], considering the classical criterion of minimize the makespan. Table II is described below: The first column lists the names of the instances, followed by their characteristics (number of jobs × number of machines); The other columns list the best values found by each algorithm, followed by the time, in seconds, necessary to find the solution. The best makespan values obtained are indicated in bold.

s_i is removed from the vector. The pseudocode for selecting the non-dominated solutions is described in the Algorithm 3.

**IV. COMPUTATIONAL EXPERIMENTS**

**A. Experimental setup**

The experiments were conducted on a computer with Intel Core i5 750 2.67 GHz processor, 8GB of RAM and operating system Windows 7. The algorithm was implemented in C++ language and compiled with Visual C++ compiler.

The values for the CS and SA parameters were empirically defined and adjusted according to the results obtained in the experiments. The combination of parameters and their respective values are set out in Table I.

Tests were performed considering instances of different complexity: mk01, mk05 and mk10. The stopping criterion
When analyzing the Table II, it can be seen that the metaheuristic CS proposed in this work found the best known solution for six of the ten instances considered. For instances in which CS did not found the best results, it obtained a very close value, which demonstrates the quality of the results. When analyzing the average time to obtain the solutions, it is noticed that the CS proved to be very efficient. However, it is important to emphasize that this is a comparison in which the algorithms were performed in computers with different configurations, so a direct comparison becomes limited.

C. Computational results of the CS for BOFJSP

The CS results for the BOFJSP are subdivided into two sets of instances. The results of the first instance set, which considers the parameter $\beta_1$, are listed in Table III. The results generated from the set of instances that considers the parameter $\beta_2$ are shown in Table IV. Both Tables are presented as follows. The first column indicates the name of the instance; the second column shows the number of non-dominated solutions (#NDS) found; and, the MK and TT columns refer to the respective makespan values and total tardiness of the non-dominated solutions found.

From the Tables III and IV it is verified that the CS obtained a total of 94 non-dominated solutions for the 20 considered instances. It can be seen that the increase in parameter value $\beta_2$ in relation to the parameter $\beta_1$ interfered with the results. The instances generated from the parameter $\beta_1$ obtained better results, that is, 6 non-dominated solutions more (12% better), and lower mean values of the objective functions, being 3% better in the makespan criterion and 19% better in the total tardiness criterion.

The Pareto frontier graph was generated considering the results of the Instance Mk10-$\beta_1$, which is shown in Figure 4. A set of non-dominated solutions was obtained from the set of solutions generated by CS. This perspective allows the decision-maker to more comprehensively assess the impact of each criterion in the solution.

V. CONCLUSIONS

The present study addressed the Flexible Job Shop Scheduling in a Bi-Objective way (BOFJSP). The model used is based on an adaptation of the models proposed by Özgüven et al. [25] and Melo and Ronconi [5], considering two criteria in the objective function, to minimize the makespan and the total tardiness of jobs, simultaneously.

An algorithm based on the metaheuristic Clustering Search (CS) was implemented to solve the BOFJSP. CS is a hybrid algorithm that combines a solutions-generating metaheuristic and a clustering process. The Simulated Annealing metaheuristic was used as solution generator method for the proposed CS.

The results obtained by the CS considering the makespan criterion were competitive when compared to the literature, finding the best solution for 6 of 10 classic problems consid-
TABLE IV

Non-dominated solutions for the proposed set of instances based on $\beta_k$.

<table>
<thead>
<tr>
<th>I</th>
<th>#NDS</th>
<th>MK</th>
<th>TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mk01-β₂</td>
<td>3</td>
<td>40</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43</td>
<td>139</td>
</tr>
<tr>
<td>Mk02-β₂</td>
<td>3</td>
<td>26</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td>Mk03-β₂</td>
<td>3</td>
<td>204</td>
<td>1134</td>
</tr>
<tr>
<td></td>
<td></td>
<td>207</td>
<td>1130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>221</td>
<td>1117</td>
</tr>
<tr>
<td>Mk04-β₂</td>
<td>4</td>
<td>60</td>
<td>604</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61</td>
<td>579</td>
</tr>
<tr>
<td></td>
<td></td>
<td>70</td>
<td>414</td>
</tr>
<tr>
<td></td>
<td></td>
<td>71</td>
<td>378</td>
</tr>
<tr>
<td>Mk05-β₂</td>
<td>8</td>
<td>173</td>
<td>2190</td>
</tr>
<tr>
<td></td>
<td></td>
<td>174</td>
<td>2151</td>
</tr>
<tr>
<td></td>
<td></td>
<td>176</td>
<td>2135</td>
</tr>
<tr>
<td></td>
<td></td>
<td>177</td>
<td>2106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>182</td>
<td>1540</td>
</tr>
<tr>
<td></td>
<td></td>
<td>186</td>
<td>1523</td>
</tr>
<tr>
<td></td>
<td></td>
<td>187</td>
<td>1393</td>
</tr>
<tr>
<td></td>
<td></td>
<td>190</td>
<td>1387</td>
</tr>
<tr>
<td>Mk06-β₂</td>
<td>4</td>
<td>59</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td></td>
<td>63</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td></td>
<td>67</td>
<td>307</td>
</tr>
<tr>
<td></td>
<td></td>
<td>68</td>
<td>283</td>
</tr>
<tr>
<td>Mk07-β₂</td>
<td>6</td>
<td>140</td>
<td>1699</td>
</tr>
<tr>
<td></td>
<td></td>
<td>142</td>
<td>1667</td>
</tr>
<tr>
<td></td>
<td></td>
<td>153</td>
<td>976</td>
</tr>
<tr>
<td></td>
<td></td>
<td>154</td>
<td>884</td>
</tr>
<tr>
<td></td>
<td></td>
<td>156</td>
<td>866</td>
</tr>
<tr>
<td></td>
<td></td>
<td>169</td>
<td>845</td>
</tr>
<tr>
<td>Mk08-β₂</td>
<td>4</td>
<td>523</td>
<td>6066</td>
</tr>
<tr>
<td></td>
<td></td>
<td>524</td>
<td>5917</td>
</tr>
<tr>
<td></td>
<td></td>
<td>527</td>
<td>5894</td>
</tr>
<tr>
<td></td>
<td></td>
<td>528</td>
<td>5468</td>
</tr>
<tr>
<td>Mk09-β₂</td>
<td>4</td>
<td>307</td>
<td>4881</td>
</tr>
<tr>
<td></td>
<td></td>
<td>308</td>
<td>4874</td>
</tr>
<tr>
<td></td>
<td></td>
<td>323</td>
<td>3402</td>
</tr>
<tr>
<td></td>
<td></td>
<td>326</td>
<td>3377</td>
</tr>
<tr>
<td>Mk10-β₂</td>
<td>5</td>
<td>202</td>
<td>2866</td>
</tr>
<tr>
<td></td>
<td></td>
<td>229</td>
<td>2532</td>
</tr>
<tr>
<td></td>
<td></td>
<td>233</td>
<td>2498</td>
</tr>
<tr>
<td></td>
<td></td>
<td>236</td>
<td>2468</td>
</tr>
<tr>
<td></td>
<td></td>
<td>237</td>
<td>2348</td>
</tr>
<tr>
<td>Mean</td>
<td>4.4</td>
<td>186.8</td>
<td>1843.0</td>
</tr>
</tbody>
</table>

Fig. 4. Pareto frontier - instance Mk10-β₂.

ACKNOWLEDGMENT

This research was partly supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq (processes 454569/2014-9 and 301725/2016-0), Fundação de Amparo à Pesquisa e Inovação do Espírito Santo - FAPES (processes 68199325/2014 and 73290475/2016) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES.

REFERENCES


