An Effective Heuristic for a Single-Machine Scheduling Problem with Family Setups and Resource Constraints

Júlio C. S. N. Pinheiro, José E. C. Arroyo, and Ricardo G. Tavares

Department of Computer Science, Universidade Federal de Viçosa,
Viçosa - MG, 36570-900, Brazil
julio.pinheiro@ufv.br, jarroyo@dpi.ufv.br, ricardo.tavares@ufv.br

Abstract. This paper presents a simple and effective iterated greedy heuristic to minimize the total tardiness in a single-machine scheduling problem. In this problem the jobs are classified in families and setup times are required between the processing of two jobs of different families. Each job requires a certain amount of resource that is supplied through upstream processes. The total resource consumed must not exceed the resource supply up. Therefore, jobs may have to wait and the machine has to be idle due to an insufficient availability of the resource.
The iterated greedy heuristic is tested over an extensive computational experience on benchmark of instances from the literature and randomly generated in this work. Results show that the developed heuristic significantly outperforms a state-of-the-art heuristic in terms of solution quality.
Single machine scheduling, family setup-times, resource constraints, total tardiness, meta-heuristics.

1 Introduction

This paper considers a single-machine scheduling problem with the goal of minimizing the total tardiness of the jobs. For each job is given a processing time and a due date, and it belongs to a given family. The machine processes only one job at a time without preemption. A setup time is required between the processing of two jobs of different families and during this time the machine cannot process any other job. A resource constraint is also considered in the problem. Each job requires a certain amount of a common resource provided by the machine, which is supplied by upstream processes. At any time, the cumulative resource consumption must not overcome the cumulative resource supply. Thus, jobs may have to wait and the machine has to be idle due to an insufficient availability of the resource.

This studied problem has practical applications in many industries. For example, it arises in the continuous casting stage of steel production (Herr and Goel, 2016). Ladles with liquid steel of certain grades are fed to a continuous casting machine, and each ladle contains allocated orders to it that determine

a due date. Whenever two ladles of similar steel grades (within the same setup family) are consecutively processed no setup work is needed. Still, if a change of ladles of different steel grades occurs, a setup process is required. The liquid steel is supplied by the blast furnace at a constant rate. At any time of the casting is not allowed to consume more liquid steel than the supplied by the blast furnace.

Similar situations also occur in many multi-stage production processes where upstream work systems supply a common resource that is consumed in downstream tasks executed on a machine, as in assembly lines where different products are assembled by their component parts provided by an upstream stage.

Since this problem has not been thoroughly exploited, our contribution with this work is providing a distinct and effective heuristic method for solving it.

The remainder of this paper is organized as follows. Section 2 gives a literature review of related works. Section 3 describes the problem under study. Our solution approach based on an iterated greedy (IG) algorithm is described in Section 4. Section 5 presents the implementation of a heuristic algorithm proposed in the literature for the same problem. In Section 6, the computational experiments and results are showed. Finally, the conclusions of this paper are presented in Section 7.

2 Literature Review

The problem addressed is this paper comprises three main streams of study in literature. First, single-machine scheduling problems with the goal of minimizing the total tardiness. Second, single-machine scheduling with setup regards. Third, scheduling problems with resource constraints.

Without setup considerations, the single-machine scheduling with total tardiness minimization is proved to be NP-hard (Du and Leung, 1990). Scheduling problems with tardiness minimization are studied by Koulamas (2010) and Sen et al. (2003). For single-machine problems with total tardiness minimization, Emmons (1969) described conditions that when fulfilled would result in an optimal schedule. Lawler (1977) presented a decomposition approach that separates the problem into two mutually exclusive sub-problems using the longest job as a separator.

Surveys of Allahverdi et al. (2008) and Potts and Kovalyov (2000) present the main studies on scheduling problems with setup times considerations. Many meta-heuristics approaches were applied for minimizing the total tardiness on single-machine scheduling problems with sequence-dependent setup times. Gupta and Smith (2006) presented a greedy randomized adaptive search procedure (GRASP) heuristic as well as a space-based local search procedure; Liao and Juan (2007) introduced an ant colony optimization (ACO) based algorithm; Lin and Ying (2008) proposed a simulated annealing and a tabu search algorithm; Ying et al. (2009) developed a local search based iterated greedy heuristic; and Sioud et al. (2012) used a hybrid genetic algorithm. An exact branch and bound algorithm was presented by Bigras et al. (2008). For a variant of the problem where the goal is to minimize the weighted tardiness of the jobs, an exact

method based on successive sublimation dynamic programming and an iterated local search (ILS) heuristic were presented by Tanaka and Araki (2013) and Subramanian et al. (2014), respectively.

Some authors addressed scheduling problems with family setup-times and total tardiness objective. Gupta and Chantaravarapan (2008) studied the problem under consideration of the group technology assumption (GTA) (see e.g. Potts and Van Wassenhove, 1992). They presented a mixed-integer programming (MIP) formulation that solves small problem instances as well as a heuristic algorithm for larger instances. Based on the properties described by Emmons (1969), Schaller (2007) developed two branch and bound procedures for the family scheduling problem with and without GTA. Moreover, Schaller and Gupta (2008) proposed exact and heuristic methods, with and without GTA, for a single-machine scheduling problem with family setups. Furthermore, Jacob and Arroyo (2016) presented iterated local search heuristics for a single-machine scheduling problem with sequence-dependent family setup times to minimize total tardiness.

Briskorn et al. (2010) studied a single-machine scheduling problem where the unavailability of the required resource prevents a job of being processed. They considered the minimizing of the maximum lateness and the number of tardy jobs. Briskorn et al. (2013) studied the single-machine problem with inventory constraints to minimize the total weighted completion times. Based on properties derived for an optimal solution, they presented a branch and bound and a dynamic programming method for solving the problem. They concluded that even for small problem instances with 20 jobs, exact approaches are not efficient for solving the problem, then heuristics methods are required.

The formulation of the problem studied in this paper is given by Herr and Goel (2016). They presented a MIP model as well as an ILS heuristic for solving the problem. Furthermore, they considered two variants of the problem: first, setup times are only considered between jobs of different families; second, setup times are also considered between jobs of the same family. In this paper we address the first variant of the problem.

3 Problem Description

A complete MIP model for problem under study is presented by Herr and Goel (2016). The problem can be described as follows. Let J the set of n jobs to be processed on a single machine. Each job j has a processing time p_j, a due date d_j and a quantity q_j representing the amount of resource required from the machine when the job j is processed. It is assumed that the resource is consumed at a constant rate of q_j/p_j. Also, each job belongs to a setup family f_j. For any pair of jobs $i, j \in J$, if i is scheduled before j and $f_i \neq f_j$, a setup time of duration $s_{ij} > 0$ is required between processing the jobs. Otherwise, $s_{ij} = 0$.

The single machine can process only one job at a time without preemption. Initially the machine has a resource amount r^* and it is supplied at a constant rate r per unit of time. In order to prevent that the cumulative demand of a job

surpasses the cumulative supply of the machine, the machine has to remain idle and the starting of the job has to be delayed. This idle time must be sufficient so the delayed job can be processed without resource constraints. Whether in case of setup time or idle time, the machine cannot process any other job. The goal of the problem is to find a schedule (processing sequence of the jobs) that minimizes the total tardiness.

Following the MIP model of Herr and Goel (2016), we assume the processing of a job may be delayed for any period of time due unavailability of resource. For a job j in a sequence, let Q_j denotes the cumulative demand of resources of all jobs up to j. The tardiness of a job j is computed as follows.

The completion time of the first job i in the sequence is

$$C_i = \max\{p_i, \frac{Q_i - r^*}{r}\} \tag{1}$$

The completion times of the other jobs j is computed as

$$C_j = \max\{C_i + s_{ij} + p_j, \frac{Q_j - r^*}{r}\} \tag{2}$$

where i is the job processed before j. Then, the tardiness of any job j is

$$T_j = \max\{0, C_j - d_j\} \tag{3}$$

Therefore, the total tardiness of all the jobs is $TT = \sum_{j \in J} T_j$

4 Solution Approach

Herr and Goel (2016) showed that solving instances of the problem under study with a large number of jobs using a MIP solver is inefficient in terms of computational time. Even if all the jobs belonged to a same family and the initial amount of resource was sufficiently large the problem would be reduced to the problem studied by Du and Leung (1990), which is proven to be NP-hard. Therefore, the problem studied in this paper is also NP-hard.

In this study we developed an iterated greedy (IG) algorithm to tackle the problem. The IG algorithm was first used in scheduling problems by Ruiz and Stützle (2007) to solve a permutation flow shop scheduling problem, and since then has been applied to others types of scheduling problems (see e.g. Ying et al. 2009, and Fanjul-Peyro and Ruiz, 2010). A typical IG algorithm has few control parameters and is simple to implement.

The general scheme of the proposed IG heuristic is presented in Algorithm 1. Our algorithm has two input parameters: α (destruction level) and $nIter$ (maximum number of consecutive iterations of the algorithm without improvements of the best solution). The algorithm begins generating an initial solution (Step 1). This initial solution is generated by using the earliest due date (EDD) greedy rule. That is, an initial sequence is obtained by sorting the jobs in non decreasing order of their due dates.

Algorithm 1 Iterated Greedy $(\alpha, nIter)$

1: $S_{\text{best}} \leftarrow$ InitialSolution()
2: $S \leftarrow S_{\text{best}}$
3: $k \leftarrow 0$
4: **while** $k < nIter$ **do**
5: $S_{\text{partial}} \leftarrow$ Destruction(S, α)
6: $S \leftarrow$ Reconstruction(S_{partial})
7: $S \leftarrow$ LocalSearch$(S, nIter)$
8: **if** $TT(S) < TT(S_{\text{best}})$ **then**
9: $S_{\text{best}} \leftarrow S$
10: $k \leftarrow 0$
11: **else**
12: $k \leftarrow k + 1$
13: **end if**
14: **end while**
15: **return** S_{best}

The iterations of the algorithm are computed in Steps 4 to 14 until the maximum number of consecutive iterations without improvements is reached. In the destruction phase (Step 5), $\alpha \times n$ jobs are randomly selected and removed from the sequence. In the reconstruction phase, the jobs are greedily reinserted (Step 6). For each job removed, it is tentatively reinserted in all positions of the sequence and then put in the position that gives the lowest total tardiness of the partial solution. Afterwards, a local search (LS) procedure is performed in order to improve the new constructed solution (Step 7). A counter k is used to get the maximum number of consecutive iterations without improvements. If the solution returned by the local search procedure improves the current best solution, the best solution obtained so far is updated (Step 9), and the counter k is set to zero (Step 10). If the best solution is not improved, the counter k is incremented (Step 12). The IG algorithm returns the best solution found (Step 15).

The next subsection provides an explanation of the local search procedure used in the IG algorithm.

4.1 Local Search Procedure

A local search (LS) procedure based on six neighborhood operators is performed to improve the solution given by the reconstruction phase. At each iteration of the LS procedure, a random neighbor solution is generated from the general current solution, then its total tardiness is computed. If neighbor solution improves the general solution, the general solution is updated and a counter variable is reset. Otherwise, the counter variable is incremented. The LS procedure ends when the current solution is not improved during $nIter$ consecutive iterations.

The pseudocode of the LS procedure is given in Algorithm 2. The used neighborhood operators are described in next subsection.

Algorithm 2 Local Search $(S, nIter)$

```
1: k ← 0
2: while k < nIter do
3:    Select at random a neighborhood operator
4:    S_neighbor ← RandomNeighbor(S)
5:    if TT(S_neighbor) < TT(S) then
6:       S ← S_neighbor
7:       k ← 0
8:    else
9:       k ← k + 1
10:   end if
11: end while
12: return S
```

4.2 Neighborhood Operators

The neighborhood operators are algorithmic objects for modifying a solution (schedule or sequence). Each operator modifies a sequence in a specific way. The operators used in our heuristic are the same six operators used by Herr and Goel (2016). Some of those operators regards the fact that in family scheduling problems a subset of jobs within a same family processed consecutively in a sequence can be grouped as a batch. Such batch-based operators have the advantage of maintaining some structural properties of the solution and avoiding unnecessary setup times (Herr and Goel, 2016).

The Batch Move and Job Move operators select a batch and a job, respectively, and move them to another position in the schedule. The Batch Exchange and Job Exchange operators select two different batches and two different jobs, respectively, and swap them. The Batch Break operator selects a batch, sorts it by the the due dates of its jobs and breaks it in the position with the largest due date difference. After that, the two batches are inserted in different positions in the schedule. The Batch Combine operator selects two batches of the same family and bind them, then insert the combined batch in the schedule.

5 Implemented Algorithms from Literature

To the best of our knowledge, there is only one paper that studies the addressed problem. Herr and Goel (2016) proposed an iterated local search (ILS) heuristic to solve the problem. In this work, we re-implemented this ILS heuristic according to the original paper in order to evaluate the efficiency and effectiveness of our IG algorithm.

The ILS implementation can be described as follows. Each neighborhood operator described in Section 4.2 was implemented as a first-improvement local search procedure. An initial solution is generated by using the EDD rule. Then, at each iteration the local search procedures are randomly sorted and consec-

utively applied to the incumbent solution. The stopping criterion of each local search operator is whether a local minimum was found.

The perturbation approach used in the ILS algorithm calculates a promising position for each job in the sequence in terms of total tardiness minimisation (Herr and Goel, 2016). The promising position is calculated based on the lateness of each job. Positive lateness suggests a shifting of the job towards the beginning of the sequence, whereas negative lateness suggests a shifting of the job towards the ending of the sequence. As in Herr and Goel (2016), the maximum number of iterations used in the ILS heuristic was 3.

6 Computational Experiments

In this section we describe the computational experiments carried out in order to study the performance of our IG algorithm. We compare the IG algorithm against the ILS algorithm proposed by Herr and Goel (2016). First, we ensure that the results of our ILS implementation are at least as good as the results presented by Herr and Goel (2016). We also compare several configurations of the IG algorithm with different parameters settings.

All the heuristic algorithms were coded in C++ and executed on a PC machine IntelR Core TM i7-4790K CPU @ 4.00GHz x 8 with 32GB RAM, running Ubuntu 14.04 LTS 64 bits. The quality of the solutions were measured by the relative percentage deviation (RPD) from the best known solution (reference solution). The RPD measure represents the percentage of how much experimental results differs from a reference value, and is calculated following the equation:

$$RPD(\%) = 100 \times \frac{TT_{\text{experimental}} - TT_{\text{reference}}}{TT_{\text{reference}}} \tag{4}$$

where $TT_{\text{reference}}$ is the value of the reference solution, and $TT_{\text{experimental}}$ is the obtained solution by a given heuristic.

The performance of the heuristic algorithms are tested on 80 small instances provided by Herr and Goel (2016), and 30 large instances randomly generated in this paper according to (Herr and Goel, 2016). The factors of the small instances are: number of jobs $n \in \{8, 10, 12, 15, 20, 30, 40, 50\}$ and number of families $\in \{1, 2, 3, 4, 5\}$. The factors of the large instances are: number of jobs $n \in \{100, 150, 200\}$ and number of families $\in \{5, 6, 7\}$.

6.1 ILS Effectiveness

In this subsection we analize the performance of the ILS algorithm of Herr and Goel (2016). We denote by ILS_I the ILS algorithm implemented in this paper, and by ILS_L the ILS algorithm implemented by Herr and Goel (2016).

The ILS_I algorithm was ran on the instances provided by Herr and Goel (2016) and the obtained results are compared with the results of the ILS_L algorithm available in the literature. The performance of each ILS algorithm is

measured by RPD determined by equation (4), where $TT_{\text{reference}}$ is the minimum TT value found for the instance, and $TT_{\text{experimental}}$ is the minimum TT value found by a given ILS algorithm. Therefore, we are able to determine whether ILS_I is able to found results at least as low as the presented in the literature.

The results of the experiment were analyzed by means of Kruskal-Wallis statistical test by using the RPD measure as response variable. In this test two hypothesis are tested: the null hypothesis states that the performance of all the implementations tested are statically similar; and the alternative hypothesis states at least one implementation performance is significantly different. Figure 1 shows the mean plot and confidence intervals at a 95% confidence level. Overlapping intervals suggest that could be no difference between implementations. Since there are no overlapping intervals, the ILS_I presents the lowest mean, and the calculated P-value = $0.000 \leq 0.05$ suggests the null hypothesis can be rejected. We can conclude that the implemented ILS_I is effective enough to represent the implemented one in literature. Table 1 shows how the ILS implementations perform as the number of jobs increases.

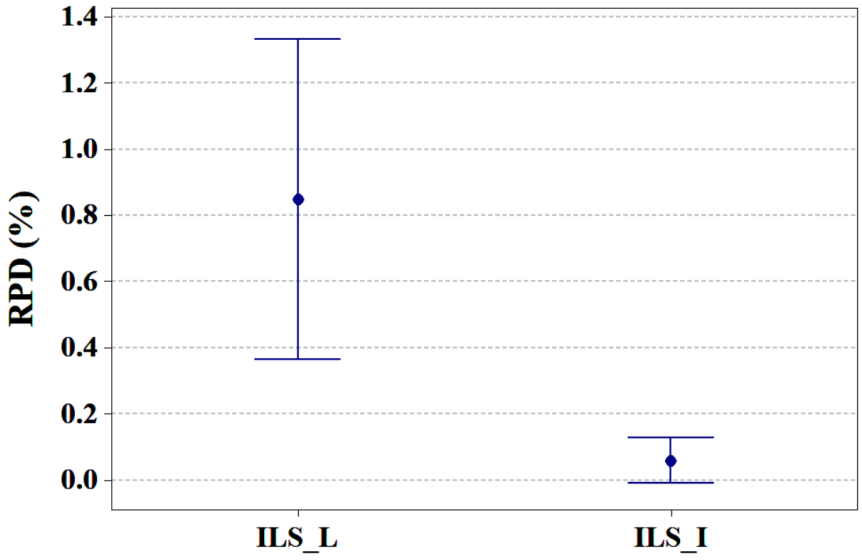

Fig. 1. Mean plot and confidence intervals at 95% confidence level for ILS_L vs. ILS_I experiment.

Table 1. RPD of ILS implementations on literature instances per number of jobs.

	Number of jobs							
Implementation	8	10	12	15	20	30	40	50
ILS_L	0.00	0.00	0.00	0.42	0.24	2.32	0.86	2.95
ILS_I	0.00	0.17	0.00	0.31	0.00	0.00	0.00	0.00

6.2 IG Parameter Setting

The developed IG depends only on two parameters: the number of jobs to be randomly removed in the destruction phase ($\alpha \times n$), and the maximum number of iterations of the algorithm ($nIter$). Preliminary tests were conducted to determine the parameters calibrations. Each parameter was tested at three levels: $\alpha \in \{0.10, 0.15, 0.20\}$ and $nIter \in \{100, 200, 300\}$. We carried out a full factorial experiment with the levels to be tested, resulting in 9 versions of our IG algorithm. Each instance was solved 30 times by all the 9 versions of the algorithm.

The results were analyzed by means of Kruskal-Wallis using the RPD as response variable. Following equation (4), $TT_{\text{experimental}}$ was made as the average TT value found for the instance by a given IG version, and $TT_{\text{reference}}$ was made as smallest TT value found for the instance among all the versions. Fig. 2 shows the mean plot and confidence intervals at a 95% confidence level. We can see that almost every version interval overlaps with one another as the P-value $= 0.085 > 0.05$ from the test indicates that the null hypothesis assumption is the most likely to be correct.

Although no significant difference level is indicated by the test, some conclusions can be derived from Figure 2 about the IG parameters. We can see that as the number of iterations $nIter$ grows the means turn lower, and as the α value increases the intervals turn wider. Thus, we are prone to chose an algorithm version with a large $nIter$. Among the largest $nIter$ values, the IG with $\alpha = 0.15$ and $nIter = 300$ outstands for bearing the lowest RPD. Therefore, this configuration is the chosen one to be used in the next experiments.

6.3 ILS vs. IG

In this section we present the comparison of solution quality generated by the algorithms ILS_I and IG. Two experiments are performed. In the first experiment, the algorithms are compared on the 80 small instances provided by Herr and Goel (2016). In the second experiment, the algorithms are compared on the 30 large instances generated in this paper. The performance of each algorithm is also measured by RPD determined by equation (4), where $TT_{\text{reference}}$ is the best known solution obtained among the two algorithms.

Results on small instances: These experimental results were analyzed by means of Kruskal-Wallis test. The obtained P-value $= 0.034 \leq 0.05$ indicates

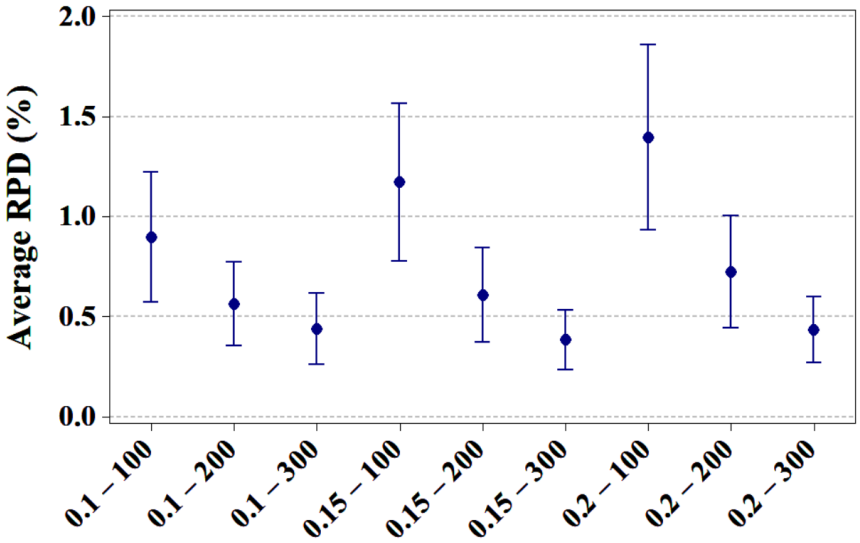

Fig. 2. Mean plot and confidence intervals at 95% confidence level for IG calibration experiment.

that the null hypothesis can be rejected, i.e., one algorithm is significant different. Figure 3 shows, for the two heuristics, the mean plot and confidence intervals with 95% confidence level. Since there are no overlapping intervals and the IG holds the lowest RPD, it is safe to conclude that the IG outperforms the ILS on small instances. Table 2 shows how the heuristic algorithms perform as the number of jobs increases.

Table 2. Average RPD of the heuristics on small instances per number of jobs.

	Number of jobs (n)							
Heuristic	8	10	12	15	20	30	40	50
ILS	0.00	0.17	0.00	0.31	0.48	0.99	1.48	2.36
IG	0.00	0.09	0.00	0.00	0.01	0.00	0.05	0.64

Results on large instances: The results for large instances are presented in Table 3 and Figure 4. Figure 4 shows the means plot and confidence intervals at 95% confidence level. Overlapping intervals indicates that no statistically significant difference exists among the overlapped means, i.e. the algorithms are

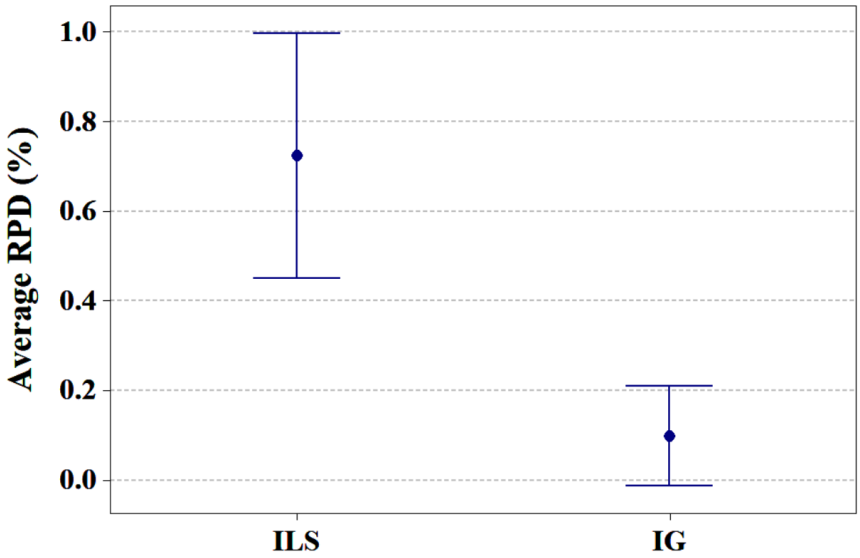

Fig. 3. Mean plot and confidence intervals at 95% confidence level for ILS vs. IG experiment on small instances.

statistically equivalent. Although, submitting the results for testing by means of Kruskal-Wallis test, the calculated P-value $= 0.012 \leq 0.05$ and the IG lowest RPD prones us to conclude that the IG heuristic also outperforms the ILS on large instances in this experiment. Table 3 shows how the heuristics perform as the number of jobs increases.

Table 3. Average RPD of the heuristics on large instances per number of jobs.

	Number of jobs		
Heuristic	100	150	200
ILS	2.61	1.50	1.82
IG	1.31	1.12	1.68

7 Conclusions

In this study we consider a single-machine scheduling problem with family setups and resource constraints with the goal of minimizing the total tardiness. For the best of our knowledge this problem has not been fully exploited in the literature,

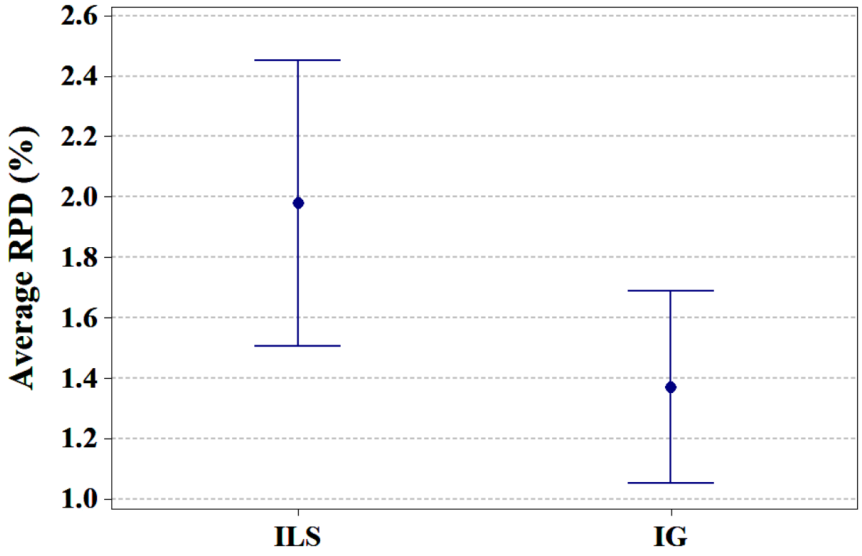

Fig. 4. Mean plot and confidence intervals at 95% confidence level for ILS vs. IG experiment on large instances.

consequently, methods for solving this problem are so far in shortage. Since the problem is NP-hard and exact methods are time inefficient, heuristic methods are required to solve the problem. We presented a simple and effective iterated greedy algorithm which uses a local search procedure based on six neighborhood operators. Two set of instances were used to analyze the performance of the IG algorithm: small instances provided by Herr and Goel (2016), and large instances generated in this paper. Extensive computational experiments have shown that the performance of our IG algorithm increases as the number of iterations increases as well. Computational experiments demonstrated the effectiveness of our IG algorithm. It outperforms the ILS algorithm of Herr and Goel (2016). The obtained results have been statistically tested.

Future research is to extend our approach for other manufacturing environments, such as parallel processing machines.

Acknowledgments. The authors thanks the financial support of FAPEMIG, CAPES and CNPq, Brazilian research agencies.

References

Allahverdi, A., Ng, C., Cheng, T. E., Kovalyov, M. Y.: A survey of scheduling problems with setup times or costs. European journal of operational research, 187(3), 985–1032 (2008)

Bigras, L.-P., Gamache, M., Savard, G.: The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization, 5(4), 685–699 (2008)
Briskorn, D., Choi, B.-C., Lee, K., Leung, J., Pinedo, M.: Complexity of single machine scheduling subject to nonnegative inventory constraints. European Journal of Operational Research, 207(2), 605–619 (2010)
Briskorn, D., Jaehn, F., Pesch, E.: Exact algorithms for inventory constrained scheduling on a single machine. Journal of scheduling, 16(1), 105–115 (2013)
Du, J., Leung, J. Y.-T.: Minimizing total tardiness on one machine is np-hard. Mathematics of operations research, 15(3), 483–495 (1990)
Emmons, H.: One-machine sequencing to minimize certain functions of job tardiness. Operations Research, 17(4), 701–715 (1969)
Fanjul-Peyro, L., Ruiz, R.: Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research, 207(1), 55–69 (2010)
Gupta, J. N., Chantaravarapan, S.: Single machine group scheduling with family setups to minimize total tardiness. International Journal of Production Research, 46(6), 1707–1722 (2008)
Gupta, S. R., Smith, J. S.: Algorithms for single machine total tardiness scheduling with sequence dependent setups. European Journal of Operational Research, 175(2), 722–739 (2006)
Herr, O., Goel, A.: Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. European Journal of Operational Research, 248(1), 123–135 (2016)
Koulamas, C.: The single-machine total tardiness scheduling problem: review and extensions. European Journal of Operational Research, 202(1), 1–7 (2010)
Lawler, E. L.: A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of discrete Mathematics, 1, 331–342 (1977)
Liao, C.-J., Juan, H.-C.: An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research, 34(7), 1899–1909 (2007)
Lin, S., Ying, K.: A hybrid approach for single-machine tardiness problems with sequence-dependent setup times. Journal of the Operational Research Society, 59(8), 1109–1119 (2008)
Potts, C. N., Kovalyov, M. Y.: Scheduling with batching: A review. European journal of operational research, 120(2), 228–249 (2000)
Potts, C. N., Van Wassenhove, L. N.: Integrating scheduling with batching and lotsizing: a review of algorithms and complexity. Journal of the Operational Research Society, 43(5), 395–406 (1992)
Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3), 2033–2049 (2007)
Schaller, J.: Scheduling on a single machine with family setups to minimize total tardiness. International Journal of Production Economics, 105(2), 329–344 (2007)
Schaller, J. E., Gupta, J. N.: Single machine scheduling with family setups to minimize total earliness and tardiness. European Journal of Operational Research, 187(3), 1050–1068 (2008)
Sen, T., Sulek, J. M., Dileepan, P.: Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. International Journal of Production Economics, 83(1), 1–12 (2003)

Sioud, A., Gravel, M., Gagné, C.: A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. Computers & Operations Research, 39(10), 2415–2424 (2012)

Subramanian, A., Battarra, M., Potts, C. N.: An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. International Journal of Production Research, 52(9), 2729–2742 (2014)

Tanaka, S., Araki, M.: An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. Computers & Operations Research, 40(1), 344–352 (2013)

Vilar Jacob, V., Arroyo, J. E. C.: Ils heuristics for the single-machine scheduling problem with sequence-dependent family setup times to minimize total tardiness. Journal of Applied Mathematics, 2016 (2016)

Ying, K.-C., Lin, S.-W., Huang, C.-Y.: Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Systems with Applications, 36(3), 7087–7092 (2009)