Heurísticas para Agendamento de Produção de uma Máquina Linear e Roteamento de Entrega com Frota Heterogênea de Veículos

Gabriel de Paula Felix and José E. C. Arroyo

Department of Computer Science, Universidade Federal de Viçosa,
Viçosa - MG, 36570-900, Brazil
gabriel.felix@ufv.br, jarroyo@dpi.ufv.br

Abstract. Este artigo aborda um problema de programação multiobjetivo do sequenciamento de produção de uma máquina linear integrado ao roteamento de veículos para entrega. Nesta abordagem as tarefas possuem penalidades de atraso, datas de vencimento e são caracterizadas por diferentes volumes, com uma frota heterogênea e limitada de veículos (HFVRP). Os critérios a serem minimizados são referentes ao atraso total ponderado e também aos custos de transporte e entrega, incluindo rotas escolhidas e veículos utilizados.
O problema é classificado como NP-Difícil no senso comum, portanto heurísticas eficientes são necessárias para obter soluções próximas da ótima em tempo computacional razoável. Neste trabalho são propostas duas heurísticas baseadas em Iterated Local Search e um Algoritmo Genético (AG). Instâncias para o problema são geradas com base em trabalhos anteriores e os resultados obtidos são comparados com o solver de otimização CPLEX. Os resultados mostram que as heurísticas aplicadas possuem performance em tempo e qualidade superior às soluções do software.
Keywords: Single machine scheduling, vehicle routing, heterogeneous fleet, time windows, linear programming, meta-heuristics.

1 Introdução

O processo de agendamento de produção e roteamento de veículos é uma tarefa que está presente em manufaturas, e possui grande importância econômica por conta dos prazos impostos, custos de entrega e manuseio do tempo gasto. Separadamente, o problema de agendamento em uma máquina linear com minimização do atraso total é provado como NP-Difícil (Du and Leung, 1990). Estudos são realizados na área de pesquisa operacional para integrar os dois problemas. (Mohammad Tamannaei, Morteza Rasti-Barzoki), estudaram o problema minimizando o atraso total e os custos de transporte utilizando tarefas com tamanhos iguais e custos iguais para todos veículos. Foram propostos três algoritmos genéticos (A.G.) e um método exato Branch-and-Bound (B&B).

Este trabalho ao buscar uma representação realista da integração entre o processo de agendamento de produção de uma máquina linear e roteamento de

veículos, considera tarefas de diferentes volumes e uma frota heterogênea e limitada de veículos, i.e., possuem capacidades e custos variados. (Christian Ullrich, 2013) desenvolveu um Algoritmo Genético (A.G.) integrando agendamento de produção de máquinas paralelas e roteamento de uma frota heterogênea, em um ambiente com janelas de tempo de entrega e veículos com diferentes capacidades.

As aplicações para este problema possuem grande abrangência de setores industriais (como o setor quimico, metalúrgico e têxtil), e a crescente pressão nos mercados globais força as empresas à manusearem seus recursos de forma adequada para garantir um bom funcionamento. Há pouco tempo, Hassanzadeh and RastiBarzoki (2017) estudaram a otimização na cadeia de produção considerando o problema de roteamento de veículos com penalidades de atraso e consumo total de recursos.

Nas últimas décadas, o problema do estudo de roteamento de veículos tornou-se uma questão de foco na literatura acadêmica (Braekers, Ramaekers e Van Nieuwenhuyse, 2016). Na abordagem deste artigo, uma vez que os veículos por possuem diferentes custos e capacidades, apresentam uma concepção desafiadora de utilização dos recursos, tornando cruciais as decisões de quais veículos utilizar. M. Gendreau, G. Laporte and C. Musaragayi (1999), propuseram um algoritmo Tabu Search para o problema de Roteamento de Veículos com Frota Heterogênea (HVRP), e foram obtidas soluções de boa qualidade para as instâncias abordadas. Condotta, Knust, Meier, and Shakhlevich (2013) estudaram o problema de minimizar o atraso máximo de uma máquina linear e considerando datas de entrega para os trabalhos, com veículos com mesma capacidade para entrar as tarefas.

Pelo conhecimento dos autores deste artigo, o problema integrado de sequenciamento em uma máquina simples e roteamento com veículos de capacidades e custos variáveis, tarefas de diferentes tamanhos, e com o objetivo de minimizar o atraso ponderado total e custos de entrega, ainda não foi abordado na literatura. São componentes deste trabalho:

- Duas abordagens heurísticas de Iterated Local Search (ILS) e um Algoritmo Genético (G.A.).
- Um modelo de programação linear para solução dos problemas integrados.

2 Descrição do Problema

Nesta seção é apresentado um modelo matemático de programação linear para solução do problema. As informações sobre parâmetros, índices, variáveis de decisão que serão levados em consideração na resolução deste, são descritos abaixo:

Indices:

i, j	Index of jobs.
k	Index of vehicles.
K	Number of vehicles.
N	Number of jobs.

Instance Parameters:

Q_k	Capacity of each vehicle k.
F_k	Cost of each vehicle k.
P_i	Processing time associated to job i.
t_{ij}	Travel time between customer of job i and customer of job j.
w_i	Penalty applied to delivery tardiness of job i.
s_i	Size of job i.
d_i	Due date associated to job i.

Decision Variables:

C_i	Completion time of job i
X_{ij}^k	Binary decision variable which indicates with value '1' that a vehicle k went from customer of job i to customer of job j, and '0' otherwise.
Y_k	Binary variable which represents with value '1' if a vehicle k is used, and '0' otherwise.
S_k	Time of a vehicle k starts to be used.
A_{ij}	Binary variable which represents if a job i is processed before job j, value '1' if it's true, and '0' otherwise.
D_i	Delivery time of job i.
T_i	Tardiness of job i.

Seja uma máquina linear que lidará com N tarefas, isto é, cada tarefa i de tamanho s_i deverá ser processada em sequência a partir do início do processo ($t = 0$). Um cliente ordena somente uma tarefa. Caso uma tarefa i seja processada antes de uma tarefa j na máquina, a variável A_ij possuirá valor '1', ou '0' caso contrário. O tempo de completude de processamento para tarefa i é representado por C_i. Para a entrega das tarefas, são disponibilizados K veículos, realizando circuitos partindo da origem 'O', sendo a distância entre dois clientes i e j representada por t_ij. Na rota percorrida para entrega, caso uma tarefa i seja entregue imediatamente anterior à uma tarefa j e seja entregue por um veículo k, esta será representada pela variável X_{ij}^k com valor '1', ou '0' caso contrário. A distância de trajeto entre dois clientes i e j é representada por t_{ij}. Este conjunto de variáveis combinadas descreve a primeira parte da função objetivo, que minimiza o a distância de entrega percorrida pelos veículos:

$$\min \sum_{i,j=0}^{N} \sum_{k=0}^{K} X_{ij}^k * t_{ij} \tag{1}$$

Cada veículo k possui capacidade Q_k e custo de utilização F_k, além de possuir tempo de ínicio S_k. Caso um veículo k seja utilizado, será representado em Y_k com o valor '1', ou '0' caso contrário. Dessa forma, a segunda parte da função

objetivo é representada pela minimização dos custos dos veículos:

$$\min \sum_{k=0}^{K} F_k * Y_k \tag{2}$$

Cada tarefa i possui tempo de processamento P_i e data de vencimento d_i vinculada, que pode ser penalizada por uma constante w_i multiplicada pelo seu atraso T_i. O tempo de chegada no destinatário é representado por D_i. A variável de atraso T é dada por $D_i - d_i$, i.e., a diferença entre o tempo de chegada e a data de vencimento. Dessa forma, $T_i = \max(0,\ D_i - d_i)$ e a terceira parte da função objetivo é a minimização do atraso total ponderado das entregas:

$$\min \sum_{i=1}^{N} w_i * T_i \tag{3}$$

Os problemas de minimização descritos acima tratam partes específicas do processo de produção ou entrega de uma manufatura. Finalmente, ao somar os três problemas de minimização, é gerado o problema tratado neste artigo, que representa a minimização dos custos de entrega e seu atraso total ponderado:

$$\min \sum_{i,j=0}^{N} \sum_{k=0}^{K} X_{ij}^{k} * t_{ij} + \sum_{k=0}^{K} F_k * Y_k + \sum_{i=1}^{N} w_i * T_i \tag{4}$$

Quando não tratado o Problema de Minimização de Atraso, resta o Problema de Roteamento com Frota Heterogênea de Veículos (HFVRP), uma versão de VRP, que separadamente é provado como strong NP-Hard (Chen,2010). Por consequência, o problema descrito é também de complexidade strong NP-Hard.

3 Representação de Solução

A representação de uma solução é disposta linearmente, de modo que tarefas e veículos sigam acompanhados de seus respectivos códigos de identificação. O conjunto de tarefas N do problema será dividido em K subconjuntos, sendo cada subconjunto destinado à um veículo específico. O tempo de ínicio de funcionamento de um veículo, S, é dado pela soma dos tempos de processamento P destas tarefas mais o somatório acumulado das tarefas processadas anteriormente (tendo início em $t = 0$). Em cada subconjunto há definição de uma rota, que indica o trajeto percorrido por um veículo. Seja R uma solução viável, $k \in K$ um veículo e S_k sua rota completa, sua representação é dada por:

$$S_i = O \rightarrow C_{j_1} \rightarrow C_{j_2} \rightarrow ... \rightarrow C_{j_{|N_i|}} \rightarrow O \tag{5}$$

Neste contexto, $J \in N$ é o conjunto de tarefas carregadas pelo veículo k e $|N_k|$ a quantidade de tarefas dispostas neste veículo. Todo veículo parte da origem, e deve retornar para ela, visitando cada cliente somente uma vez. Em Figura 1 é apresentada a representação visual de uma solução.

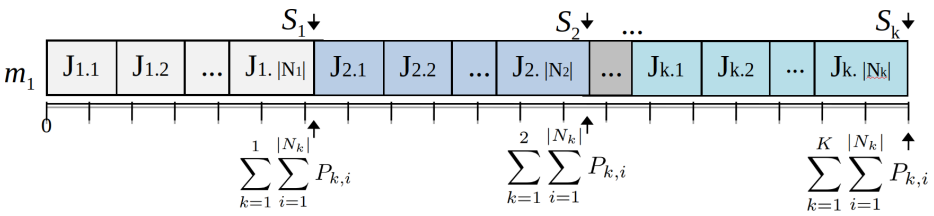

Fig. 1. Representação de solução com tarefas organizadas em lotes.

Um subconjunto de tarefas pode não possuir tarefas e ser designado à um veículo w, indicando que este não está em uso e portanto $|N_w| = 0$. Um caso hipotético de rede de transporte é descrito abaixo, no qual a origem e seis clientes (C_1 à C_6) são dispostos cartesianamente, sendo a distância entre eles representada pelas distâncias euclidianas dos pontos (Tabela 1).

Table 1. Euclidian distance (ED) between two points in the network.

ED(x,y)	Origin	C_1	C_2	C_3	C_4	C_5	C_6
Origin	0	27	260	253	254	112	90
C_1	27	0	265	270	237	114	105
C_2	260	265	0	474	212	371	175
C_3	253	270	474	0	507	178	307
C_4	254	237	212	507	0	346	231
C_5	112	114	371	178	346	0	198
C_6	90	105	175	307	231	198	0

A descrição da Frota de Veículos é descrita na Tabela 2, onde três veículos heterogêneos(V_1, V_2 e V_3) são disponibilizados para transportar as tarefas. Estas informações são fundamentais para a integração do problema de HFVRP durante as entregas, uma vez que os veículos possuem capacidades e custos distintos.

Table 2. Vehicles information: costs and capacities.

Vehicle	Capacity (Q)	Price (F)
V_1	204	1224
V_2	186	1116
V_3	160	960

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

$$C_i = \max\{p_i, \frac{Q_i - r^*}{r}\} \tag{6}$$

The completion times of the other jobs j is computed as

$$C_j = \max\{C_i + s_{ij} + p_j, \frac{Q_j - r^*}{r}\} \tag{7}$$

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

$$T_j = \max\{0, C_j - d_j\} \tag{8}$$

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_best ← InitialSolution()
2: S ← S_best
3: k ← 0
4: while k < nIter do
5:    S_partial ← Destruction(S, α)
6:    S ← Reconstruction(S_partial)
7:    S ← LocalSearch(S, nIter)
8:    if TT(S) < TT(S_best) then
9:       S_best ← S
10:      k ← 0
11:   else
12:      k ← 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 $(S, nIter)$

$k \leftarrow 0$
while $k < nIter$ **do**
  Select at random a neighborhood operator
  $S_{\text{neighbor}} \leftarrow \text{RandomNeighbor}(S)$
  if $TT(S_{\text{neighbor}}) < TT(S)$ **then**
    $S \leftarrow S_{\text{neighbor}}$
    $k \leftarrow 0$
  else
    $k \leftarrow k + 1$
  end if
end while
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 consecutively 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:

$$RPD(\%) = 100 \times \frac{TT_{\text{experimental}} - TT_{\text{reference}}}{TT_{\text{reference}}} \tag{9}$$

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 (9), 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 2 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 3 shows how the ILS implementations perform as the number of jobs increases.

Table 3. 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.

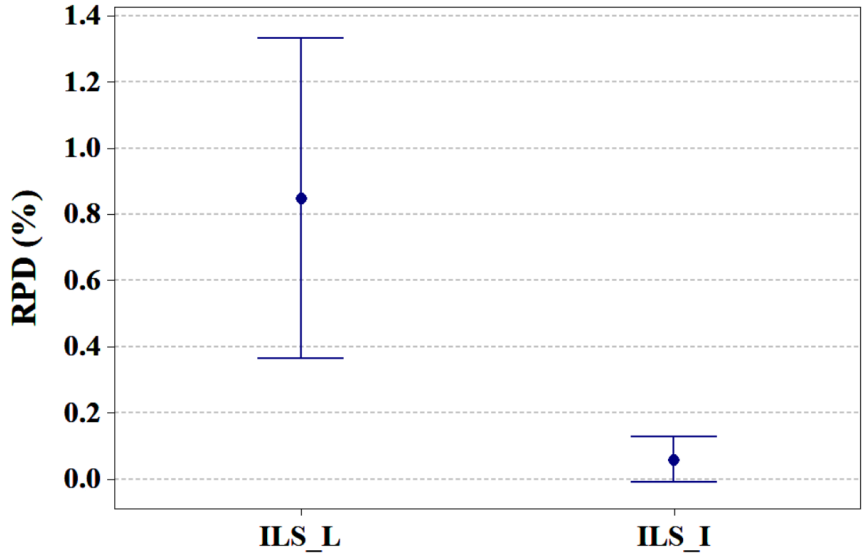

Fig. 2. Mean plot and confidence intervals at 95% confidence level for ILS_L vs. ILS_I experiment.

The results were analyzed by means of Kruskal-Wallis using the RPD as response variable. Following equation (9), $TT_{\text{experimental}}$ 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. 3 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 3 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

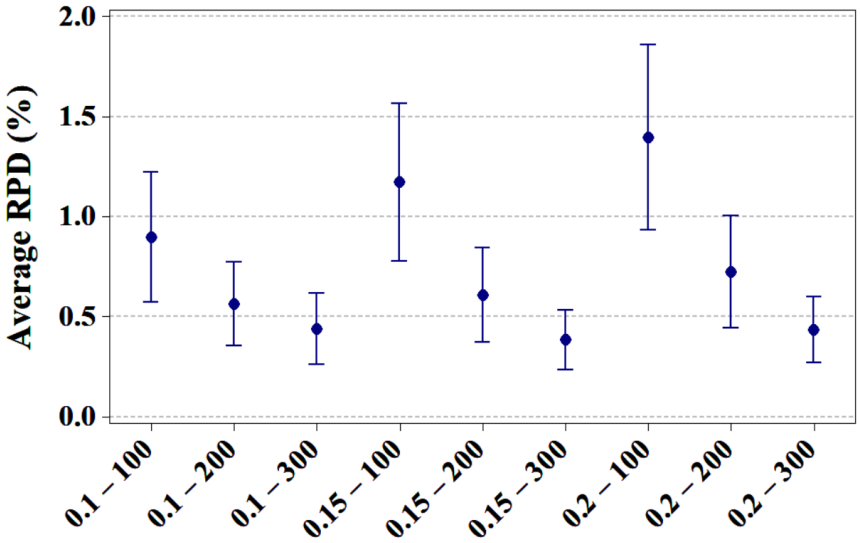

Fig. 3. Mean plot and confidence intervals at 95% confidence level for IG calibration experiment.

30 large instances generated in this paper. The performance of each algorithm is also measured by RPD determined by equation (9), 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 that the null hypothesis can be rejected, i.e., one algorithm is significant different. Figure 4 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 4 shows how the heuristic algorithms perform as the number of jobs increases.

Table 4. 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

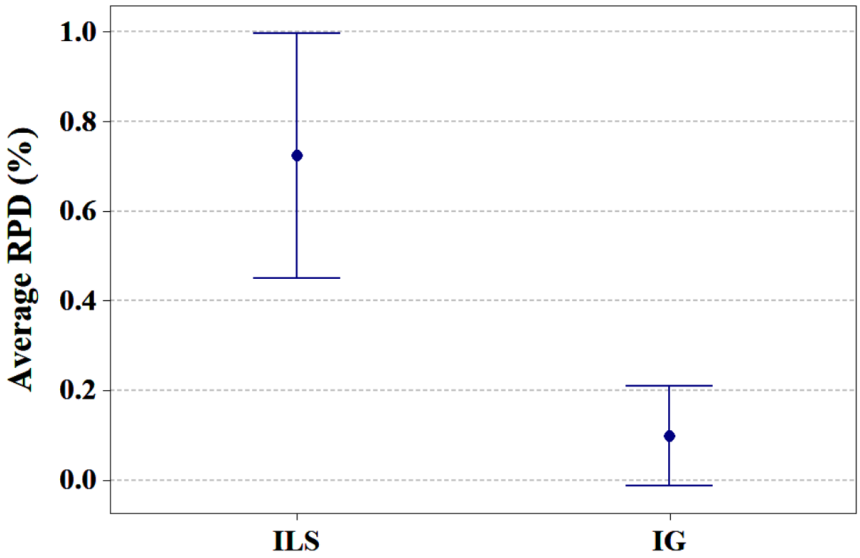

Fig. 4. Mean plot and confidence intervals at 95% confidence level for ILS vs. IG experiment on small instances.

Results on large instances: The results for large instances are presented in Table 5 and Figure 5. Figure 5 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 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 5 shows how the heuristics perform as the number of jobs increases.

Table 5. 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

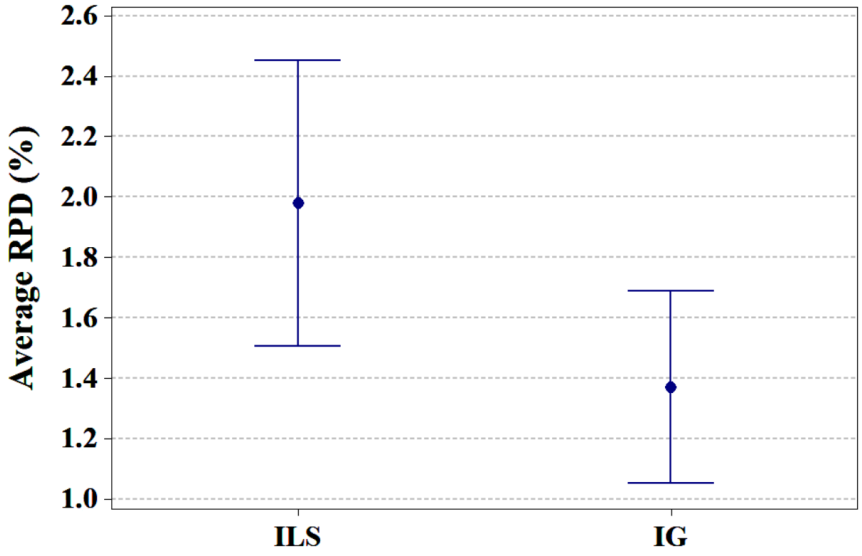

Fig. 5. Mean plot and confidence intervals at 95% confidence level for ILS vs. IG experiment on large instances.

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, 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 lot-sizing: 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)