UMA HEURISTICA ILS-IG PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS NUM AMBIENTE ASSEMBLY FLOWSHOP COM TRÊS ESTÁGIOS E TEMPOS DE SETUP DEPENDENTES DA SEQUÊNCIA

Saulo Cunha Campos

José Elias C. Arroyo

Departamento de Informática, Universidade Federal de Viçosa (UFV)
Viçosa – Minas Gerais – MG – Brasil – 36.570-000
saulo.campos@ufv.br, jarroyo@dpi.ufv.br

RESUMO

Neste trabalho abordamos o problema *Assembly flowshop* com três estágios, onde existem *m* máquinas paralelas no primeiro estágio, uma máquina de transporte no segundo estágio e uma máquina de montagem no terceiro estágio. No primeiro estágio, diferentes partes do produto são fabricadas de forma independente nas máquinas paralelas. No segundo estágio, as peças fabricadas são coletadas e transferidas para o próximo estágio. No terceiro estágio as peças são montadas obtendo o produto final. O objetivo é programar uma sequência de *n* tarefas nas máquinas para minimizar o atraso total. Este problema possui muitas aplicações em indústrias de manufatura e pertence à classe NP-difícil de problemas de otimização combinatória. A fim de obter boas soluções, propomos um algoritmo baseado na aplicação conjunta da metaheurística ILS (*Iterated Local Search*) com a estratégia IG (*Iterated Greedy*). Ambas as técnicas fazem parte do estado da arte para problemas de programação de tarefas *flowshop*. Um estudo comparativo é apresentado onde é analisada a aplicação do ILS e do IG, separadamente, com a sua aplicação conjunta (ILS-IG) e com a heurística GRASP-RVND da literatura. Os resultados experimentais mostram um grande avanço na resolução deste problema, sendo ILS-IG o algoritmo mais eficiente.

PALAVRAS CHAVE. Sequenciamento de tarefas, *flowshop*, Tempos de setup dependentes da sequência, Metaheurísticas, ILS, *Iterated Greedy* (IG) e RLS.

ABSTRACT

In this paper we address a three-stage assembly flowshop scheduling problem where there are *m* machines at the first stage, a transportation machine at the second stage and an assembly machine at the third stage. At the first stage, different parts of a product are manufactured independently on parallel production lines. At the second stage, the manufactured parts are collected and transferred to the next stage. At the third stage, the parts are assembled into final products. The objective is to schedule a sequence of *n* jobs on the machines to minimize the total tardiness. This problem has many applications in manufacturing industries and belongs to the class of NP-hard combinatorial optimization. In order to obtain good solutions, we propose a method based on the joint application of metaheuristic ILS (Iterated Local Search) with IG strategy (Iterated Greedy). Both techniques are part of the state-of -art for flowshop problems of programming tasks. A comparative study is presented comparing the application of ILS and IG, separately, and also their joint application (ILS-IG), with the GRASP-RVND heuristic. The experimental results show great progress in solving this problem, and that the ILS -IG is the more efficient algorithm.

KEYWORDS. *Assembly flowshop scheduling*, *Sequence-dependent setup times*, Metaheuristics, ILS, *Iterated Greedy* (IG) e RLS.

1. Introdução

Os problemas de programação (*scheduling*) são problemas de tomada de decisão comumente encontrados em sistemas de manufatura. Neste tipo de problemas pretende-se encontrar a melhor sequência de operações de manufatura para minimização (em alguns casos, maximização) de um ou mais critérios (objetivos) (SHEN *et al*, 2006).

Estes problemas são amplamente estudados na literatura, sendo que os primeiros trabalhos são remotos da década de 50 (ALLAHVERDI *et al.* 2008). A motivação para investigação deste tipo de problema vem de dois aspectos. O primeiro está relacionado com a importância tática para os negócios: Segundo Pinedo (2005) e Lustosa *et al*. (2008) a programação da produção evolui devido a alta concorrência de mercado e a globalização que pressiona as empresas a reduzirem seus custos. O segundo aspecto está relacionado com o fato da maioria destes problemas pertencerem à classe de problemas NP-Difícil, fazendo com que diversos autores direcionem suas pesquisas para produção de algoritmos de alto desempenho computacional para resolvê-los. Na literatura existem diversos trabalhos de revisões sobre este assunto, como exemplo: Ruiz e Vázquez-Rodríguez (2010) e Ribas *et al.* (2010).

Neste trabalho é abordado um subproblema da programação da produção conhecido como "*Assembly Flowshop Scheduling* – AFS", que é um ambiente de produção onde diversas partes do produto (peças e componentes), são fabricadas de forma independente em diferentes linhas de produção, e, em seguida, são transferidas para um local onde é realizada a montagem do produto, que geralmente é feito em uma única máquina. Este ambiente é encontrado, por exemplo, em indústrias que fabricam placas de circuitos (PINEDO, 2005). Nestas indústrias, diversos componentes são fabricados de forma independente por máquinas paralelas em um primeiro estágio (às vezes, até mesmo em fábricas diferentes), e depois, em um segundo estágio, são agrupados e soldados às placas, formando o produto final. Koulamas e Kyparisis (2001) afirma que este tipo de ambiente de produção vem crescendo em resposta a pressão do mercado sobre as empresas para que ofereçam uma variedade maior de produtos e customizações.

Na literatura, o problema AFS é abordado considerando dois (2sAFS) ou três (3sAFS) estágios. Ambos os casos são NP-Difícil, sendo o último abordado pelo presente trabalho. O 3sAFS consiste em executar (ou produzir) *n* tarefas (ou produtos) em três etapas. Na primeira etapa (estágio 1), diferentes partes de uma tarefa (produto) são processadas em *m* máquinas paralelas independentes. Já na segunda etapa (estágio 2), conhecida como etapa de transporte, as partes produzidas são coletadas e transferidas para uma linha de montagem, que corresponde a terceira etapa (estágio 3), onde o produto final é montado. Tanto na segunda quanto na terceira etapa (estágios 2 e 3), as tarefas são processadas por uma máquina simples.

Já o 2sAFS não possui a etapa de transporte. Dessa forma, imediatamente após a etapa de produção no estágio 1 o produto final é montado em um segundo estágio. Esta é a única diferença entre eles, porém Koulamas e Kyparisis (2001) afirma que o modelo com três estágios é mais realista e o considera como uma evolução do modelo com dois estágios.

Potts *et al*. (1995) foram uns dos primeiros a abordar o problema AFS. Eles provaram que o problema é NP-Difícil e utilizaram métodos heurísticos para minimização do *makespan*. Al-Anzi e Allahverdi (2007) propuseram a minimização do atraso máximo das tarefas, considerando tempos de *setup* (não dependentes da sequência), através da aplicação de diversas metaheurísticas. Seidgar *et al.* (2013) consideraram um algoritmo competitivo imperialista (ICA) para minimizar a função linear ponderada para os seguintes objetivos: *makespan* e tempo médio de fluxo. Yan *et al.* (2014) também abordaram o problema utilizando função linear ponderada, porém com a proposta de minimizar o *makespan* e os custos de atrasos e adiantamento, além disso, eles utilizaram a metaheurística VNS combinado com algoritmos *electromagnetism-like*. Fattahi *et al.* (2013) abordaram uma variação do 2sAFS, onde o ambiente de produção no primeiro estágio é um *flowshop* híbrido. Eles propuseram um modelo matemático e várias heurísticas baseadas no algoritmo de Johnson. Já Mozdgir *et al.* (2012) abordaram o problema considerando múltiplas máquinas não idênticas no estágio de montagem.

O 3sAFS foi abordado pela primeira vez por Koulamas e Kyparisis (2001). Os autores analisaram diversas heurísticas construtivas e propuseram a minimização do *makespan*. Hatami *et*

al (2010) foram os primeiros a tratar o 3sAFS com tempos de *setup* dependentes da sequência das tarefas, porém, aplicado somente às máquinas do primeiro estágio. Eles usaram as metaheurísticas *simulated annealing* e busca tabu para minimizar a função linear dada pela soma ponderada dos objetivos: atraso máximo e fluxo médio das tarefas. Andrés e Hatami (2011) propuseram um modelo matemático MIP para minimização do tempo total de conclusão, considerando tempos de *setup* no primeiro e terceiro estágios. Maleki *et al.* (2012) consideraram o 3sAFS com bloqueio e tempos de *setup* dependentes da sequência das tarefas, mas com o objetivo de minimizar a função linear ponderada com a combinação dos objetivos, tempo médio de conclusão e *makespan*, utilizando uma metaheurística baseada no *simulated annealing*. Já Em 2013, Campos *et al.* (2013) abordaram o mesmo problema considerado por Hatami *et al* (2010) e propuseram uma abordagem baseada na combinação das metaheurísticas GRASP e RVND, obtendo grande melhoria nos resultados finais.

Recentemente têm surgido abordagens multiobjectivos para 3sAFS. Nestas abordagens mais de um objetivo são minimizados simultaneamente. Como por exemplo, os trabalhos de Tajbakhsh et al. (2013) e Campos e Arroyo (2014). O primeiro abordou o problema com a utilização de algoritmo genético e otimização por enxame de partículas (PSO) para minimização do *makespan* e a soma de custos de adiantamentos e atrasos, enquanto o segundo, propõe a utilização do tradicional algoritmo para resolução de problemas multiobjectivo NSGA2 (*non dominated sorting genetic algorithm II*) combinado com o algoritmo IG (iterated greedy) para minimização do tempo total de conclusão e atraso total.

Diferentemente da maioria dos trabalhos da literatura e com o intuito de deixar o problema ainda mais realístico, consideramos tempos de *setup* dependentes da sequência em todos os estágios. A proposta aqui é resolver o problema pela aplicação combinada dos algoritmos ILS – *Iterated Local Search* – (LOURENÇO et al., 2002) e IG – *Iterated Greedy* (RUIZ e STÜTZLE, 2007) – com o intuito de minimizar o atraso total das tarefas.

O ILS é uma metaheurística simples, que a partir de uma solução inicial, aplica buscas locais de forma iterativa, e, para impedir que o algoritmo fique estagnado em ótimos locais, utiliza a estratégia de perturbar a solução corrente (HOOS e STÜTZLE, 2004). Naderi *et al.* (2010) afirma que o ILS tem sido amplamente utilizado para construir soluções do estado-da-arte para problemas de sequenciamento de tarefas em ambientes *flowshop*. O IG, também aplica buscas locais iterativamente, porém para que não fique preso em ótimos locais, usa uma estratégia de destruição-construção da solução. Ruiz e Stützle (2008) aplicaram o IG com grande sucesso para problemas em ambiente *flowshop* com tempos de *setup* dependentes da sequência e demostraram que tal algoritmo, apesar de sua simplicidade, produz resultados de estado-da-arte.

Os detalhes destes algoritmos são vistos nas próximas seções do artigo, que está estruturado na seguinte forma: a seção 2 apresenta a definição formal do problema abordado, a seção 3 apresenta as soluções heurísticas supracitadas, a seção 4 descreve os experimentos computacionais realizados, e, por fim, na seção 5, as conclusões.

2. Definição do problema

O problema investigado neste trabalho é o sequenciamento de tarefas em um ambiente *Assembly Flowshop* com três estágios. Este problema consiste em processar (ou fabricar) um conjunto de n tarefas (produtos), sendo que cada tarefa possui m componentes (ou peças) que são fabricados de forma independente. No primeiro estágio, os componentes das tarefas são fabricados em um ambiente de m máquinas paralelas. No segundo estágio, as peças de uma tarefa são transportadas para uma máquina onde é feita a montagem do produto, constituindo o terceiro estágio da produção. Os estágios 2 e 3 são realizadas por máquinas simples diferentes. Todas as máquinas processam apenas uma tarefa por vez e não podem ser interrompidas durante seu processamento. Observe que para cada tarefa existem m+2 operações distintas, sendo que m operações independentes são feitas no primeiro estágio e as outras duas operações são realizadas nos dois últimos estágios, respectivamente.

Os três estágios caracterizam o problema como um ambiente de produção *flowshop*. Ou seja, uma tarefa j só pode iniciar seu processamento no estágio 2 (transporte) quando todas suas

peças são finalizadas no estágio 1, e a montagem (estágio 3) só poderá ser iniciada após a tarefa *j* ser totalmente finalizada no estágio 2.

O sequenciamento das *n* tarefas nas máquinas é representado por uma permutação simples (sequência), que denota a ordem de execução das tarefas nas máquinas e nos estágios. Todas as tarefas estão disponíveis para serem processadas no tempo zero e nenhuma máquina pode processar mais de uma tarefa por vez.

As notações dos dados de entrada do problema são mostradas a seguir:

- $t_{[k,j]}$ – tempo de processamento da tarefa *j* na máquina *k*, no estágio 1*;*
- $tt_{[j]}$ – tempo de processamento da tarefa *j* na máquina de transporte do estágio 2;
- $ta_{[j]}$ – tempo de processamento da tarefa *j* na máquina de montagem do estágio 3;
- $s1_{[k,j-1,j]}$ – tempo de preparação (*setup*) da máquina *k*, no estágio 1, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1,...*n);*
- $s2_{[j-1,j]}$ – tempo de preparação (*setup*) da máquina de transporte, no estágio 2, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1,...*n);*
- $s3_{[j-1,j]}$ – tempo de preparação (*setup*) da máquina de montagem, no estágio 3, entre duas tarefas consecutivas *j*-1 e *j* (*j*=1,...*n);*
- $d_{[j]}$ – data de entrega da tarefa *j*.

O objetivo do problema é determinar o sequenciamento das tarefas que minimize o atraso total das tarefas (*T*), calculado pela seguinte equação:

$$T = \sum_{j=0}^{n} max\{0, C_{3_{[j]}}\}$$

$C_{3_{[j]}}$, $C_{2_{[j]}}$ e $C_{1_{[j]}}$ representam os tempos de conclusão da tarefa *j*, respectivamente, nos estágios 3, 2 e 1. Eles são calculados pelas seguintes equações:

$$C_{3_{[j]}} = \max\left\{C_{2_{[j]}}, \quad C_{3_{[j-1]}}\right\} + s3_{[j-1,j]} + ta_{[j]}$$

$$C_{2_{[j]}} = max\left\{C_{1_{[j]}}, \quad C_{2_{[j-1]}}\right\} + s2_{[j-1,j]} + tt_{[j]}$$

$$C_{1_{[j]}} = \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} s1_{[k,i-1,i]} + t_{[j,k]}\right\}$$

A Figura 1 mostra um exemplo do problema. O problema em questão possui *n*=5 tarefas e *m*=2 (*M1* e *M2*) máquinas paralelas no estágio 1. As máquinas de transporte e montagem são, respectivamente, *Mt* e *Ma*. Os blocos pretos denotam os tempos de *setup*. As datas de entrega das tarefas são d_1 = 19, d_2 = 15, d_3 = 33, d_4 = 35 e d_5 = 24. Os tempos de processamento e os tempos de *setup* podem ser vistos na Figura 1. A solução em questão é dada pela sequência *S* = {*j2*, *j1*, *j4*, *j3*, *j5*}. A figura 1 também apresenta os tempos de conclusão de cada tarefa e seus atrasos. O valor da função objetivo é *T*=(0+5+0+5+15)=25.

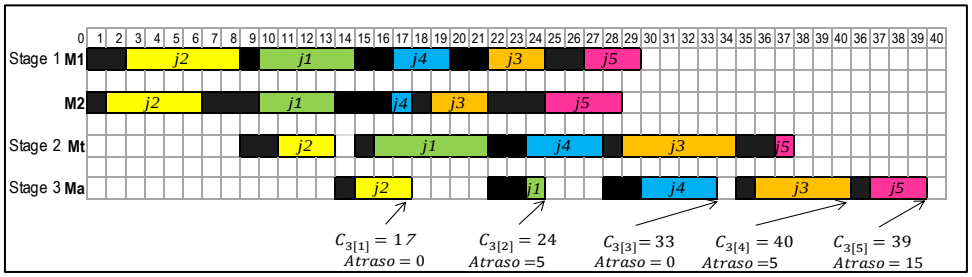

Figura 1: Gráfico de *Gantt* para o sequenciamento das tarefas {j2, j1, j4, j3, j5}

3. Soluções heurísticas

Para problemas da classe NP-Difícil não se conhecem algoritmos de tempo polinomial

para a obtenção de soluções ótimas. Os melhores algoritmos, baseados em métodos exatos, para esses problemas possuem tempo exponencial em função do tamanho da entrada, o que torna inviável a utilização desses algoritmos (LEVITIN, 2003). Como alternativas são utilizadas as metaheurísticas, que são métodos aproximados genéricos para geração de soluções de alta qualidade (próxima da ótima) em tempo computacional polinomial (TALBI, 2009).

Neste trabalho avaliamos o desempenho de três algoritmos baseados em metaheurísticas: ILS, IG e ILS-IG. ILS corresponde a aplicação da metaheurística ILS (*Iterated Local Search*) proposta por Lourenço *et al*. (2002); IG refere-se ao algoritmo proposto por Ruiz e Stützle (2007); e ILS-IG corresponde a aplicação combinada das duas anteriores.

3.1. A metaheurística ILS (*Iterated Local Search*)

Metaheurística simples e robusta, que de acordo com Lopes *et al.* (2013) é formada por quatro operadores: Construção da solução inicial, busca local, perturbação e critério de aceitação.

Assim como a maioria das metaheurísticas, o ILS inicia a partir de uma solução inicial. Após a geração da solução inicial, o ILS aplica iterativamente busca local na solução corrente explorando assim um dado ótimo local. Para que o algoritmo não fique estagnado em apenas um ótimo local, é aplicado o operador de perturbação, que muda a solução corrente a fim de promover a exploração de outra área do espaço de soluções. Em seguida aplica-se o operador de aceitação que define se a solução será considerada para próxima iteração ou não.

O Algoritmo 1 mostra o pseudocódigo do ILS. Consideramos como critério de aceitação, aceitar sempre a melhor solução encontrada ou aceitar uma solução pior de acordo com uma constante de probabilidade. Nas próximas subseções, são apresentadas as características dos métodos de busca local e perturbação.

```
Algoritmo 1: ILS(CritérioParada, n, T, d_max, β)
Entradas:
  n: Número de tarefas do problema;
  T: Tamanho inicial da perturbação;
  d_max: Condição para aumento do tamanho da perturbação;
  β: Probabilidade de aceitar uma solução pior;
Inicio
  S ← GeraSoluçãoInicial( ); //FRB4*
  S ← BuscaLocal(S);
  S* ← S;  numPert ← T;  d ← 0;
  Enquanto ( Critério_Parada não satisfeito ) Faça
    S' ← Perturbação(S, numPert);
    S'' ← BuscaLocal(S', S*);  //RLS
    Se ( f(S'') < f(S*) ) então
      S* ← S'';
      numPert ← T;
      d ← 0;
    Senão
      d ← d + 1;
      Se ( d ≥ d_max ) Então
         numPert ← numPert + T;
         Se (numPert > n) Então
            numPert = T;
         d ← 0;
    Se ( f(S'') < f(S) ) então
      S ← S'';
    Senão
      Se ( rand(0..1) ≤ β ) então
        S ← S'';
  Fim_Enquanto
  Retorna S*;
Fim
```

3.1.1. Geração da Solução Inicial

A geração da solução inicial é fator importante para o sucesso na aplicação de uma metaheurística (TALBI, 2009). Quanto melhor a solução inicial, mais rapidamente a metaheurística tende a convergir para um ótimo local. Por este fato, testamos duas heurísticas

com comprovado histórico de sucesso na resolução de problemas de programação da produção: NEH (NAWAZ et al, 1983) e FRB4* (PAN e RUIZ, 2014).

NEH é uma heurística construtiva gulosa, tradicional da literatura, que funciona inserindo e testando cada tarefa t_j de uma sequência candidata T (t_1, t_2, ..., t_n), ordenada por um dado critério, em todos as posições de uma sequência parcial *S*. A tarefa t_j deve permanecer na posição em que se obtém o melhor resultado para *f(S)*. O procedimento se repete até que todas as tarefas de *T* tenham sido inseridas na sequência *S*, formando assim uma sequência completa.

FRB4* foi proposta por Rad *et al.* (2009) e aperfeiçoada por Pan e Ruiz (2014). A ideia é similar ao NEH, porém após encontrar a melhor posição *p* para inserção de uma dada tarefa corrente t_j na sequência parcial *S*, exploram-se também *y* tarefas antecessoras (s_{j-1}, s_{j-2}, ..., s_{j-y}) e sucessoras (s_{j+1}, s_{j+2}, ..., s_{j+y}) à posição *p* da sequência parcial *S*. Assim, as *y* tarefas antecessoras e sucessoras são reinseridas iterativamente na sequência parcial *S*, testando todas as posições até encontrar a posição que resulta no melhor valor da função objetivo. A rotina segue em repetição até que todas as *y* tarefas antecessoras e sucessoras tenham sido reinseridas na sequencia *S*.

Para que as heurísticas tenham sucesso deve ser definido o critério de ordenação da sequencia candidata *T* e o valor *y* de tarefas adjacentes que serão exploradas pelo FRB4*. Estes valores foram analisados através de experimentos computacionais descritos na seção 4.

3.1.2. Busca Local RLS (*Referenced Local Search*)

O procedimento da busca local RLS (PAN *et al.*, 2008) recebe como entrada uma sequência *S*, correspondente a solução corrente onde deve ser aplicada a busca local, e uma sequência *S**, que é a melhor solução conhecida. Tal procedimento é baseado na remoção e reinserção das tarefas da sequência *S*, de forma iterativa, de acordo com o NEH, até que todas as tarefas tenham sido movimentadas e nenhuma melhora tenha sido gerada. No entanto, a definição sobre qual tarefa será movimentada é feita de acordo com a ordem das tarefas na sequência *S**. Ou seja, a primeira tarefa a ser desconectada e reinserida na sequência *S* é a que se encontra na primeira posição de *S** (*S**[0]), a segunda tarefa será a *S**[1], e assim sucessivamente (*S**[*i*], *S**[*i+1*], ..., *S**[*n*]). O processo de exploração da solução *S* reinicia sempre que uma melhora em *f(S)* é encontrada e o algoritmo continua com a tarefa *S**[*i*+1 mod *n*]. O pseudocódigo do RLS é apresentado no Algoritmo 2.

```
Algoritmo 2: RLS(S, S*, n)
Entradas:
  S: Sequência onde será aplicada a busca local;
  S*: Sequência de referência para escolha das tarefas exploradas em S;
  n: Tamanho da sequência;
Início
  cont ← 0;   i ← 0;
  Faça
    p ← Localizar(S,S*[i]); //retorna a posição de S*[i] em S
    S' ← S - S[p];
    S' ← MelhorPosicao(S',S*[i]); //Insere S*[i] na posição p que resulta no
                                  //menor valor de f(S') e retorna a sequência
    Se ( f(S') < f(S) ) Então
       S ← S';
       cont ← 1; //reinício
    Senão
       cont ← cont + 1;
    i ← (i + 1 mod n);
  Enquanto( cont ≤ n );
  Retorna S;
Fim
```

3.1.3. Perturbação

A perturbação é uma estratégia para fazer com que a busca local explore outras áreas do espaço de solução e não fique presa em apenas um ótimo local. Ela deve ser grande o suficiente para fazer com que a busca local explore outro ótimo local, mas, ao mesmo tempo, não pode ser tão grande ao ponto de perder boas características já encontradas na solução, fazendo com que o ILS se torne um procedimento de busca aleatória.

Neste trabalho, o procedimento de perturbação é aplicado de forma semelhante ao trabalho de Rego *et al*. (2012) e em níveis, como proposto por Jacob *et al*. (2013). Ou seja, após um número d_{max} de iterações sem que haja melhora da solução S^*, o tamanho da perturbação (nível) é aumentado. A ideia é garantir que a busca local faça a exploração apropriada dos vizinhos próximos a solução corrente antes de se deslocar para outro ponto mais distante.

O procedimento de perturbação funciona fazendo a inversão de um bloco de tarefas, com tamanho *numPert*, através da troca de tarefas. A escolha do bloco é aleatória e as tarefas são trocadas de acordo com o nível da perturbação. Por exemplo, para o nível k, o valor de *numPert* é $k \times 2$ (significa que $k \times 2$ tarefas serão movimentadas), então as tarefas das posições $j_1, j_2, ..., j_k$ são trocadas, respectivamente, pelas tarefas $j_{k\times2}, j_{k\times2-1},..., j_{k+1}$. A Figura 2 (a) e (b) ilustram o movimento de perturbação produzido respectivamente para os níveis 2 e 3. Observe pela Figura 2 (b) que, quando a posição de troca é maior que n, o movimento é realizado com as tarefas das primeiras posições.

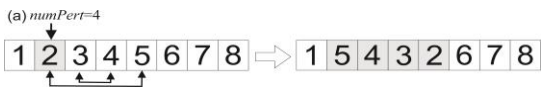

Figura 2 (a): Perturbação de nível 2 (numPert=4) realizada na sequência 12345678 com início na tarefa 2

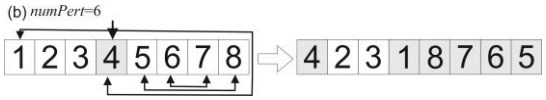

Figura 2 (b): Perturbação de nível 3 (numPert=6) realizada na sequência 12345678 com início na tarefa 4

3.2. IG (*Iterated Greedy*)

O IG é um método simples e eficaz aplicado pela primeira vez por Ruiz e Stützle (2007) e posteriormente repetido por diversos outros autores na literatura. O algoritmo inicia com a construção de uma solução inicial e, em seguida, aplica iterativamente (até que um critério de parada seja satisfeito) seus dois operadores básicos: destruição-construção e busca local.

O operador de destruição-construção (DC) remove k tarefas de uma sequência S, e, em seguida, faz a reinserção de cada uma delas ao gosto da heurística NEH. Nós avaliamos duas formas para o operador de destruição-construção: A primeira, tradicional (DCa), que remove k tarefas aleatórias para reinserção, e, uma segunda (DC*), que remove um bloco de k tarefas adjacentes, ou seja, apenas a posição p ($p \leq n–k$) de início do bloco é definida aleatoriamente.

Após a aplicação do operador DC, a busca local RLS é aplicada e em seguida é definida a aceitação da solução. Em nossa proposta, caso a sequência resultante da busca local (S'') seja melhor que a solução corrente S, o algoritmo seguirá aplicando o operador DC sobre a nova sequência encontrada ($S \leftarrow S''$). Caso contrário, assim como no ILS, consideramos um critério de probabilidade (β) para aceitar soluções piores. O pseudocódigo do IG é mostrado no Algoritmo 3.

```
Algoritmo 3: IG(CritérioParada, K, β)
Entradas
  K: número de tarefas desconectadas da sequência S na rotina DC;
  β: Probabilidade de aceitar uma solução pior;
Inicio
  S ← GeraSoluçãoInicial( ); //FRB4*
  S ← BuscaLocal(S,S); //RLS
  S* ← S;
  Enquanto ( Critério_Parada não satisfeito ) Faça
    S' ← DC(S, k); //Aplica DCa ou DC*;
    S'' ← BuscaLocal(S', S*); //RLS
    Se ( f(S'') < f(S) ) então
      S ← S'';
      Se ( S'' < S* ) Então
        S* ← S'';
    Senão ( rand(0..1) ≤ β ) então
      S ← S'';
  Fim_Enquanto
  Retorna S*;
Fim
```

3.3. ILS-IG

A principal proposta deste trabalho é avaliar a aplicação combinada do ILS com o IG. Assim, trocamos o processo de busca local do ILS pelo IG (DC + busca local). O pseudocódigo

do ILS-IG é apresentado no Algoritmo 4. A ideia central é aplicar o IG iterativamente e, após um número d_{max} de iterações sem melhora na solução corrente, efetuar uma perturbação na melhor solução conhecida até o momento, reiniciando assim o processo do IG.

Algoritmo 4: ILS-IG(*CritérioParada, T, d_{max}, k, β*)

```
Entradas:
  T: Tamanho inicial da perturbação;
  d_max: Condição para aumento do tamanho da perturbação;
  k: número de tarefas desconectadas da sequência S na rotina DC;
  β: Probabilidade de aceitar uma solução pior;
Inicio
  S ← GeraSoluçãoInicial( ); //FRB4*
  S ← BuscaLocal(S, S); //RLS
  S* ← S;  numPert ← T;   d ← 0;
  Enquanto ( Critério_Parada não satisfeito ) Faça
    S' ← DC(S, k); //Aplica DCa ou DC*;
    S'' ← BuscaLocal(S', S*); //RLS
    Se ( f(S'') < f(S*) ) então
      S* ← S'';
      numPert ← T;
      d ← 0;
    Senão
      d ← d +1;
      Se ( d ≥ d_max ) Então
         numPert ← numPert + T;
         Se (numPert > n) Então
            numPert = T;
         d ← 0;
    Se ( f(S'') < f(S) ) então
      S ← S'';
    Senão
      Se ( rand(0..1) ≤ β ) então
        S ← S'';
  Fim_Enquanto
  Retorna S*;
Fim
```

4. Experimentos Computacionais

Nesta seção do trabalho detalhamos os experimentos computacionais realizados. Os algoritmos foram implementados em linguagem C++ e todos os experimentos foram realizados em um computador com processador Intel Core 2 Quad Q6600, de 2,40 GHz, com memória RAM de 4.0 GB e sistema operacional Windows 7 32 bits. A metodologia dos experimentos e análise dos dados foi inspirada no Desenho de Experimentos (DOE) (MONTGOMERY, 2006).

4.1. Geração de instâncias para o problema

Para realização dos experimentos foram geradas instâncias do problema de tamanhos variados, de acordo com Liefooghe *et al*. (2012). A variação da quantidade de tarefas é n = {30, 50, 80, 100, 200} e quantidade de máquinas paralelas no primeiro estágio é m = {5, 10, 20}, totalizando 15 combinações diferentes. Para cada combinação $n \times m$ foram gerados 10 diferentes instâncias aleatórias, totalizando assim 150 problemas. Os tempos de processamento, as datas de entrega e os tempos de *setup* são gerados aleatoriamente de acordo com as respectivas distribuições uniforme: [0, 99], [0, 49] e [3 × p, (n+2) × p], onde p corresponde a média do tempo de processamento previamente gerado anteriormente.

4.2. Critério de parada dos algoritmos e Medida de desempenho

O critério de parada de todos os algoritmos é o tempo de execução, onde o tempo máximo é calculado pela seguinte fórmula: $n \times m \times c$, sendo c uma constante de tempo.

A métrica utilizada para avaliar os resultados dos experimentos é o Desvio Percentual Relativo que é denotado por RPD e obtido pela seguinte fórmula: RPD = $\frac{f_{method} - f_{best}}{f_{best}} \times 100$, onde f_{method} corresponde ao valor da função objetivo obtido por um dado algoritmo e f_{best} correspondente à melhor solução obtida a partir da execução de todos os algoritmos comparados.

4.4. Calibração dos algoritmos

Os experimentos para calibração têm por objetivo extrair as configurações que resultam no melhor desempenho. Neste caso, utilizamos c=50 e 75 das 150 instâncias geradas. Sobre cada instância, os algoritmos foram rodados 10 vezes totalizando 750 execuções cada. Devido à alta gama de parâmetros, não testamos todas as combinações possíveis e fixamos os valores dos parâmetros que não estavam sendo avaliados no momento, pois para calibrar todas as combinações de parâmetros dos três algoritmos propostos seriam necessárias 1488 horas de processamento. A análise dos dados, em todos os experimentos deste tipo, foi feita usando o teste de Tukey da Diferença Honestamente Significativa HSD (*Honestly Signicant Dierence*) com nível de confiança de 95%.

Construção da solução inicial

Duas heurísticas diferentes foram avaliadas para construção da solução inicial: NEH e FRB4*. Para FRB4* foi utilizado o valor y=3, como explicado na seção 3.1. Antes da aplicação das heurísticas as tarefas foram ordenadas de acordo com as seguintes regras de despacho:

- EDD – *Earliest Due Date*: Ordena as tarefas em ordem crescente das suas datas de entrega.
- TLB – T*ardiness Lower Bound*: Ordena as tarefas em ordem decrescente do limitante inferior do atraso total: $d_{[j]} - \{ (\max_{k=1,\dots,m} t_{[j,k]}) + tt_{[j]} + ta_{[j]} \}, \forall_{j=1,\dots,n}$.

A heurística FRB4*-EDD obteve o melhor desempenho em relação às outras, como pode ser visto na Figura 3.

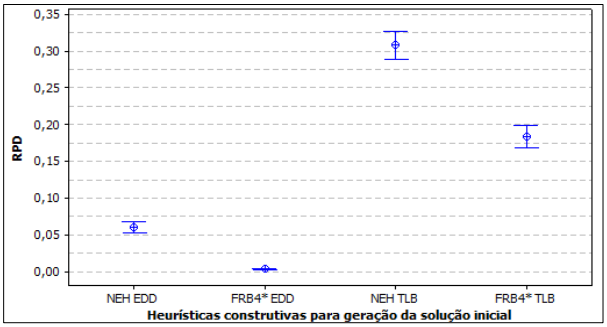

Figura 3: Gráfico das médias e intervalos HSD de Tukey com nível de confiança de 95% para geração da solução inicial.

Calibração do ILS.

Os parâmetros do ILS (T, d_{max} e β) foram analisados separadamente com os seguintes conjuntos de valores: $T \in \{2, 4\}$, $d_{max} \in \{15, 20, 25\}$ e $\beta \in \{0, 0.3, 0.6, 1\}$. Os resultados podem ser vistos através das Figuras 4, 5 e 6. Os melhores valores foram T=2, d_{max}=20 e β=0,3.

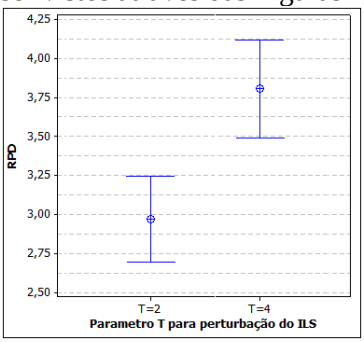

Figura 4: Resultado da calibração do parâmetro T do ILS

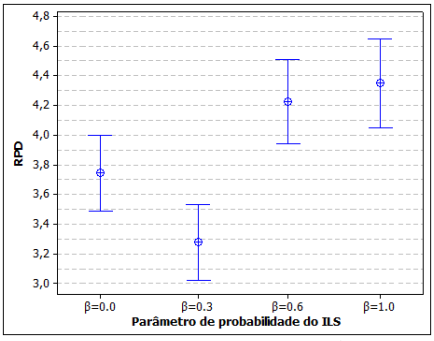

Figura 5: Resultado da calibração do parâmetro β do ILS

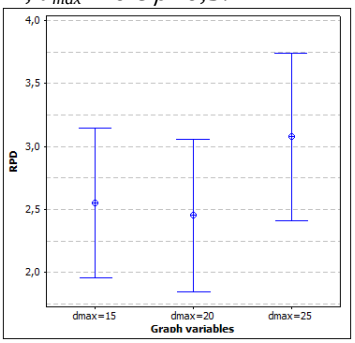

Figura 6: Resultado da calibração do parâmetro d_{max} do ILS

Calibração do IG

Os parâmetros do IG são: K e β. Para eles foram testados, respectivamente, os seguintes conjuntos de dados: $k \in \{4, 7, 10, 13\}$ e $\beta \in \{0, 0.3, 0.6, 1\}$. Como podem ser vistos nas Figuras 7 e 8 os melhores valores foram T =4, β =0.3. Além disso, testamos também o procedimento de destruição-construção, comparando sua abordagem tradicional, que desconecta tarefas

aleatoriamente (DCa), com uma abordagem de desconexão de tarefas por bloco (DC*). Através da Figura 9 é possível perceber que não há diferenças estatísticas significantes entre as abordagens, porém na média esta última é mais eficiente.

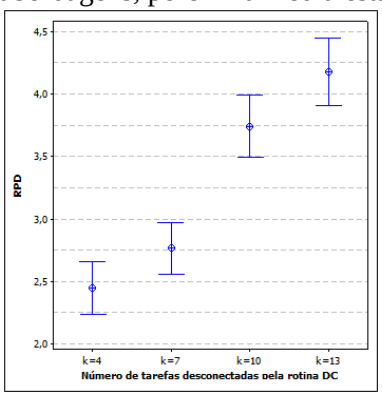

Figura 7: Resultado da calibração do parâmetro *k* do IG

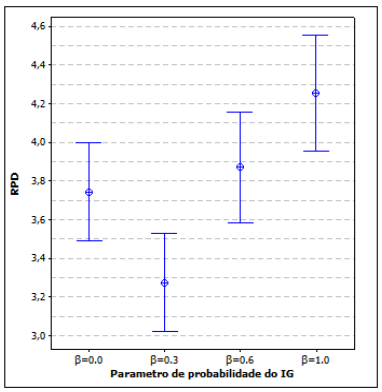

Figura 8: Resultado da calibração do parâmetro *β* do IG

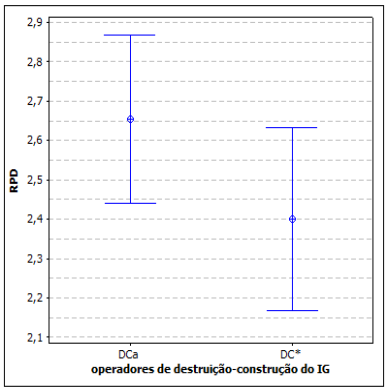

Figura 9: Comparação das estratégias de destruição do procedimento de DC do IG

Calibração do ILS-IG

Para os parâmetros do ILS-IG, optamos por utilizar os mesmos valores encontrados anteriormente para o ILS e para o IG.

4.5. Comparação dos algoritmos propostos

Neste experimento comparativo analisamos o desempenho dos algoritmos ILS, IG e ILS-IG juntamente com a heurística GRASP-RVND (CAMPOS *et al.*, 2013), que foi desenvolvida para resolução deste mesmo problema.

Neste experimento foi considerado c=100 e todas as 150 instâncias geradas. Para cada instância, os algoritmos foram rodados 10 vezes, totalizando 1500 execuções cada. Para análise dos dados, em primeiro lugar, verificamos as principais premissas de ANOVA (normalidade, homocedasticidade e independência) a fim de aplicá-la, porém o teste de homocedasticidade não foi satisfeito. Sendo assim, realizamos o teste não paramétrico de Kruskal-Wallis e apresentamos os resultados através do gráfico de médias resultante do teste Tukey da Diferença Honestamente Significativa HSD (*Honestly Signicant Dierence*) com nível de confiança de 95%.

Tabela 1: RPD médio dos algoritmos GRASP-RVND, ILS, IG e ILS-IG

n	*m*	GRASP-RVND	ILS	IG	ILS-IG
30	5	2,10	1,15	**0,50**	0,58
30	10	2,13	0,96	**0,41**	0,48
30	20	1,90	0,67	0,34	**0,30**
50	5	7,54	3,40	2,43	**2,05**
50	10	6,63	2,83	1,83	**1,67**
50	20	6,29	3,34	1,99	**1,89**
80	5	15,70	5,62	3,90	**3,23**
80	10	13,64	4,83	3,91	**3,58**
80	20	9,83	3,80	2,44	**2,20**
100	5	16,68	4,76	4,51	**3,15**
100	10	16,59	4,64	3,62	**2,95**
100	20	11,25	3,45	2,55	**2,12**
200	5	24,36	**3,33**	6,33	5,77
200	10	21,98	**3,02**	5,47	5,14
200	20	18,57	**2,43**	4,11	4,12
Média		**4, 06**	**1,80**	**1, 10**	**1,01**

Tabela 2: Teste de múltiplas comparações pelo RPD

Comparação	Sig.	Diferença	+/- Limites
GRASP-RVND ↔ IG	*	9,34063	0,368146
GRASP-RVND ↔ ILS	*	9,09681	0,368146
GRASP-RVND ↔ ILS-IG	*	9,70755	0,368146
IG ↔ ILS		-0,243816	0,368146
IG ↔ ILS-IG		0,366923	0,368146
ILS ↔ ILS-IG	*	0,610739	0,368146

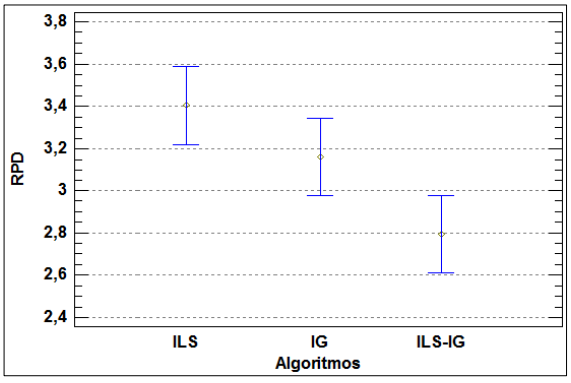

Figura 10: Gráfico de médias e intervalos HSD de Tukey

Os resultados obtidos são mostrados na Tabela 1 com base no RPD. É possível ver que para maioria das instâncias, o ILS-IG alcança os melhores resultados, enquanto o IG é melhor

para instâncias menores (n=30) e o ILS foi melhor para as maiores instâncias (n=200).

O resultado do teste não paramétrico de Kruskal-Wallis com 95% de confiança resultou em um valor *P-Value* = 0, o que mostra que existe diferenças estatísticas significativas entre os algoritmos. A Tabela 2 mostra o teste de múltiplas comparações onde é possível ver os pares de algoritmos comparados, a diferença média entre amostras da comparação, o intervalo de incerteza e o indicativo sobre se há diferenças estatísticas significativas (coluna 2). Este teste deixa claro que todos os algoritmos aqui propostos são superiores ao GRASP-RVND. O teste também mostra que há diferença significativa entre o ILS-IG e o ILS e que não há diferenças significativas entre o IG e o ILS e nem entre o IG e o ILS-IG. Entretanto, a Figura 10 mostra através do gráfico das médias e intervalos HSD de Tukey com nível de confiança de 95%, que apesar do ILS-IG não possuir diferença estatística para o IG, em média, seus resultados são melhores.

5. Conclusão

Este artigo propôs um algoritmo híbrido, que combina as técnicas ILS e IG, para o problema do *assembly floshop* com três estágios, com tempos de *setup* dependentes da sequência, para minimização do atraso total das tarefas. Foram realizados diversos experimentos de calibração e análise estatística para comparação dos algoritmos. No estudo comparativo realizado consideramos problemas de tamanho médio e grande, e, de acordo com os resultados, os algoritmos aqui propostos são superiores ao GRASP-RVND. Os resultados também mostram que o ILS-IG possui, em média, resultados superiores ao IG e ao ILS. Dessa forma podemos concluir que o ILS-IG obteve excelente desempenho para resolução deste problema.

Agradecimentos: Os autores agradecem ao CNPq, FAPEMIG e à CAPES pelo apoio financeiro ao desenvolvimento e publicação deste trabalho.

Referências

Al-Anzi, F. S. e Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, vol. 182 no.1, 80-94.

Andrés, C. e Hatami, S. (2011). The three stage assembly permutation flowshop scheduling problem. *5th International Conference on Industrial Engineering and Industrial Management (Cartagena, Colombia)*, 867-875.

Campos, S. C., Arroyo, J. E. C. e Gonçalves, L. B. (2013). Uma heurística GRASP-VND para o problema de sequenciamento de tarefas num ambiente assembly flowshop com três estágios e tempos de setup dependentes da sequência. *Anais do XLV Simpósio Brasileiro de Pesquisa Operacional (Natal-RN, Brasil).*

Campos, S. C., Arroyo, J. E. C. (2014). NSGA-II with Iterated Greedy for a bi-objective three-stage assembly flowshop scheduling problem. *In Proceeding of the nineteenth annual conference companion on Genetic and evolutionary computation conference companion (GECCO)*. ACM, 429-436.

Fattahi, P., Hosseini, S. M. H. e Jolai, F. (2013). A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, vol. 65 no.5-8, 787-802.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R. e Maboudian, Y. (2010). Two meta-heuristics for three-stage assembly flowshop scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, vol.50 no.9-12, 1153–1164.

Jacob, V. V., Arroyo, J. E. C., dos Santos, A. G. (2013) Heurística busca local iterada para o sequenciamento de tarefas em uma maquina com tempos de preparação dependentes da família. *Anais do XLV Simpósio Brasileiro de Pesquisa Operacional (Natal-RN, Brasil).*

Koulamas, C., Kyparisis, G. (2001). The three-stage assembly flowshop scheduling problem. *Computer & Operation Research*, vol. 28, 689–704.

Levitin, A. (2003). Introduction to the design & analysis of algorithms. *Addison-Wesley Reading*.

Liefooghe, A., Basseur, M., Humeau, J., Jourdan, L. e Talbi, E. G. (2012). On optimizing a bi-objective flowshop scheduling problem in an uncertain environment. *Computers &*

Mathematics with Applications, vol.64 no.12, 3747-3762.

Lopes, H. S., Rodrigues, L. C. A. e Steiner, M. T. A. (2013). *Meta-heurísticas em pesquisa operacional*, Curitiba, PR: Omnipax.

Lourenço, H. R., Martin, O. e Stutzle T. (2002) Iterated local search. *In Handbook of Metaheuristics. Vol. 57 of Operations Research and Management Science*. Kluwer Academic Publishers, 2002, 321–353)

Maleki, D. A., Modiri, M., Tavakkoli, M. R. e Seyyedi, I. (2013). A Three-Stage Assembly Flow Shop Scheduling Problem With Blocking And Sequence-Dependent Set Up Times. *Journal of Industrial Engineering International*, vol. 8 no.1, 1-7.

Montgomery D. C. (2006) Design and analysis of experiments (7th ed.). *New York: Wiley*.

Mozdgir, A., Fatemi Ghomi, S. M. T., Jolai, F. e Navaei, J. (2013). Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. *International Journal of Production Research*, vol. 51 no.12, 3625-3642.

Naderi, B., Ruiz, R. e Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. *Computers & Operations Research*, vol. 37 no.2, 236-246.

Nawaz, M., Enscore, E. e Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, vol. 11 no.1, 91–95.

Pan, Q. K. e Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, vol. 44, 41-50.

Pan, Q. K., Tasgetiren, M. F. e Liang, Y. C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, vol.55 no.4, 795-816.

Potts, C. N., Sevast'janov, V. A., Strusevich, V. A., Van Wassenhove, L. N. e E Zwaneveld, C. M. (1995) The Two-Stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research*, vol. 43 no. 2, 346-355.

Rad S. F., Ruiz R. e Boroojerdian, N. (2009) New high performing heuristics for minimizing makespan in permutation flowshops. *OMEGA, The International Journal of Management Science*, vol. 37 no 2, 331–345.

Rego, M. F., Souza, M. J. F., Arroyo, J. E. C. (2012). Algoritmos multiobjetivos para o problema de sequenciamento de famílias de tarefas em uma máquina. *La Conferencia Latinoamericana en Informática (CLEI)*, Medellín-Colômbia.

Ribas, I., Leisten, R. e Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, vol. 37 no. 8, 1439-1454.

Ruiz, R. e Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, vol. 177 no.3, 2033-2049.

Ruiz, R. e Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, vol.187 no.3, 1143-1159.

Seidgar, H., Kiani, M., Abedi, M. e Fazlollahtabar, H. (2013). An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. *International Journal of Production Research*, (ahead-of-print), 1-17.

Shen, W., Wang, L., e Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews. *IEEE Transactions on*, vol. 36 no.4, 563–577.

Tajbakhsh, Z., Fattahi, P. e Behnamian, J. (2013). Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm. *Journal of the Operational Research Society*.

Talbi, E-G. (2009). Metaheuristics: from design to implementation. *Jonh Wiley and Sons Inc*.

Yan, H. S., Wan, X. Q. e Xiong, F. L. (2014). A hybrid electromagnetism-like algorithm for two-stage assembly flow shop scheduling problem. *International Journal of Production Research*, (ahead-of-print), 1-14.