Sequenciamento de tarefas em um ambiente de produção assembly flowshop: modelagem e resolução heurística

Ricardo Gonçalves Tavares
Departamento de Informática, Universidade Federal de Viçosa (UFV)
Viçosa – Minas Gerais – MG – Brasil – 36.570-000
ricardo.tavares@ufv.br

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

RESUMO

Neste artigo é estudado um problema de programação *flowshop* com dois estágios. No primeiro estágio *m* componentes de um produto (tarefa) são fabricados independentemente utilizando *m* máquinas paralelas. No segundo estágio as componentes são montadas obtendo o produto final. O objetivo do problema é determinar o sequenciamento das tarefas de forma a minimizar o atraso total. Para resolver o problema, que é NP-Difícil, são propostos um modelo de programação inteira mista e quatro heurísticas baseadas nas meta-heurísticas *Iterated Greedy* (IG) e *Iterated Local Search* (ILS). Duas das heurísticas são métodos híbridos obtidos combinando-se IG e ILS com a heurística *Variable Neighborhood Descent*, respectivamente. Para instâncias pequenas e médias do problema, os desempenhos das heurísticas propostas são avaliados através da comparação com as soluções geradas pelo modelo. Para instâncias de grande porte, as heurísticas são comparadas entre elas. Experimentos computacionais mostram o bom desempenho das heurísticas híbridas. Os resultados obtidos são validados através de um teste estatístico.

PALAVARAS CHAVE. Sequenciamento, assembly flowshop, meta-heurísticas, otimização.

ABSTRACT

This paper studies a flowshop scheduling problem with two stages. At the first stage, different parts of a product (job) are manufactured independently on *m* parallel machines. At the second stage the manufactured parts are assembled into a final product. The goal of the problem is determine the job schedule in order to minimize the total tardiness. To solve this NP-hard problem, we propose a mixed integer programming model and four heuristics based on *Iterated Greedy* (IG) and *Iterated Local Search* (ILS). Two heuristics are hybrid algorithms. They are developed by combining IG and ILS with *Variable Neighborhood Descent* heuristic, respectively. For small and medium instances, the performance of the heuristics are compared with solutions obtained by the model. For large instances, the heuristics are compared between them. The computational experiments show the good performance of the hybrid heuristics. The results are validated by a statistical test.

KEYWORDS. Scheduling, Assembly flowshop, meta-heuristics, optimization.

1. Introdução

Neste trabalho é estudado o problema conhecido como *Assembly Flowshop Scheduling* com dois estágios (2sAFS). Neste problema existem *n* produtos (tarefas) a serem fabricados em dois estágios. Cada produto é constituído de *m* componentes que são fabricados independentemente por *m* máquinas paralelas, respectivamente. A fabricação dos componentes corresponde ao primeiro estágio. No segundo estágio, os componentes fabricados são montados por uma máquina obtendo o produto final. O objetivo do problema é determinar o sequenciamento das tarefas de tal maneira que o atraso total das tarefas, com relação aos prazos de entrega, seja minimizado. Quando $m = 1$ (uma única máquina no primeiro estágio), o problema 2sAFS é reduzido ao problema *flowshop* permutacional com duas máquinas, que é considerado NP-Difícil quando for minimizado o atraso total(Du e Leung, 1990). O problema 2sAFS tem sido bastante abordado na literatura considerando a minimização do *makespan* (máximo tempo finalização de todas as tarefas). Há muitos problemas práticos que podem ser modelados como o problema 2sAFS, e devido a suas aplicações e importância teórica, o problema 2sAFS tem recebido, recentemente, muita atenção na literatura sobre *scheduling* (Allahverdi e Aydilek, 2014).

Os primeiros trabalhos sobre o problema 2sAFS foram apresentados por Lee et al. (1993), Potts et al. (1995) e Gupta et al. (1997). Todos estes autores consideram a minimização do *makespan*. Allahverdi e Al-Anzi (2006a) estudam o problema 2sAFS considerado tempos de preparação separados do tempo de processamento. Para minimizar o *makespan*, eles propõem duas heurísticas baseadas, respectivamente, nas meta-heurísticas *Particle Swarm Optimization* (PSO) e Busca Tabu (BT). Allahverdi e Al-Anzi (2006b) abordaram o problema 2sAFS considerando a minimização do máximo atraso das tarefas (*maximum lateness*). Eles aplicam as heurísticas PSO e BT para resolver o problema. Vários trabalhos sobre aplicações de meta-heurísticas foram desenvolvidos para minimizar a soma ponderada do *makespan* e a média dos tempos de conclusão das tarefas (Al-Anzi e Allahverdi, 2009; Seidgar et al., 2014). Dentre essas meta-heurísticas destancam-se *Simulated Annealing* (SA), *Ant Colony Optimization* (ACO), *Variable Neighbourhood Search* (VNS), PSO e BT. Mozdgir et al. (2013) abordam o problema 2sAFS com tempos de preparação dependentes da sequência no primeiro estágio, também consideram a minimização da soma ponderada do *makespan* e a média dos tempos de conclusão e propõem uma heurística híbrida baseada na meta-heurística *Variable Neighbourhood Search* (VNS). Navaei et al. (2013) estudam o problema 2sAFS com múltiplas máquinas de montagem. Para a minimização do *makespan* eles propõem um modelo de programação inteira e um heurística hibrida baseado na meta-heurística SA.

O atraso total das tarefas é uma medida de desempenho muito importante na programação da produção. Em muitas situações, os atrasos das tarefas podem levar a custos de punição ou cancelamento de pedidos pelos clientes. Allahverdi e Aydilek (2014) são os primeiros autores que consideram a minimização do atraso total no problema 2sAFS. Estes autores propõem heurísticas baseadas em Algoritmos Genéticos e SA.

Neste trabalho propõe-se um modelo de Programação Inteira Mista (PIM) e quatro heurísticas para a minimização do atraso total no problema 2sAFS. As duas primeira heurísticas são baseadas nas meta-heurísticas *Iterated Greedy* (IG) e *Iterated Local Search* (ILS), respectivamente. As outras duas heurísticas são métodos híbridos que combinam IG e ILS com *Variable Neighborhood Descent* (VND). As heurísticas IG (Ruiz e Stützle, 2007), ILS (Lourenço et al., 2010) e VND (Hansen e Mladenovic, 2003) são métodos simples, robustos e eficientes que são aplicados para resolver diferentes problemas de otimização combinatória. IG e ILS ainda são pouco aplicados para problemas de sequenciamento de tarefas.

2. Definição Formal e Modelagem do Problema

O 2sAFS consiste em fabricar um conjunto de *n* produtos (tarefas), sendo que cada tarefa possui *m* componentes que são fabricados de forma independente. No primeiro estágio, os componentes das tarefas são fabricados em um ambiente de *m* máquinas paralelas (cada

componente é feito em uma máquina). Quando todos os componentes de um produto estiverem prontos, no segundo estágio, o produto é montado. A montagem é realizada por uma única máquina. Neste problema, todas as tarefas estão disponíveis para serem processadas em um tempo zero, todas as máquinas processam apenas uma tarefa por vez e não podem ser interrompidas durante seu processamento. Note que para cada tarefa existem m+1 operações distintas, sendo que m operações independentes são feitas no primeiro estágio e uma operação é realizada no segundo estágio. Os dois estágios caracterizam o problema como um ambiente de produção *flowshop*, pois uma tarefa i só pode iniciar a montagem (no estágio 2) quando todas seus componentes forem finalizadas no estágio 1.

O objetivo do problema é determinar o sequenciamento das tarefas nas máquinas de forma soma dos atrasos dastarefas (atraso total) seja minimizado. No problema abordado, os seguintes dados são conhecidos: o tempo de processamento da tarefa i na máquina k do primeiro estágio (P_{ik}), tempo de montagem da tarefa i (PA_i) e data de entrega da tarefa i (d_i).

A seguir apresenta-se um modelo PIM para o problema. Este modelo é uma adaptação do modelo apresentado por Andrés e Hatami (2011). Os seguintes índices são usados no modelo: i = 1,..,n (índices da tarefas); k = 1,2,..,m (índices de máquinas do primeiro estágio); j = 1,2,...,n (índices para indicar posições de tarefa na sequência). O modelo usa os seguintes paramentos e variáveis de decisão:

- $d_{[j]}$: data de entrega da tarefa que está na posição j.
- $X_{i[j]}$: igual a 1 se a tarefa i está na posição j da sequência, caso contrário zero.
- $CP_{[j]k}$: tempo de conclusão da tarefa que está na posição j na máquina k do primeiro estágio.
- $C_{[j]}$: tempo de conclusão no segundo estágio da tarefa que está na posição j.
- $T_{[j]}$: atraso total da tarefaque está na posiçãoj.

O modelo é:

$$\text{Min} \quad \sum_{j=1}^{n} T_{[j]} \tag{1}$$

Sujeito a:

$$\sum_{j=1}^{n} X_{i[j]} = 1 \; ; \; \forall_{i=1,\ldots,n} \tag{2}$$

$$\sum_{i=1}^{n} X_{i[j]} = 1 \; ; \; \forall_{j=1,\ldots,n} \tag{3}$$

$$CP_{[1]k} = \sum_{i=1}^{n} (P_{ik} * X_{i[1]}) \; ; \; \forall_{k=1,\ldots,m} \tag{4}$$

$$CP_{[j]k} = CP_{[j-1]k} + \sum_{i=1}^{n} (P_{ik} * X_{i[j]}) \; ; \; \forall_{j=2,\ldots,n}, \forall_{k=1,\ldots,m} \tag{5}$$

$$C_{[1]} = CP_{[1]k} + \sum_{i=1}^{n} (PA_i * X_{i[1]}) \; ; \; \forall_{k=1,\ldots,m} \tag{6}$$

$$C_{[j]} = C_{[j-1]} + \sum_{i=1}^{n} (PA_i * X_{i[j]}) \; ; \; \forall_{j=2,\ldots,n} \tag{7}$$

$$C_{[j]} = CP_{[j]k} + \sum_{i=1}^{n} (PA_i * X_{i[j]}) \; ; \; \forall_{j=1,\ldots,n}, \forall_{k=1,\ldots,m} \tag{8}$$

$$T_{[j]} \geq C_{[j]} - d_{[j]} \; ; \forall_{j=1,\ldots,n} \tag{9}$$

$$T_{[j]} \geq 0, \qquad CP_{[j]k} \geq 0, \qquad C_{[j]} \geq 0; \; \forall_{j=1,\ldots,n}, \forall_{k=1,\ldots,m} \tag{10}$$

$$X_{i[j]} = 0 \; ou \; 1 \; ; \; \forall_{i=1,\ldots,n}, \forall_{j=1,\ldots,n} \tag{11}$$

A função objetivo (1) minimiza o atraso total. A restrição (2) garante que em todas as posições da sequência tenha apenas uma tarefa atribuída, enquanto a restrição (3) garante que cada tarefa é atribuída a uma única posição. As restrições (4) e (5) calculam o tempo de conclusão de cada tarefa nas máquinas do primeiro estágio. As restrições (6) e (7) calculam o tempo de conclusão no segundo estágio. A restrição (8) garante que a execução da segunda fase comece depois que todas as *m* operações da primeira fase sejam concluídas. O atraso de cada tarefa é calculado pela restrição (9). As restrições (10) e (11) representam as condições de domínio das variáveis.

3. Métodos Heurísticos para o Problema 2sAFS

Neste trabalho, para determinar uma solução viável e de boa qualidade para o problema 2sAFS, foram utilizadas as meta-heurísticas *Iterated Greedy* (IG) e *Iterated Local Search* (ILS). Quatro heurísticas foram desenvolvidas. As duas primeiras são algoritmos convencionais: IG e ILS. As outras duas são algoritmos híbridos obtidos pela combinação do IG e ILS com a heurística VND: IG-VND e ILS-VND. Nas próximas subseções são apresentadas as descrições das implementações das heurísticas.

3.1.Heurística IG

IG, proposta por Ruiz e Stutzle (2007), é um algoritmo heurístico muito utilizado para resolver problemas de sequenciamento. Uma descrição geral do IG é dada no Algoritmo 1. O algoritmo começa gerando uma solução inicial *S*. Em seguida aplica-se uma busca local nessa solução obtendo-se a melhor solução atual S_b. O laço principal do algoritmo consiste de três procedimentos: Destruição-Construção, Busca Local e o Critério de Aceitação. O procedimento de Destruição-Construção gera uma nova solução S_p a partir da melhor solução S_b. A solução S_p é melhorada pela Busca Local. Se a solução *S* encontrada pela Busca Local for melhor que S_b, então atualiza-se S_b (Critério de Aceitação). Note que, no algoritmo são utilizados três parâmetros, *prob*, D_{max} e *Cparada*. O parâmetro D_{max} determina o nível máximo de destruição da solução no procedimento Destruição-Construção. O nível de destruição é determinado pelo contador *d* que é inicializado em 2 e ele é incrementado, até um valor máximo D_{max}, caso não sejam encontradas melhorias na solução S_b. Se é encontrada uma melhoria na solução, *d* volta para seu valor inicial 2. O parâmetro *prob* é utilizado para fazer uma redução no tamanho da vizinhança na Busca Local. Os procedimentos do IG são executados enquanto um critério de parada *Cparada* não é satisfeito. A seguir descrevem-se os procedimentos principais do algoritmo IG.

Algoritmo 1: **IG**(*prob*, D_{max}, *Cparada*)

```
S ←ConstruçãoInicial( );
S_b←Busca Local (S, prob);
d ← 2;
Enquanto Cparada faça
   S_p←Destruição-Construção(S_b, d );
   S←Busca Local (S_p, prob);
   Se f(S) < f(S_b) então //Critério de Aceitação
      S_b ← S;
      d ← 2;
   Senão Se d<D_max então
      d ← d + 1;
   Senão d ← 2;
Fim_Enquanto
Retorna S_b;
```

3.1.1. Construção Inicial

A construção inicial de uma solução é realizada através da regra de prioridade *Earliest Due Date* – EDD, onde as tarefas são ordenadas em ordem crescente das datas de entrega. De acordo com essa ordenação determina-se uma sequência inicial das tarefas. No problema 2sAFS, considera-se que a sequência de tarefas é mesma para todas as máquinas (Potts et al., 1995).

3.1.2.Destruição e Construção

A Destruição consiste em remover aleatoriamente *d* tarefas da solução atual (S_b), obtendo uma solução parcial *S'*. A Construção aplica um algoritmo construtivo guloso para reconstruir *S'* e obter a solução completa. Para cada tarefa removida, procura-se a melhor posição para ser inserida na sequência de *S'*. Ou seja, testa-se todas as posições possíveis, procurando a posição que gera o menor valor da função objetivo (atraso total) da solução parcial, para que a próxima tarefa removida seja inserida. No final da Construção tem-se uma nova solução completa (S_p).

3.1.3. Busca Local

A busca local é usada para melhorar a solução inicial *S* e a solução S_p obtida pelo procedimento de Destruição-Construção. A Busca local começa com uma solução *S* ou S_p, e gera uma vizinhança que contém soluções que são alcançadas através de movimentos individuais realizados na solução corrente. Desta vizinhança, uma solução que é melhor que a solução corrente é selecionada. A solução escolhida torna-se a nova solução corrente e o processo continua até que um ótimo local seja alcançado. Neste estudo, o procedimento de Busca Local é baseado na vizinhança por inserção. Esta vizinhança consiste em remover cada tarefa *i* da sua posição original e inseri-la nas *n*-1 posições restantes. Este movimento gera uma vizinhança de tamanho máximo de $(n\text{-}1)^2$. A solução corrente é substituída pela primeira solução vizinha que melhore o valor do atraso total. O procedimento é repetido enquanto há melhoria na solução corrente. Neste procedimento é utilizado um parâmetro *prob* que define a probabilidade de uma tarefa *i* ser escolhida para ser inserida nas demais posições da sequência. Isto permite a redução do tamanho da vizinhança.

3.2. IG com VND

A heurística IG com VND (IG-VND) é similar ao IG, porém no procedimento de Busca Local aplica-se a heurística *Variable Neighborhood Descent* – VND (Hansen e Mladenovic, 2003), que é descrita a seguir. O procedimento VND utiliza duas estruturas de vizinhanças: vizinhança por inserção N_1 (utilizada pela Busca Local convencional - seção 3.1.3) e a vizinhança por troca N_2. Esta segunda vizinhança consiste em trocar cada tarefa *i* da sequência com as *n*-1 tarefas restantes, assim, gerando $n(n\text{-}1)/2$ soluções vizinhas. Para reduzir esta vizinhança, também é utilizado o parâmetro *prob* que define a probabilidade de uma tarefa *i* ser trocada com outra. Vale ressaltar que, a vizinhança N_1 é uma das melhores vizinhanças em problemas de sequenciamento (Ruiz e Stützle, 2007). O algoritmo VND recebe como parâmetro de entrada uma solução *S* a ser melhorada. Inicialmente, utilizando a vizinhança N_1 determina-se a melhor solução vizinha (S_v) de *S*. Se a solução S_v for melhor que a solução *S*, o algoritmo retoma a busca com a solução S_v e utilizando a mesma vizinhança N_1. Caso contrário, o algoritmo retoma a busca com a mesma solução *S* e utilizando a vizinhança por troca N_2. Se a melhor solução S_v na vizinhança N_2 for melhor que a solução corrente *S*, o algoritmo retoma a busca com a vizinhança N_1. Caso contrário, o algoritmo termina a busca, retornando a melhor solução encontrada.

3.3.Heurísticas ILS e ILS-VND

O ILS é uma meta-heurística simples, robusta e efetiva que aplica repetidamente uma Busca Local a soluções obtidas ao perturbar soluções ótimas locais previamente visitadas. O algoritmo do ILS assemelha-se com o algoritmo do IG apresentado na seção 3.1, diferenciando apenas no procedimento de Destruição-Construção. No ILS é utilizado um procedimento de Perturbação para escapar de ótimos locais. Neste trabalho, a perturbação de uma solução consiste em realizar um número *d* de trocas de tarefas escolhidas aleatoriamente. Igual que no IG, o

parâmetro *d* inicia com valor 2 e ele é incrementado até um valor máximo D_{max}, caso não sejam encontradas melhorias na solução corrente. No ILS é utilizado o mesmo procedimento de Busca Local apresentado em 3.1.3.

No algoritmo ILS-VND, utiliza-se como busca local o procedimento VND apresentado em 3.2.

4. Testes Computacionais e Análise Estatística dos Resultados

Nesta seção, experimentos computacionais e testes estatísticos são realizados a fim de avaliar o desempenho do modelo matemático proposto e das quatro heurísticas propostas. As heurísticas foram implementados em linguagem C++ e compiladas com o Microsoft Visual C++ 2010. Já o modelo matemático foi implementado em C++ utilizando a biblioteca Concert do CPLEX 12.4,e também compilado com o Microsoft Visual C++ 2010. Os experimentos foram realizado em um computador com o processador Intel Core i7-4700K com velocidade de 4.00GHz, memória RAM de 32.0 GB e sistema operacional Windows 8.1 Pro. A métrica utilizada para avaliar os resultados dos algoritmos é o Desvio Percentual Relativo (*RPD*) que é obtido pela seguinte fórmula: $RDP = 100\times(f_{algoritmo} - f_{melhor})/f_{melhor}$, onde $f_{algoritmo}$ corresponde ao valor da função objetivo obtido por um dado algoritmo e f_{melhor} correspondente à melhor solução encontrada a partir da execução de todos os métodos comparados. Cada heurística foi executada 10 vezes para resolver uma instância e o *RPD* é calculado considerando o valor médio das 10 execuções.

4.1. Instâncias do Problema

As instâncias do problema foram geradas da mesma maneira que Allahverdi al. (2014). Vale lembrar que esses autores semente geraram instâncias com 30, 40, 50, 60 tarefas, no entanto essas instâncias não são disponibilizadas. Neste trabalho foram considerados dois conjuntos de instâncias: instâncias de pequeno porte, com número de tarefas $n = \{10, 12, 15, 20\}$ e número de máquinas no primeiro estágio $m = \{2, 4, 6\}$; e instâncias de médio-grande porte, com $n = \{30, 40, 50, 60, 100, 200\}$ e $m = \{5, 10, 20\}$. Foram considerados 9 cenários datas de entrega (*cd*). Para cada configuração $n\times m\times cd$, 3 instâncias foram geradas. Assim, foram geradas 324 instâncias de pequeno porte e 486 de médio-grande porte. As instâncias e os resultados obtidos serão disponibilizados no site da nossa instituição.

4.2. Parâmetros dos Algoritmos Heurísticos

As heurísticas IG e IG-VND possuem três parâmetros a serem definidos: *prob*, D_{max} e *CParada*. Para o parâmetro *prob*, que é usado para reduzir o tamanho das vizinhanças, foram testados os seguintes valores: {0,2, 0,5, 0,6, 0,7, 0,8}. Para o parâmetro D_{max} foram testados $n/2$, $n/5$ e $n/8$. Como critério de parada é utilizado um tempo máximo de execução do algoritmo. Este tempo foi fixado em $Cparada = n\times m\times 100$ milissegundos. Após realizar os testes computacionais com esses parâmetros, os melhores valores obtidos foram $prob = 0{,}5$ e $D_{max} = n/5$. Os mesmos parâmetros foram utilizados nos algoritmos ILS e ILS-VND, já que eles são algoritmos similares ao IG e IG-VND, respectivamente. A diferença está apenas na perturbação das soluções. Vale ressaltar que no ILS, o nível de perturbação não é fixa, ele varia no intervalo $[2, D_{max}]$.

4.3 Comparações das Heurísticas com o Modelo Matemático

As heurísticas propostas foram avaliadas através da comparação com os resultados obtidos pelo solver CPLEX. O modelo de PIM, para as instâncias com até 40 tarefas, é resolvido com o CPLEX utilizando um tempo limite de uma hora. Quando não encontrado uma solução ótima até o tempo estabelecido, o CPLEX retorna a melhor solução encontrada até o momento. Para cada heurística avaliada, foi calculado o *RPD* considerando a solução do solver CPLEX como a melhor solução.

Tabela 1: Média dos *RPD*sna comparação das heurísticas com oCPLEX.

Instâncias $n \times m$	IG	ILS	IG-VND	ILS-VND	CPLEX tempo(s)
10 x 2	0,1231	**0**	0,0126	**0**	0,42
10 x 4	0,0405	**0**	**0**	**0**	0,43
10 x 6	**0**	**0**	**0**	**0**	0,45
12 x 2	0,0009	**0**	0,0002	**0**	0,49
12 x 4	0,0190	**0**	0,0028	**0**	0,65
12 x 6	0,1535	**0**	0,0062	**0**	0,71
15 x 2	0,0040	0,0006	0,0006	**0**	0,83
15 x 4	**0**	**0**	**0**	**0**	1,20
15 x 6	0,0004	**0**	**0**	**0**	1,91
20 x 2	0,0113	0,0073	**0,0015**	0,0029	8,80
20 x 4	0,0044	0,0040	-0,0005	**-0,0008**	141,67
20 x 6	0,0193	0,0100	**0,0040**	0,0060	242,77
30 x 5	0,0276	0,0576	**-0,0025**	0,0006	769,97
30 x 10	-0,1015	-0,0511	**-0,1285**	-0,1232	1153,03
30 x 20	-0,2130	-0,2086	-0,2401	**-0,2482**	1228,46
40 x 5	-0,0478	0,0750	**-0,1625**	-0,1392	1664,92
40 x 10	-0,0669	-0,0074	**-0,0991**	-0,0840	2404,99
40 x 20	-0,8514	-0,7630	**-0,9323**	-0,8891	3076,14
Médias	-0,0487	-0,0486	**-0,0854**	-0,0819	

Na Tabela 1 são apresentados os valores médios do *RPD* para cada grupo $n \times m$ com 27 instâncias, e os tempos médios gastos pelo CPLEX. Valores da média do $RPD \geq 0$ significam que as soluções heurísticas foram comparadas com as soluções ótimas obtidas pelo CPLEX. Valores negativos significam que em algumas instâncias do grupo, a solução heurística é melhor que a solução do CPLEX. Analisando a Tabela 1, pode-se verificar que o algoritmo ILS-VND obteve a solução ótima em todas as instâncias com até 15 tarefas. Para as instâncias com 20, 30 e 40 tarefas, em geral, o IG-VND foi melhor. Também pode ser observado que, o uso do VND provocou uma melhoria nos algoritmos ILS e IG. É importante mencionar que para um total de 486 instâncias, 445 soluções ótimas foram obtidas pelo CPLEX, dos quais 451, 437, 397 e 380 soluções ótimas foram obtidas pelos métodos IG-VND, ILS-VND, ILS e IG, respectivamente.

4.4. Resultados para Instâncias de Grande Porte

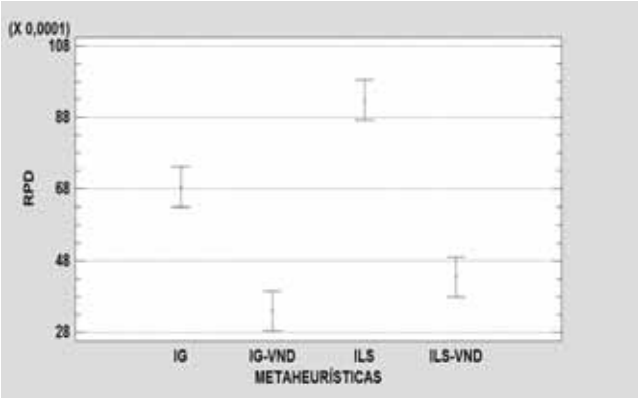
Os algoritmos são comparados nas instâncias com 50, 60, 100, 200 tarefas. Para cada instância,os algoritmos foram executados 10 vezes e considera-se a média dos valores da função objetivo obtidos para calcular o *RPD*. Para instâncias de grande porte não foi possível obter soluções ótimas aplicando o solver CPLEX. Desta forma, os valores médios são comparados com o melhor valor da função objetivo que é encontrado com todas as execuções de todos os algoritmos. A Tabela 2 mostra os *RPDs* médios para cada grupo de instância e cada algoritmo. Os valores em negrito representam as melhores medias. Observa-se que o algoritmo IG-VND apresentou as melhores médias, exceto para o grupo de instâncias 200×20. Este algoritmo obteve uma média global de 0,34%, e o segundo melhor algoritmo ILS-VND obteve 0,43%. Percebe-se claramente que, com o uso do VND os algoritmos IG e ILS obtiveram uma melhoria em todas as instâncias. Também nota-se que o algoritmo IG é consideravelmente melhor que o algoritmo ILS. Uma vez que as diferenças entre as médias do RPD são pequenas, é importante executar um teste estatístico, para verificar se existem diferenças estatisticamente significativas entre os algoritmos. Foi aplicado teste de Análise de Variância (ANOVA) para comparar os quatro algoritmos. Antes foram verificadas as hipóteses do teste, que são: normalidade, homogeneidade da variância e independência dos resíduos. O teste ANOVA indicou que existem diferenças significativas entre os 4 algoritmos. A Figura 1 mostra que pelo teste HSD de Tukey o algoritmo IG-VND apresenta uma diferença significativa em relação aos algoritmos IG e ILS com um nível de confiança de 95%. Porém, estatisticamente, não existe uma diferença entre os algoritmos IG-VND e ILS-VND, embora o IG-VND apresente uma melhor média em 91,6% dos grupos de instâncias de

grande porte. Na Figura também observa-se que, os algoritmos IG e ILS melhoraram significativamente com uso do VND.

Tabela2: Média dos RPD para comparações dos algoritmos heurísticos.

Instância *n*×*m*	IG	ILS	IG-VND	ILS-VND
50 x 5	0,13	0,26	**0,06**	0,11
50 x 10	0,24	0,43	**0,12**	0,20
50 x 20	0,25	0,38	**0,11**	0,18
60 x 5	0,32	0,70	**0,10**	0,22
60 x 10	0,38	0,64	**0,18**	0,33
60 x 20	0,42	0,64	**0,28**	0,40
100 x 5	0,71	1,17	**0,27**	0,45
100 x 10	0,60	1,01	**0,33**	0,54
100 x 20	0,76	1,25	**0,39**	0,65
200 x 5	1,65	1,86	**0,55**	0,57
200 x 10	1,28	1,33	**0,65**	0,69
200 x 20	1,49	1,46	1,03	**0,87**
Médias	0,69	0,93	**0,34**	0,43

Figura 1: Gráfico de médias e intervalos HSD de Tukeypara os algoritmos comparados.



5. Conclusões

Nesse trabalho foi estudado um problema de sequenciamento de tarefas que é fortemente NP-Difícil. Para resolver tal problema, inicialmente foi proposto um modelo de PIM para obtenção de soluções ótimas em instâncias de pequeno e médio porte. Já que a resolução do modelo para instâncias de grande porte é inviável, foram propostos quatro algoritmos heurísticos, sendo dois híbridos que combinam as técnicas IG, ILS com VND. Os testes computacionais mostraram que com o uso da heurística VND na fase de busca local dos algoritmos IG e ILS, os resultados melhoraram significativamente. Além disso, IG apresentou um desempenho superior que o ILS, e o IG-VND foi o melhor algoritmo na solução do problema. Os resultados obtidos foram validados estaticamente. De acordo com o nosso conhecimento, na literatura não existem aplicações do IG e ILS para o problema 2sAFS com critério atraso total. Dessa forma, podemos concluir que este trabalho contribui para o estado da arte do problema. Como trabalhos futuros sugerem-se aplicar as heurísticas propostas para outras variações do problema, por exemplo, considerando tempos de preparação e considerando um estágio de transporte entre os estágios de produção e montagem.

Agradecimentos: Os autores agradecem à FAPEMIG pela bolsa de Iniciação Científica concedida para desenvolvimento deste trabalho.

Referências

Allahverdi, A., e Al-Anzi, F. S. (2006a). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. *International Journal of Production Research*, *44*(22), 4713-4735.

Allahverdi, A., e Al-Anzi, F. S. (2006b). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers & Operations Research*, *33*(4), 1056-1080.

Allahverdi, A., e Aydilek, H. (2014). The two stage assembly flowshop scheduling problem to minimize total tardiness. *Journal of Intelligent Manufacturing*, 1-13.

Al-Anzi, F. S., e Allahverdi, A. (2009). Heuristics for a two-stage assembly flowshop with bicriteria of maximum lateness and makespan. *Computers & Operations Research*, *36*(9), 2682-2689.

Andrés, C., e Hatami, S. (2011, September). The three stage assembly permutation flowshop scheduling problem. In *V international conference on industrial engineering and industrial management* (pp. 867-875).

Du, J. e Leung, J. (1990). Minimizing total tardiness on one machine is NP-hard. *Operations Research, 38(1):2236.*

Gupta, J. N., Strusevich, V. A., e Zwaneveld, C. M. (1997). Two-stage no-wait scheduling models with setup and removal times separated. *Computers & Operations Research*, *24*(11), 1025-1031.

Hansen, P., e Mladenovic, N. (2003). *A tutorial on variable neighborhood search*. *Grouped'études et de recherche en analyse des décisions, HEC Montréal*.

Lee, C. Y., Cheng, T. C. E., e Lin, B. M. T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, *39*(5), 616-625.

Lourenço, H. R., Martin, O. C., e Stützle, T. (2010). Iterated local search: Framework and applications. *Handbook of Metaheuristics* (pp. 363-397). Springer US.

Mozdgir, A., FatemiGhomi, 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*, *51*(12), 3625-3642.

Navaei, J., Ghomi, S. F., Jolai, F., Shiraqai, M. E., e Hidaji, H. (2013). Two-stage flow-shop scheduling problem with non-identical second stage assembly machines. *The International Journal of Advanced Manufacturing Technology*, *69*(9-12), 2215-2226.

Potts, C. N., Sevast'Janov, S. V., e Zwaneveld, C. M.(1995). The two-stage assembly scheduling problem: complexity and approximation. *Operations Research*, *43*(2), 346-355.

Ruiz, R., e Stützle, T.(2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033-2049.

Seidgar, H., Kiani, M., Abedi, M., e Fazlollahtabar, H. (2014). An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. *International Journal of Production Research*, *52*(4), 1240-1256.