CLAIO SBPO
Congreso Latino-Iberoamericano de Investigación Operativa
Simpósio Brasileiro de Pesquisa Operacional
24 a 28/09
Rio de Janeiro
Brasil - 2012

# UM ALGORITMO BASEADO EM *ITERATED GREEDY* PARA MINIMIZAÇÃO DO MAKESPAN NO PROBLEMA DE SEQUENCIAMENTO FLOWSHOP HÍBRIDO E FLEXÍVEL

**Eduardo Camargo de Siqueira**[1], **Sérgio Ricardo de Souza**[1], **Marcone Jamilson Freitas Souza**[2]

[1]Av. Amazonas, 7675, Nova Gameleira
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil

eduardosiqueira@dppg.cefetmg.br, sergio@dppg.cefetmg.br

[2]Departamento de Computação, Campus Universitário
Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais

marcone@iceb.ufop.br



***Resumo.*** *Este trabalho propõe um algoritmo heurístico baseado no procedimento* Iterated Greedy *para resolução do problema de sequenciamento flowshop híbrido e flexível. O algoritmo proposto utiliza o método guloso NEH para geração das soluções iniciais e um procedimento de busca local, denominado* Iterative Improvement Insertion*, para refinar essa solução. Os experimentos computacionais mostram que o algoritmo proposto é capaz de superar resultados da literatura em um subconjunto de instâncias do problema.*

**PALAVRAS-CHAVE:** *Sequenciamento de tarefas, Flowshop, Metaheurísticas, Makespan.*

***Abstract.*** *This paper proposes a heuristic algorithm based on the Iterated Greedy procedure for solving hybrid flexible flowshop scheduling problems. The proposed algorithm uses the greedy method NEH to generate initial solutions and the Iterative Improvement Insertion procedure in order to refine the solutions. The computational experiments show that the proposed algorithm is able to outperform results of the literature for a subset of instances of the problem.*

**KEYWORDS:** *Scheduling, Flowshop, Metaheuristics, Makespan.*

## 1 Introdução

Problemas de sequenciamento aparecem em vários setores de atividade, entre eles o da produção industrial. Nesse contexto, o problema consiste em definir uma sequência de tarefas a serem executadas em um conjunto de máquinas, satisfazendo a um conjunto de restrições operacionais e atendendo a um certo objetivo, que pode ser a minimização do tempo de término de todas as tarefas, a minimização dos tempos de atraso na conclusão das tarefas, entre outros.

Existem diversos estudos a respeito dos problemas de sequenciamento, porém sempre existiu uma distância entre teoria e prática. Muitos modelos desenvolvidos não incorporam várias restrições operacionais, impedindo, assim, sua aplicação em situações reais. Há, no entanto, uma tendência recente em desenvolver abordagens de solução para problemas reais (Ruiz et al., 2008).

Neste sentido, este trabalho considera o problema real descrito em Ruiz et al. (2008), que trata uma variação do problema de *flowshop* híbrido (HFS, do inglês *Hybrid Flowshop*), em que um conjunto de tarefas passa por um conjunto de estágios e para cada estágio existe um conjunto de máquinas paralelas não-relacionadas. A característica de um *flowshop* é que o fluxo de processamento nas máquinas é o mesmo, ou seja, todas as tarefas seguem a mesma sequência de estágios. No entanto, no problema considerado os estágios podem não ser todos executados. Tal variante, denominada *Flowline* Híbrido e Flexível (HFFL, do inglês *Hybrid Flowshop and Flowline*), é, desta forma, uma generalização do HFS e do *Flowline* Flexível. O critério de otimização considerado é o de minimizar o maior tempo de conclusão das máquinas, o chamado *makespan*.

Tendo em vista o fato de o HFFL ser da classe NP-difícil (Ruiz et al., 2008), este trabalho propõe um método iterativo chamado *Iterated Greedy* (Ruiz e Stützle, 2005). A utilização desse algoritmo foi motivada pelo seu uso bem sucedido na solução de problemas de sequenciamento de tarefas, como relatados em Ruiz e Stützle (2007) e Ribas et al. (2011).

O restante deste trabalho está organizado da seguinte forma. Na seção 2 é feito um levantamento bibliográfico dos problemas de *flowshop*. Na seção 3 o problema abordado é formulado e exemplificado. O algoritmo proposto, inspirado no trabalho de Ruiz e Stützle (2007), é descrito na seção 4. Na seção 5 são mostrados os resultados encontrados, enquanto a última seção conclui o trabalho e aponta os trabalhos futuros.

## 2 Revisão da Literatura

Como comentado anteriormente, o HFFL é uma generalização do *flowshop* híbrido (HFS). O HFS, por sua vez, é um problema diferente do *Flowshop* Flexível e do *Flowline* Flexível, pois para esses dois últimos problemas as máquinas disponíveis em cada estágio são idênticas. O HFS não tem essa restrição. Alguns estágios podem ter apenas uma máquina, mas pelo menos um estágio deve ter um grupo de máquinas em paralelo. Estas máquinas geralmente são diferentes (Burtseva et al., 2012).

Segundo Ribas et al. (2010), as pesquisas sobre agendamento HFS apareceram na década de 1970. Um dos primeiros trabalhos sobre HFS na modelagem do sistema de produção em uma indústria de fibras sintéticas é o de Salvador (1973). Garey e Johnson (1979) mostraram que o problema HFS com o objetivo *makespan* é NP-completo. Assim, como o HFFL é uma generalização do HFS, então o HFFL também o é.

Ruiz e Vázquez-Rodríguez (2010) fazem uma revisão de algoritmos exatos, heurísticas e meta-heurísticas para o HFS. Nishi et al. (2010) apresentam um método de relaxação lagrangiana com geração de cortes para esse problema. Ziaeifar et al. (2011) apresentam uma nova modelagem matemática para o HFS, bem como um algoritmo genético. Yaurima et al. (2009) modelaram um problema de *Flowshop* Híbrido em linhas de montagem de placas de circuito e utilizaram um Algoritmo Genético para resolvê-lo.

Naderi et al. (2010) chama a atenção para duas questões importantes a respeito de HFFL: a determinação da sequência em cada estágio e a distribuição das tarefas nas máquinas em cada estágio. É apresentado um algoritmo baseado na meta-heurística *Iterated Local Search* (ILS) com o objetivo de minimizar o *makespan*.

Defersha (2010) apresenta um modelo matemático para o problema de sequenciamento HFFL baseado em uma técnica de divisão de tarefas em subtarefas. Defersha e Chen (2011) desenvolvem um processo de solução baseado em Algoritmo Genético para o HFFL. O algoritmo foi implementado em plataformas de computação sequenciais e paralelas, e os desempenhos avaliados e comparados. Urlings e Ruiz (2010) e Zandieh et al. (2010) também propõem Algoritmos Genéticos para resolução desse problema. O objetivo nesses dois últimos trabalhos é a minimização do *makespan*.

Pacciarelli e D'Ariano (2011) e Venditti et al. (2010) apresentam problemas reais de sequenciamento em indústrias farmacêuticas. A meta-heurística Busca Tabu é utilizada para resolver tais problemas.

Urlings et al. (2010) tratam a minimização do *makespan* em problemas HFFL por meio de uma nova meta-heurística, denominada SRS. Esse novo método combina um algoritmo guloso iterativo e o ILS.

## 3 Formulação do problema

O problema HFFL é definido sobre um conjunto de tarefas $N = \{1, 2, 3, ..., n\}$, que devem ser executadas em um conjunto de estágios $M = \{1, 2, 3, \cdots, m\}$. Para cada etapa há um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas podem saltar estágios. Consideram-se, ainda, as seguintes características:

- $F_j$: Conjunto de estágios que a tarefa $j$ visita, sendo $1 < F_j < m$;
- $p_{ilj}$: Tempo de processamento da tarefa $j$ na máquina $l$ e estágio $i$.
- $rm_{il}$: Tempo de *release* da máquina $l$ no estágio $i$. Ele representa o tempo de início dos processos na máquina. Nenhuma tarefa pode iniciar na máquina antes do tempo de *release*.
- $E_{ij}$: Conjunto de máquinas elegíveis para a tarefa $j$ no estágio $i$.
- $lag_{ilj}$: Tempo de atraso entre o fim da tarefa $j$, na máquina $l$ do estágio $i$, e o início do próximo estágio da tarefa $j$.
- $S_{iljk}$: Tempo de preparação (*setup*) da máquina $l$ no estágio $i$, quando a tarefa $k$ é executada logo após a tarefa $j$. Existe um valor binário associado, $A_{iljk}$, que indica se o *setup* é antecipativo, ou seja, $A_{iljk}$ assume o valor 1 se a preparação pode ser feita antes que a tarefa seja liberada na fase anterior e o valor 0, caso contrário.

De forma a exemplificar o problema, seja uma instância com 5 tarefas e 3 estágios, com duas máquinas nos dois primeiros estágios e uma máquina no último estágio. A Tabela 1 mostra quais máquinas são elegiveis para cada tarefa em cada estágio. Nesta Tabela, $j$

indica uma tarefa, e a linha $i$ representa cada um dos 3 estágios. Por essa Tabela, pode-se verificar, por exemplo, que a tarefa 1 pode ser executada nas máquinas 1 e 2 no estágio 1, na máquina 4 no estágio 2 e na máquina 5 no estágio 3. Pode-se verificar, também, que as tarefas 1 e 5 visitam todos os estágios, enquanto as tarefas 2, 3 e 4 pulam os estágios 3, 1 e 2, respectivamente.

**Tabela 1. Elegibilidade**

| | $i$ | 1 | 2 | 3 |
|---|---|---|---|---|
| $j$ | 1 | 1,2 | 4 | 5 |
| | 2 | 1,2 | 3,4 | - |
| | 3 | - | 3 | 5 |
| | 4 | 2 | - | 5 |
| | 5 | 1,2 | 3,4 | 5 |

A Tabela 2 contém o tempo de *release* para cada máquina e o tempo de processamento de cada tarefa em cada máquina, enquanto a Tabela 3 mostra os tempos de atraso. Nestas duas Tabelas, $j$ indica uma tarefa, a linha $i$ representa cada um dos 3 estágios, e a linha $l$ representa cada uma das 5 máquinas. Na Tabela 2, a linha $rm_{il}$ indica os tempos de *release* para cada máquina, e cada célula $p_{ilj}$ os tempos de processamento.

**Tabela 2. Tempos de processamento e *release***

| | $i$ | 1 | | 2 | | 3 |
|---|---|---|---|---|---|---|
| | $l$ | 1 | 2 | 3 | 4 | 5 |
| | $rm_{il}$ | 4 | 3 | 8 | 16 | 23 |
| | $p_{ilj}$ | | | | | |
| $j$ | 1 | 10 | 15 | 0 | 8 | 6 |
| | 2 | 6 | 9 | 11 | 4 | 0 |
| | 3 | 0 | 0 | 9 | 0 | 8 |
| | 4 | 0 | 10 | 0 | 0 | 6 |
| | 5 | 11 | 14 | 6 | 12 | 3 |

**Tabela 3. Tempo de atraso**

| | $i$ | 1 | | 2 | |
|---|---|---|---|---|---|
| | $l$ | 1 | 2 | 3 | 4 |
| $j$ | 1 | 10 | 2 | 0 | -4 |
| | 2 | 2 | -2 | 0 | 0 |
| | 3 | 0 | 0 | 3 | 0 |
| | 4 | 0 | 1 | 0 | 0 |
| | 5 | -5 | -6 | -3 | 8 |

A Tabela 4 mostra os tempos de preparação (tempos de *setup*), que são dependentes da sequência, e seus valores binários associados. Esse valor binário representa se o *setup* é antecipativo (1), ou não (0). Nesta Tabela, $j$ e $k$ indicam uma tarefa, a linha $i$ representa cada um dos 3 estágios, e os dados de cada máquina em cada estágio estão separados por vírgula. Por exemplo, o tempo de *setup* entre as tarefas 1 e 4 na máquina 2 do primeiro estágio é igual a 8 e o valor binário é igual a 1 (antecipativo), porém o *setup* na máquina 1 não existe pois essa máquina não é elegível para a tarefa 4.

**Tabela 4. Tempos de *setup* e valores $A_{iljk}$**

| | $i$ | 1 | | | | |
|---|---|---|---|---|---|---|
| | $k$ | 1 | 2 | 3 | 4 | 5 |
| $j$ | 1 | -,- | 3(1),6(1) | -,- | -,8(1) | 4(1),2(1) |
| | 2 | 4(0),5(0) | -,- | -,- | -,6(1) | 1(1),4(1) |
| | 3 | -,- | -,- | -,- | -,- | -,- |
| | 4 | -,8(0) | -,- | -,- | -,- | -,2(1) |
| | 5 | 6(0),10(0) | -,- | -,- | -,4(0) | -,- |
| | $i$ | 2 | | | | |
| $j$ | 1 | -,- | -,6(1) | -,- | -,- | -,6(1) |
| | 2 | -,5(1) | -,- | 6(0),- | -,- | 4,2(0) |
| | 3 | -,- | 8(0),- | -,- | -,- | 5(1),- |
| | 4 | -,- | -,- | -,- | -,- | -,- |
| | 5 | -,4(1) | -,- | -,- | -,- | -,- |
| | $i$ | 3 | | | | |
| $j$ | 1 | - | - | 6(1) | 3(1) | 9(1) |
| | 2 | - | - | - | - | - |
| | 3 | 4(0) | - | - | 1(0) | 8(1) |
| | 4 | 5(0) | - | - | - | 2(1) |
| | 5 | 2(0) | - | - | 6(0) | - |

A Figura 1 ilustra um possível sequenciamento para este exemplo. Note que o *makespan* para esse sequenciamento é igual a 63.

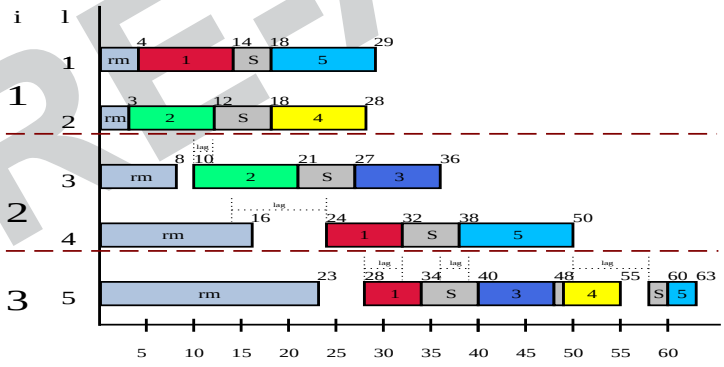


**Figura 1. Diagrama de GANTT.** $C_{max} = 63$

## 4 Descrição do Algoritmo Proposto

### 4.1 Representação de uma solução

Uma solução $\pi$ do problema é representada por uma dupla $(s, M)$, em que $s$ é um vetor de estágios e $M$ é uma matriz de máquinas. Nesta representação, a cada estágio está associado um vetor que contém a sequência de tarefas para processamento, e para cada tarefa está associada uma coluna da matriz $M$ identificando em qual máquina a tarefa é executada em cada estágio. A Figura 2 ilustra uma solução $\pi$ para o problema apresentado no exemplo dado.

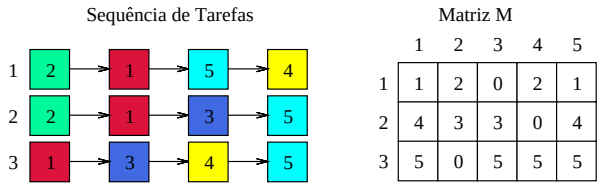


**Figura 2. Representação de uma solução**

Nesta Figura, por exemplo, verifica-se que a sequência de tarefas no primeiro estágio é 2, 1, 5 e 4, ou seja, a tarefa 2 precede a 1, que por sua vez precede a 5 e, finalmente, a tarefa 4 é a última do estágio 1. Verifica-se, também, pela matriz $M$, que a tarefa 1 é executada na máquina 1 no primeiro estágio, na máquina 4 no segundo estágio e na máquina 5 no terceiro estágio. O valor nulo constante na terceira linha e segunda coluna da matriz $M$ indica que a tarefa 2 não é executada no terceiro estágio.

## 4.2 Algoritmo proposto

O algoritmo proposto, chamado `IG-NEH-III`, é baseado no procedimento heurístico *Iterated Greedy* – IG (Ruiz e Stützle, 2005), sendo sua solução inicial gerada por uma adaptação do método NEH (Nawaz et al., 1983). O seu pseudocódigo está esquematizado no Algoritmo 1.

Na linha 2 do Algoritmo 1 é gerada uma solução inicial com o método guloso NEH. Este método funciona como segue. Inicialmente, calcula-se o tempo médio de processamento em cada estágio para cada tarefa, isto é, se uma tarefa $j$ pode passar por duas máquinas no estágio $i$ e, essas máquinas consomem os tempos $p_{ij1}$ e $p_{ij2}$, então a essa tarefa $j$ será atribuído o tempo médio, calculado como $\bar{p}_{ij} = \frac{p_{ij1} + p_{ij2}}{2}$. A seguir, é calculado, para cada tarefa $j$, o somatório desses tempos médios de processamento em todos os estágios, isto é, é calculado o tempo médio de processamento da tarefa $j$, $\bar{p}_j = \sum_i \bar{p}_{ij}$. Na primeira iteração são selecionadas as duas tarefas com maiores $\bar{p}_j$ e realizado um procedimento que procura o melhor sequenciamento possível entre essas duas tarefas. Neste sequenciamento a tarefa escolhida é executada na máquina que possa terminá-la o mais cedo. Este sequenciamento é usado como base para inserir a terceira tarefa. Na segunda iteração é, então, escolhida a terceira tarefa com o maior valor de $\bar{p}_j$ e sequenciada na melhor posição possível. O processo continua até que todas as tarefas tenham sido sequenciadas. Observa-se que esta estratégia de geração de solução inicial gera uma sequência que é a mesma para todos os estágios.

Na linha 3 é aplicado um procedimento de busca local, denominado *Iterative Improvement Insertion* (III), para refinar a solução. Este refinamento, cujo pseudocódigo está mostrado no Algoritmo 2, consiste em selecionar uma tarefa da solução corrente e inseri-la na melhor posição possível do sequenciamento. Se houver melhora na solução corrente, ela é atualizada. Esses passos são repetidos até que todas as tarefas sejam consideradas. A ordem de seleção das tarefas é aleatória.

Da linha 5 à 26, as fases de destruição, reconstrução e aceitação da solução são repetidas até que o critério de parada seja atendido. A destruição (linhas 8 a 10) consiste em retirar $d$ (parâmetro de entrada) tarefas da solução corrente e inseri-las em um vetor $\pi_R$. Para reconstruir a solução (linhas 11 a 13), cada tarefa de $\pi_R$ é inserida na melhor

**Algoritmo 1** : IG-NEH-III

**Entrada:** $d$, *Temperatura*, *criterioParada*

1. **início**
2. $\pi \leftarrow$ construcaoNEH();
3. $\pi \leftarrow$ III($\pi$); {Busca Local}
4. $\pi_b \leftarrow \pi$;
5. **repita**
6. $\pi' \leftarrow \pi$;
7. $\pi_R \leftarrow \emptyset$;
8. **para** $w \leftarrow 1$ **até** $d$ **faça**
9. Retire uma tarefa aleatoriamente de $\pi'$ e a insira em $\pi_R$;
10. **fim**
11. **para** $w \leftarrow 1$ **até** $d$ **faça**
12. Retire a tarefa $\pi_R(w)$ e a insira na melhor posição possivel de $\pi'$;
13. **fim**
14. $\pi' \leftarrow$ III($\pi'$); {Busca Local}
15. **se** $C_{\max}(\pi') < C_{\max}(\pi)$ **então**
16. $\pi \leftarrow \pi'$;
17. **se** $C_{\max}(\pi) < C_{\max}(\pi_b)$ **então**
18. $\pi_b \leftarrow \pi$;
19. **fim**
20. **senão**
21. $pr \leftarrow$ Número real aleatório entre 0 e 1;
22. **se** $pr < e^{-(C_{\max}(\pi')-C_{\max}(\pi))/Temperatura}$ **então**
23. $\pi \leftarrow \pi'$;
24. **fim**
25. **fim**
26. **até** *criterioParada* ser satisfeito;
27. **retorne** $\pi_b$;
28. **fim**



posição possível do sequenciamento. Depois é aplicado novamente o método de busca local descrito no Algoritmo 2. Após isso, se a solução gerada for melhor que a solução corrente, então a solução corrente é atualizada, e se ela for também melhor que a melhor solução, esta é atualizada.

A solução corrente poderá ser atualizada também caso um número real aleatório, entre 0 e 1, for menor que $e^{-\frac{C_{\max}(\pi')-C_{\max}(\pi)}{Temperatura}}$, sendo o valor de *Temperatura* calculado pela equação (1):

$$Temperatura = T \times \frac{\sum_i \sum_j \bar{p}_{ij}}{10 \times n \times m} \quad (1)$$

em que $T$ é um parâmetro de entrada do algoritmo e $n$ e $m$ são a quantidade de tarefas e estágios, respectivamente. Este critério de aceitação é considerado para diversificar as soluções geradas.

**Algoritmo 2** : III

**Entrada:** $\pi$

1. **início**
2. *melhora* $\leftarrow$ *Verdadeiro*;
3. **enquanto** *melhora* **faça**
4. *melhora* $\leftarrow$ *Falso*;
5. **para** $w \leftarrow 1$ **até** $n$ **faça**
6. Retirar, aleatoriamente, uma tarefa $k$ de $\pi$ (sem repetição)
7. $\pi' \leftarrow$ Resultado da inserção da tarefa $k$ na melhor posição possivel de $\pi$;
8. **se** $C_{\max}(\pi') < C_{\max}(\pi)$ **então**
9. $\pi \leftarrow \pi'$;
10. *melhora* $\leftarrow$ *Verdadeiro*;
11. **fim**
12. **fim**
13. **fim**
14. **retorne** $\pi$
15. **fim**

## 5 Resultados

O algoritmo proposto foi implementado em C++, utilizando-se a IDE Borland C++ Builder 6. Os testes foram executados em um computador Intel Core i5-2310, 2.90GHz, com 4 GB de memória RAM, sob sistema operacional Windows 7 64 bits.

Para testá-lo, foram utilizadas 8 famílias de instâncias, de Ruiz et al. (2008), cujas características principais estão mostradas na Tabela 5. Nesta Tabela, $n$ é a quantidade de máquinas, $m$ é o número de estágios e $m_i$ o número de máquinas em cada estágio. Como em cada família há 12 instâncias, há um total de 96 instâncias.

**Tabela 5. Família de Instâncias**

| Família de Instâncias | $n$ | $m$ | $m_i$ |
|---|---|---|---|
| 15_2_1_1 | 15 | 2 | 1 |
| 15_2_3_1 | 15 | 2 | 3 |
| 15_3_1_1 | 15 | 3 | 1 |
| 15_3_3_1 | 15 | 3 | 3 |
| 50_4_2_1 | 50 | 4 | 2 |
| 50_4_4_1 | 50 | 4 | 4 |
| 50_8_2_1 | 50 | 8 | 2 |
| 50_8_4_1 | 50 | 8 | 4 |

Foram consideradas quatro variantes do algoritmo testado, que se diferem entre si pelos valores adotados para seus parâmetros $d$ e $T$, definidos no Algoritmo 1 e equação (1), respectivamente. A Tabela 6 mostra os valores adotados em cada variante.

Inicialmente foram realizadas várias análises de probabilidade empírica (Aiex et al., 2007) em diversas instâncias do problema para verificar qual delas era capaz de alcançar um dado valor alvo mais rapidamente. Em todas elas o comportamento foi o mesmo. A Figura 3 ilustra uma dessas análises. Ela foi gerada a partir da aplicação das variantes do algoritmo a uma instância da família 15_3_3_1, nomeada em Ruiz et al. (2008) por

**Tabela 6. Parâmetros das variantes do algoritmo proposto**

| Variante | $d$ | $T$ |
|---|---|---|
| IG-NEH-III1 | 4 | 0,2 |
| IG-NEH-III2 | 8 | 0,2 |
| IG-NEH-III3 | 4 | 0,5 |
| IG-NEH-III4 | 8 | 0,5 |

`Ism_15_3_3_1-200_0_50_75-125_50-100_-99-99_0-0_3`. O algoritmo foi executado 100 vezes para cada uma das variantes consideradas. Sempre que a função objetivo atingia o valor 1030, que é distante 2% do valor ótimo, a execução era interrompida e o tempo registrado. Esses tempos foram, então, ordenados de forma crescente e, para cada tempo $t_i$ foi associada uma probabilidade acumulada, dada por $p_i = \frac{i - 0,5}{100}$. A seguir, os pontos $(t_i, p_i)$ foram plotados, resultando na Figura 3.

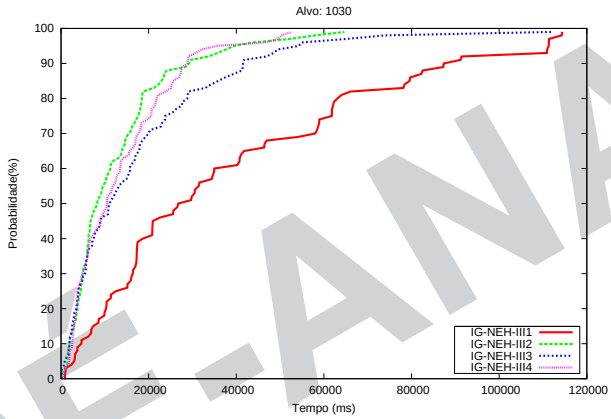


**Figura 3. Análise de Probabilidade Empírica**

O gráfico da Figura 3 mostra que a variante IG-NEH-III4 é a primeira a atingir o alvo desejado com uma probabilidade de quase 100%, alcançando-o em cerca de 500000 milisegundos, enquanto que a variante IG-NEH-III2 o alcança em pouco mais de 600000 ms. As variantes IG-NEH-III1 e IG-NEH-III3 alcançam essa probabilidade em aproximadamente 110000 ms, porém a curva do IG-NEH-III3 se mantém acima da do IG-NEH-III1.

Em vista deste comportamento, a variante IG-NEH-III4 foi escolhida para ser comparada com um algoritmo da literatura, no caso, o Algoritmo Genético de Urlings e Ruiz (2010). Foram feitas 30 execuções do algoritmo proposto para cada uma das 96 instâncias. O critério de parada para esses testes foi o tempo de execução, dado por $50 \times n \times \sum_{i=1}^{m} m_i$ milisegundos, sendo $n$ o número de tarefas e $m_i$ a quantidade de máquinas por estágio. Este critério é o mesmo adotado em Ruiz et al. (2008).

O algoritmo IG-NEH-III4 foi comparado em relação a dois aspectos: 1) Capacidade de encontrar as melhores soluções existentes e 2) Variabilidade das soluções finais. Para avaliar o primeiro aspecto, calculou-se o *gap* das melhores soluções geradas pelo algoritmo para cada grupo de instâncias em relação aos valores ótimos (no caso das instâncias envolvendo 15 tarefas) ou melhores valores da literatura (no caso das instâncias de 50

**Tabela 7. *gap* médio da variante IG-NEH-III4**

| Instância | *gap* do melhor *makespan* | *gap* do *makespan* médio |
|---|---|---|
| 15_2_1_1 | 0,20 | 0,22 |
| 15_2_3_1 | 0,47 | 0,98 |
| 15_3_1_1 | 0,30 | 0,32 |
| 15_3_3_1 | 1,04 | 1,59 |
| 50_4_2_1 | 3,22 | 4,23 |
| 50_4_4_1 | **-3,06** | **-1,72** |
| 50_8_2_1 | **-4,44** | **-2,94** |
| 50_8_4_1 | **-12,02** | **-10,47** |
| Média | **-1,79** | **-0,97** |

tarefas). Para avaliar o segundo aspecto, em lugar dos melhores valores, são calculados os valores médios encontrados pelo algoritmo.

A Tabela 7 mostra os resultados da comparação da variante IG-NEH-III4 do algoritmo proposto com os do Algoritmo Genético de Urlings e Ruiz (2010), com relação aos dois aspectos. Na primeira coluna dessa Tabela são apresentados os conjuntos de instâncias; na segunda coluna é apresentado para cada conjunto de instâncias o *gap* dos melhores valores para o *makespan* encontrados nas 30 execuções do algoritmo IG-NEH-III4 e, finalmente, na última coluna, o *gap* dos valores médios do *makespan*. Destaca-se que neste último caso, o *gap* é em relação à melhor solução conhecida na literatura, disponibilizada em Urlings e Ruiz (2010). Os valores destacados em negrito indicam que o algoritmo proposto superou o algoritmo da literatura.

Alguns detalhes dos experimentos são apontados a seguir. Em cerca de 48% das instâncias de 15 tarefas, o algoritmo proposto conseguiu alcançar o valor ótimo em todas execuções, e em cerca de 58%, esse valor foi atingido pelo menos uma vez. Já para as instâncias de 50 tarefas, em cerca de 35% delas o algoritmo proposto encontrou melhores resultados que o da literatura em todas as execuções, e em aproximadamente 65%, os resultados da literatura foram superados pelo menos uma vez. Dos 48 ótimos globais conhecidos, o algoritmo desenvolvido foi capaz de encontrar 28 deles e aqueles em que ele não encontrou o ótimo conhecido, o *gap* foi de 1,11%. Nos 48 outros problemas-teste em que o ótimo global não era conhecido, o algoritmo IG-NEH-III4 produziu soluções, em média, 4,07% melhores, tendo em vista o *gap* da melhor solução obtida pelo algoritmo em relação à melhor solução conhecida até então.

## 6 Conclusões e trabalhos futuros

Este trabalho tratou o problema de sequenciamento *flowshop* híbrido e flexível, com o objetivo de minimizar o *makespan*. Em vista de sua complexidade, ele foi resolvido por meio de um algoritmo heurístico baseado em *Iterated Greedy*.

Nesse algoritmo, a solução inicial é gerada por uma adaptação do método NEH e refinada pela busca local denominada *Iterative Improvement Insertion* (III). Para geração de novas soluções são executados iterativamente os passos de destruição, reconstrução e busca local. É implementado um critério de aceitação no intuito de diversificar as soluções encontradas.

Quatro variantes do algoritmo proposto, que se diferem pelos parâmetros de entrada do algoritmo, foram testadas. A variante nomeada IG-NEH-III4, que fixa os parâmetros $d$ e $T$ em 8 e 0,5, foi a que teve melhor desempenho nos testes de distribuição de probabilidade empírica, em que se buscava alcançar um valor alvo o mais rapidamente possível.

Em seguida, os resultados dessa variante foram comparados com os de um Algoritmo Genético da literatura, tendo-se por base 96 problemas-teste encontrados na literatura. Os resultados mostraram que o algoritmo proposto teve um desempenho melhor nas instâncias maiores, tendo conseguido melhorar um número significativo de resultados da literatura. Nessas instâncias, os resultados encontrados foram, em média, 4,07% melhores em relação ao *gap* dos valores das melhores soluções conhecidas.

Como os trabalhos futuros propõe-se desenvolver outros tipos de busca local, bem como testar o desempenho do algoritmo em problemas-teste de maior porte.

## Agradecimentos

Os autores agradecem às agências CAPES, FAPEMIG e CNPq, bem como ao CEFET-MG, pelo apoio ao desenvolvimento deste trabalho.

## Referências

Aiex, R.; Resende, M. e Ribeiro, C. (2007). Tttplots: a perl program to create time-to-target plots. *Optimization Letters*, v. 1, p. 355–366.

Burtseva, L.; Romero, R.; Ramirez, S.; Yaurima, V.; González-Navarro, F.F. e Perez, P.F. (2012). *Lot processing in Hybrid Flow Shop scheduling problem*, p. 65–96. Production Scheduling. InTech.

Defersha, F. Melaku e Chen, M. (2011). Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem. *International Journal of Advanced Manufacturing Technology*, v. 57, p. 1–17.

Defersha, F.M. (2010). A comprehensive mathematical model for hybrid flexible flowshop lot streaming problem. *International Journal of Industrial Engineering Computations*, v. 2, p. 283–294.

Garey, M.R. e Johnson, D.S. (1979). *Computers and intractability: a guide to the theory of NP-completness*. Freeman, San Francisco.

Naderi, B.; Ruiz, R. e Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. *Computers & OperationsResearch*, v. 37, p. 236–246.

Nawaz, M.; Jr, E.E. Enscore e Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *The International Journal of Management Science*, v. 11, p. 91–95.

Nishi, T.; Hiranaka, Y. e Inuiguchi, M. (2010). Lagrangian relaxation with cut generation for hybrid flowshop scheduling problems to minimize the total weighted tardiness. *Computers & Operations Research*, v. 37, p. 189–198.

Pacciarelli, D. e D'Ariano, A. (2011). Increasing the reliability of production schedules in a pharmaceutical packaging department. *Journal of Medical Systems*, v. 35, p. 1–15.

Ribas, I.; Companys, R. e Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, v. 39, p. 293–301.

Ribas, I.; Leisten, R. e Framiñan, J.M. (2010). Review and classification of hybrid flowshop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, v. 37, p. 1439–1454.

Ruiz, R.; Sivrikaya, F. e Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, v. 35, p. 1151–1175.

Ruiz, R. e Stützle, T. (2005). An iterated greedy algorithm for the flowshop problem with sequence dependent setup times. *The 6th Metaheuristics International Conference*, p. 817–826, (2005).

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

Ruiz, R. e Vázquez-Rodríguez, J.A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*, v. 205, p. 1–18.

Salvador, M.S. (1973). A solution to a special class of flowshop scheduling problems. *Symposium on the theory of scheduling and its applications*, p. 83–91. Berlin:Springer, (1973).

Urlings, T. e Ruiz, R. (2010). Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. *International Journal of Metaheuristics*, v. 1, p. 30–54.

Urlings, T.; Ruiz, R. e Stützle, T. (2010). Shifting representation search for hybrid flexible flowline problems. *European Journal of Operational Research*, v. 207, p. 1086–1095.

Venditti, L.; Pacciarelli, D. e Meloni, C. (2010). A tabu search algorithm for scheduling pharmaceutical packaging operations. *European Journal of Operational Research*, v. 202, p. 538–546.

Yaurima, V.; Burtseva, L. e Tchernykh, A. (2009). Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Computers & Industrial Engineering*, v. 56, p. 1452–1463.

Zandieh, M.; Mozaffari, E. e Gholami, M. (2010). A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. *Journal of Intelligent Manufacturing*, v. 21, p. 731–743.

Ziaeifar, A.; Tavakkoli-Moghaddam, R. e Pichka, K. (2011). Solving a new mathematical model for a hybrid flow shop scheduling problem with a processor assignment by a genetic algorithm. *International Journal of Advanced Manufacturing Technology*, v. 56, p. 1–11.