UM ALGORITMO *ITERATED GREEDY SEARCH* APLICADO À MINIMIZAÇÃO DO *MAKESPAN* NO PROBLEMA *FLOWLINE* HÍBRIDO E FLEXÍVEL

Eduardo Camargo de Siqueira[1], **Sergio 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
siqueira@decom.cefetmg.br, sergio@dppg.cefetmg.br

[2]Departamento de Computação, Universidade Federal de Ouro Preto
Campus Universitário, Morro do Cruzeiro
Ouro Preto, Minas Gerais
marcone@iceb.ufop.br

Resumo. *Este trabalho apresenta um algoritmo baseado na metaheurística Iterated Greedy Search para resolução de um problema de sequenciamento Flowshop Híbrido e Flexível. O problema tratado é uma variação do Flowshop Híbrido e do Flowline Flexível, sendo conhecido como* Flowline *Híbrido e Flexível. Nesse problema, um conjunto de tarefas passa por um conjunto de estágios e, para cada estágio, existe um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas podem saltar estágios. O critério de otimização considerado é o de minimizar o maior tempo de conclusão das máquinas, o chamado* makespan. *Os resultados alcançados mostram que o algoritmo implementado é capaz de superar resultados da literatura em um subconjunto de instâncias do problema.*

PALAVRAS-CHAVE: *Sequenciamento de tarefas, Flowline híbrido e flexível, Iterated Greedy Search, Flowshop, Makespan.*

Abstract. *This work presents an algorithm based on the Iterated Greedy Search metaheuristic for solving a hybrid flexible flowshop scheduling problem. This problem, called hybrid and flexible flowline, is a variation of the hybrid flowshop and of the flexible flowline. In this problem, a set of jobs is submitted to a set of stages and, for each stage, there is a set of unrelated parallel machines. Some jobs not need to submitted to all stages. The goal is to minimize the makespan. The results show that the implemented algorithm is able to overcome literature results in a subset of instances of the problem.*

KEYWORDS: *Scheduling, Hybrid and Flexible Flowline, Iterated Greedy Search, Flowshop, Makespan.*

1 Introdução

Os problemas de sequenciamento são comuns em diversas áreas do conhecimento, principalmente na produção industrial. Nesse contexto, consistem em definir uma sequência de tarefas a serem executadas em um conjunto de máquinas atendendo a um certo

objetivo, como, por exemplo a redução de custos ou o aumento na capacidade de produção ou estoque.

O primeiro estudo sobre sequenciamento de tarefas é apresentado em Johnson (1954). Desde então, esse tema tem atraído um grande interesse no meio científico. Existem diversos estudos a respeito dos problemas de sequenciamento; no entanto, sempre existiu uma distância entre a teoria desenvolvida e a realidade prática. Por outro lado, há, segundo Ruiz et al. (2008), uma tendência em desenvolver abordagens que se aproximem cada mais de problemas reais. Contudo, ainda segundo Ruiz et al. (2008), tem havido pouco esforço para o desenvolvimento de modelos mais completos, que combinem casos reais.

De forma a reduzir essa lacuna, este trabalho considera um problema de sequenciamento com características muito comuns em problemas reais, como os tempos de *release* para máquinas; a existência de máquinas paralelas não-relacionadas em cada estágio; tempos de *setup* dependentes da sequência; elegibilidade de máquinas; e tempos de atraso (*lag*) entre as operações.

Esse problema, descrito em Ruiz et al. (2008), trata de 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. Esta variante, denominada *Flowline* Híbrido e Flexível (HFFL, do inglês *Hybrid Flexible 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 resolvê-lo por meio da metaheurística *Iterated Greedy Search* – IGS (Ruiz e Stützle, 2005), face ao seu sucesso na resolução de vários outros problemas de natureza combinatória.

O restante desse trabalho está dividido da seguinte forma. Na seção 2 é feito um levantamento bibliográfico dos problemas de *flowshop*. Na seção 3 o problema é apresentado e exemplificado. O algoritmo proposto 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

2.1 Problemas de *Flowshop* Híbrido

Segundo Ribas et al. (2010), os trabalhos de pesquisa 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).

Boudhar e Meziani (2010) propõem a aplicação de heurísticas construtivas para um problema de HFS com dois estágios e recirculação. Carpov et al. (2012) também propõem aplicação de heurísticas para o problema de HFS com dois estágios; nesse caso, existem máquinas paralelas não-relacionadas no segundo estágio. O objetivo, nos dois trabalhos, é

a minimização do *makespan*.

Ruiz e Vázquez-Rodríguez (2010) fazem uma revisão de algoritmos exatos, heurísticas e metaheurísticas para o HFS. Nishi et al. (2010) apresentam um método de relaxação lagrangeana com geração de cortes para esse problema, tendo como objetivo minimizar o somatório dos atrasos ponderados. Azzi et al. (2011) utilizam um algoritmo baseado em uma estratégia de agregação e separação em um problema de HFS, com o objetivo de minimizar o *makespan* e aumentar a capacidade produtiva das máquinas.

Ziaeifar et al. (2011) apresentam uma nova modelagem matemática para o HFS e Yaurima et al. (2009) modelaram um problema de HFS em linhas de montagem de placas de circuito. Nesses dois trabalhos foram utilizados Algoritmos Genéticos, com o objetivo de minimizar o *makespan* nos problemas propostos.

Tadayon e Salmasi (2012a) propõem um algoritmo heurístico para resolução de um problema HFS com elegibilidade de máquinas. Os objetivos a serem otimizados são a minimização do somatório de *completion time* dos grupos de tarefas e a minimização da soma das diferenças entre o tempo de conclusão e o tempo de entrega desses grupos. Tadayon e Salmasi (2012b) utilizam um algoritmo baseado em *Particle Swarm Optimization* (PSO) para a resolução desse problema.

2.2 Problemas de *Flowline* Híbrido e Flexível

Ruiz et al. (2008) apresenta um modelo matemático para um novo problema de sequenciamento *flowshop* híbrido e flexível, denominado *Flowline* Híbrido e Flexível (HFFL, do inglês *Hybrid and Flexible Flowline*). Com este modelo, foi possível resolver, na otimalidade, o HFFL proposto, em problemas testes de até 15 tarefas, 3 estágios e 2 máquinas em cada estágio. Nesse trabalho, também foram apresentadas heurísticas de construção para resolução de problemas de maior porte, de 50 e 100 tarefas.

Naderi et al. (2010) chamam 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*.

Urlings e Ruiz (2010) e Zandieh et al. (2010) também propõem Algoritmos Genéticos para resolução do HFFL. O objetivo considerado, nesses dois últimos trabalhos, é a minimização do *makespan*.

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 encontrados foram avaliados e comparados com outros resultados da literatura.

3 Descrição do Problema Flowline Híbrido e Flexível

O problema considerado neste trabalho é o *Flowline* Híbrido e Flexível (HFFL). Neste problema, tem-se 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, tem-se um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas podem saltar estágios. Seguem abaixo características que definem o problema:

(i) F_j: conjunto de estágios que a tarefa j visita, sendo $1 < F_j < m$;

(ii) p_{ilj}: tempo de processamento da tarefa j na máquina l no estágio i.
(iii) rm_{il}: tempo de *release* da máquina l no estágio i. Representa o tempo de início dos processos na máquina, de modo que nenhuma tarefa pode iniciar na máquina antes do tempo de *release*.
(iv) E_{ij}: conjunto de máquinas elegíveis para a tarefa j no estágio i.
(v) lag_{ilj}: tempo de latência, entre o fim da tarefa j na máquina l do estágio i e o início do próximo estágio da tarefa j.
(vi) 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.

O objetivo a ser considerado nesse trabalho será a minimização do *makespan*, ou seja, o tempo de término da última da tarefa do sistema.

Para exemplificar o problema, seja uma instância com cinco tarefas e dois estágios, e três máquinas em cada estágio. A Tabela 1 mostra quais máquinas são elegíveis 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 2 e 3, no estágio 1, e na máquina 4, no estágio 2. Pode-se verificar, também, que as tarefas 1, 2 e 5 visitam todos os estágios, enquanto as tarefas 3 e 4 pulam os estágios 1 e 2, respectivamente.

Tabela 1. Elegibilidade

	i	1	2
j	1	{2,3}	{4}
	2	{2,3}	{4,5,6}
	3	-	{5}
	4	{1,3}	-
	5	{1,2}	{4,5,6}

A Tabela 2 contém o tempo de *release* para cada máquina. A linha rm_{il} indica os tempos de *release* para cada máquina, e cada célula p_{ilj} os tempos de processamento. Por exemplo, o tempo de *release* da máquina 1 é igual a 20 e o da máquina 2 é igual a 5.

Tabela 2. Tempo de *release*

i	1			2		
l	1	2	3	4	5	6
rm_{il}	20	5	33	38	29	45

A Tabela 3 mostra o tempo de processamento de cada tarefa em cada máquina, enquanto a Tabela 4 mostra os tempos de latência. Nessas duas tabelas, j indica uma tarefa, a linha i representa cada um dos 2 estágios, e a linha l representa cada uma das 5 máquinas. Nessas tabelas, pode-se verificar que o tempo de processamento da tarefa 1 na máquina 2 é de 8, e o tempo de latência é nulo, enquanto a mesma tarefa na máquina 3 tem um tempo de processamento de 13 e tempo de latência igual a 9. O tempo de processamento de uma tarefa em uma máquina não elegível é nulo.

A Tabela 5 mostra os tempos de preparação (tempos de *setup*), que são dependentes da sequência, e seus valores binários associados. Esse valor binário indica se o *setup* é antecipativo (1), ou não (0). Nessa Tabela, j e k indicam uma tarefa, a linha i representa

Tabela 3. Tempo de processamento

i		1			2		
l		1	2	3	4	5	6
j	1	0	8	13	21	0	0
	2	0	15	15	45	46	44
	3	0	0	0	0	13	0
	4	12	0	32	0	0	0
	5	18	16	0	16	19	36

Tabela 4. Tempo de latência

i		1		
l		1	2	3
j	1	0	0	9
	2	0	0	8
	3	0	0	0
	4	2	0	-6
	5	0	-5	0

cada um dos 2 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 3 do primeiro estágio é igual a 10 e o valor binário igual a 0 (não-antecipativo), porém o *setup* nas máquinas 1 e 2 não existe pois essas máquinas não são elegíveis para a tarefa 4.

Tabela 5. Tempos de *setup* e valores A_{iljk}

i		1				
k		1	2	3	4	5
j	1	-,-,-	-,11(0),5(0)	-,-,-	-,-,10(0)	-,38(0),-
	2	-,35(0),10(0)	-,-,-	-,-,-	-,-,38(0)	-,22(0),-
	3	-,-,-	-,-,-	-,-,-	-,-,-	-,-,-
	4	-,-,46(0)	-,-,21(0)	-,-,-	-,-,-	43(0),-,-
	5	-,6(0),-	-,23(0),-	-,-,-	25(0),-,-	-,-,-
i		2				
j	1	-,-,-	33(1),-,-	-,-,-	-,-,-	29(0),-,-
	2	30(1),-,-	-,-,-	-,6(1),-	-,-,-	8(1),39(1),27(1)
	3	-,-,-	-,45(1),-	-,-,-	-,-,-	-,30(0),-
	4	-,-,-	-,-,-	-,-,-	-,-,-	-,-,-
	5	41(0),-,-	47(0),7(0),38(1)	-,48(0),-	-,-,-	-,-,-

A Figura 1 ilustra um possível sequenciamento para esse exemplo. Note que o *makespan* para esse sequenciamento é igual a 107.

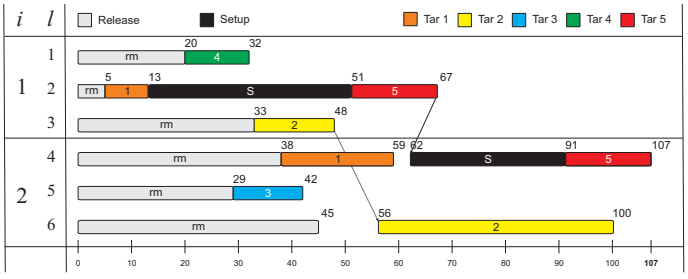

Figura 1. Diagrama de GANTT. $C_{max} = 107$

4 Metodologia

4.1 Representação Permutacional de uma solução

Uma solução é representada por uma dupla (π, M), em que π é a sequência de tarefas para processamento e M é uma matriz de máquinas. Nesta representação, todos os

estágios contém a mesma sequência de tarefas (π), chamada representação permutacional, e, para cada tarefa, está associada uma coluna da matriz M, identificando em qual máquina a tarefa é executada em cada estágio. Caso a tarefa não seja executada em dado estágio, é atribuído o valor nulo. A Figura 2 ilustra a dupla (π, M) para a solução do exemplo considerado na Fig. 1.

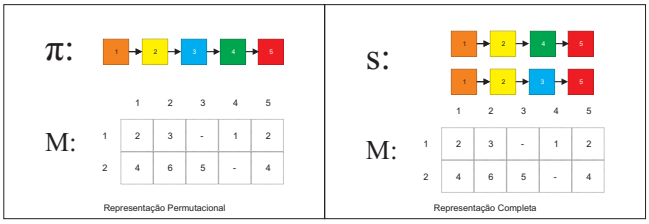

Figura 2. Representações Permutacional e Completa de uma solução.

Na Figura 2 verifica-se, por exemplo, que a sequência de tarefas é 1, 2, 3, 4 e 5, ou seja, a tarefa 1 precede a 2, que, por sua vez, precede a 3, e assim por diante. Verifica-se, também, pela matriz M, que a tarefa 1 é executada na máquina 2 no primeiro estágio e na máquina 4 no segundo estágio. O valor nulo constante na primeira linha e terceira coluna da matriz M indica que a tarefa 3 não é executada no primeiro estágio.

4.2 Representação Completa de uma solução

Uma solução Sol do problema é representada por uma dupla (s, M). Nesta representação, s é um vetor de estágios e, para cada estágio, está associado um vetor, que contém a sequência de tarefas para processamento. Além disso, M é uma matriz usada para identificar em qual máquina a tarefa é executada em cada estágio.

A Figura 2 ilustra uma solução Sol para o problema apresentado no exemplo dado na Figura 1. Na representação de sequência de tarefas, cada linha representada um estágio. Na matriz M, cada linha representa um estágio e cada coluna representa uma tarefa.

Na Figura 2 verifica-se que a sequência de tarefas no primeiro estágio é 1, 2, 4 e 5, ou seja, a tarefa 1 precede a 2, que, por sua vez, precede a 4 e, finalmente, a tarefa 5 é a última do estágio 1. Verifica-se, também, pela matriz M, que a tarefa 1 é executada na máquina 2 no primeiro estágio e na máquina 4 no segundo estágio. O valor nulo constante na primeira linha e terceira coluna da matriz M indica que a tarefa 3 não é executada no primeiro estágio.

4.3 Construção

A heurística NEH (Heurística de Nawaz-Encore-Ham), proposta em Nawaz et al. (1983) para resolução de um problema *flowshop*, é um método de construção guloso. Ela foi adaptada para o problema estudado no presente trabalho da forma como segue.

Inicialmente, calcula-se o tempo médio de processamento $\bar{p}_{ij}$ em cada estágio i para cada tarefa j, isto é, se uma tarefa j pode passar por duas máquinas no estágio i e essas máquinas consomem os períodos de tempos p_{ij1} e p_{ij2}, então, a essa tarefa j é atribuído o tempo médio, calculado pela expressão:

$$\bar{p}_{ij} = \frac{p_{ij1} + p_{ij2}}{2} \tag{1}$$

A seguir, é calculado, para cada tarefa j, o somatório desses tempos médios de processamento em todos os estágios, isto é, é calculado o período de tempo médio $\bar{p}_j$ da tarefa j, dado por:

$$\bar{p}_j = \sum_i \bar{p}_{ij} \tag{2}$$

Na primeira iteração, são selecionadas as duas tarefas com os maiores valores $\bar{p}_j$ e realizado um procedimento que procura o melhor sequenciamento possível entre essas duas tarefas. Nesse sequenciamento, a tarefa escolhida é executada na máquina que possa terminá-la o mais cedo. Esse sequenciamento é usado como base para inserir a terceira tarefa. Na segunda iteração é, então, escolhida a terceira tarefa com maior $\bar{p}_j$, que é 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.

4.4 Busca Local com Representação Permutacional

Uma heurística de Busca Local consiste em, a partir de uma solução inicial, caminhar pelo espaço de soluções, no intuito de refinar a solução corrente. Neste trabalho, utilizou-se o procedimento de busca local, proposto em Ruiz e Stützle (2005), denominado *Iterative Improvement Insertion* (III).

A busca local III consiste em retirar uma tarefa da solução corrente e reinseri-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 e não ocorra melhora. A ordem de seleção das tarefas é aleatória.

Após o término desse procedimento, a busca local começa a considerar movimentos em blocos, retirando um conjunto de tarefas adjacentes da solução corrente e reinserindo-as (em bloco) na melhor posição possível do sequenciamento.

4.5 Busca Local com Representação Completa

Ao observar o sequenciamento, verifica-se que o *makespan* é determinado por um caminho crítico. Por exemplo, na Figura 1, nota-se que o início da última operação da tarefa 5 é determinado pelo fim da operação da mesma tarefa no estágio anterior, e o início dessa operação é determinado pelo fim da primeira operação da tarefa 1, formando um caminho de operações. Uma operação é a execução de uma tarefa em um estágio. Quando esse caminho determina o *makespan*, ele é chamado de caminho crítico.

Segundo Urlings et al. (2010), é improvável que haja melhora no valor de *makespan* movendo uma operação que não está no caminho crítico. Portanto, só se aplica a busca local para as operações no caminho crítico.

Assim, o método de busca local se inicia com a operação cujo momento de conclusão é igual ao valor do *makespan* na solução corrente. No exemplo da Figura 1, essa é a tarefa 5 na máquina 4. Essa operação é realocada em todas as posições possíveis de sequenciamento em todas as máquinas elegíveis. Se houver melhora na solução, a solução corrente é atualizada, e o procedimento reinicializado. Não havendo melhora, a próxima operação no caminho crítico é considerada. Essa pode ser a operação do estágio anterior da mesma tarefa ou a operação anterior, de uma outra tarefa, na mesma máquina. Quando duas

ou mais tarefas determinar o *makespan* ou o tempo de início de outra tarefa no caminho crítico, a busca do local é interrompida. Neste caso, o *makespan* não pode ser melhorado por meio da realocação de uma única operação. O procedimento se encerra quando não houver mais tarefas no caminho crítico.

4.6 Algoritmo proposto

O algoritmo proposto, baseado no procedimento metaheurístico *Iterated Greedy Search* – IGS (Ruiz e Stützle, 2005), está esquematizado no Algoritmo 1. A solução inicial é gerada pelo método NEH adaptado, descrito na Subseção 4.3.

Na linha 2 do Algoritmo 1, é gerada uma solução inicial com o método guloso NEH. Na linha 3, é aplicado um procedimento de busca local, denominado *Iterative Improvement Insertion* – III, detalhado na Subseção 4.4, para refinar a solução.

Algoritmo 1 *Iterated Greedy Search* (IGS)

Entrada: d, *Temperatura*, *criterioParada*

1. **início**
2. $\pi \leftarrow$ construcaoNEH();
3. $\pi \leftarrow$ III(π); {Busca Local com Rep. Permutacional}
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 π' e a insira em π_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 π';
13. **fim**
14. $\pi' \leftarrow$ III(π'); {Busca Local com Rep. Permutacional}
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** $pr < exp\{-(C_{\max}(\pi') - C_{\max}(\pi))/Temperatura\}$ **então**
23. $\pi \leftarrow \pi'$;
24. **fim**
25. **fim**
26. **até** *criterioParada* ser satisfeito;
27. $Sol \leftarrow$ BuscaLocal(Sol); {Busca Local com Rep. Completa}
28. **retorne** Sol;
29. **fim**

Da linha 5 à linha 26, as fases de destruição, reconstrução e aceitação da solução são repetidas, até que o critério de parada seja atendido. A fase de destruição (linhas 8 a

10) consiste em retirar d (parâmetro de entrada) tarefas aleatoriamente da solução corrente e inseri-las em um vetor (π_R).

Para reconstruir a solução (linhas 11 a 13), cada tarefa de π_R é inserida na melhor posição possível do sequenciamento. Após, é aplicado novamente o método de busca local III. Em seguida, 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:

$$pr = \exp\left(-\frac{C_{\max}(Sol') - C_{\max}(Sol)}{\textit{Temperatura}}\right)$$

sendo o valor de *Temperatura* calculado pela expressão:

$$\textit{Temperatura} = T \times \frac{\sum_i \sum_j \bar{p}_{ij}}{10 \times n \times m}$$

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

Após o critério de parada ser atingido, é aplicado o método de busca local com representação completa na solução corrente, detalhado na Subseção 4.5. Essa representação é aqui considerada no intuito de explorar melhor o espaço de busca, visto que a representação permutacional, embora alcance bons resultados, restringe o espaço de busca.

4.6.1 Fase de Destruição com Método da Roleta

O Método da Roleta é utilizado para escolher quais tarefas serão retiradas da solução corrente e funciona como segue. Para cada tarefa da solução corrente, é calculado seu índice de aptidão (*fitness*). Assim, as tarefas recebem uma porção na roleta proporcional ao seu índice de aptidão. Por fim, a roleta é girada para se escolher qual tarefa será retirada da solução corrente. O procedimento é repetido um certo número de vezes (parâmetro d).

Como comentado anteriormente na Subseção 4.5, o *makespan* de uma solução é determinado por um caminho crítico. Portanto, o cálculo do índice de aptidão é feito baseado nesse caminho. O *fitness* de uma tarefa é a quantidade de vezes que ela aparece no caminho crítico adicionado de uma unidade. Assim, no exemplo da Figura 1, o *fitness* da tarefa 5 é igual a 3 e o da tarefa 1 é igual a 2. O restante das tarefas possui *fitness* igual a 1.

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 2 Quad, 2.67GHz, com 4 GB de memória RAM, sob sistema operacional Windows 7 32 bits.

Para testá-lo, foram utilizadas 8 famílias de instâncias, advindas de Ruiz et al. (2008), cujas características principais estão mostradas na Tabela 6. 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 6. Família de Instâncias

Família de Instâncias	n	m	m_i
15_2_1	15	2	1
15_2_3	15	2	3
15_3_1	15	3	1
15_3_3	15	3	3
50_4_2	50	4	2
50_4_4	50	4	4
50_8_2	50	8	2
50_8_4	50	8	4

Os valores dos parâmetros d e T para o algoritmo foram definidos como $d = 6$ e $T = 0,5$. Esses valores foram definidos de forma empírica, em uma bateria preliminar de testes.

O algoritmo proposto foi comparado com o melhor resultado da literatura de Urlings e Ruiz (2010) e Urlings et al. (2010) em relação a dois aspectos: (i) Soluções geradas após a construção; e (ii) Soluções geradas após o refinamento. Para avaliar o primeiro aspecto, calculou-se o *gap* das melhores soluções e das soluções médias geradas pelos algoritmos de construção 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 tarefas). Para avaliar o segundo aspecto, calculou-se o *gap* das melhores soluções e das soluções médias nas iterações do algoritmo baseado em IGS.

O algoritmo foi executado 10 vezes para cada problema teste. Toda vez que o algoritmo permanecia 300 iterações seguidas sem melhorar a solução, a execução era interrompida.

As Tabelas 7 e 8 mostram os resultados da comparação do algoritmo proposto com relação aos três aspectos apontados para as instâncias de 15 e 50 tarefas, respectivamente. Na primeira coluna dessas tabelas são apresentados os conjuntos de instâncias; nas colunas "Melhor" são apresentados os valores que se referem ao *gap* dos melhores valores para o *makespan* da variante; nas colunas "Médio" é apresentado o *gap* dos valores médios do *makespan*. Observa-se que, nestas tabelas, um valor negativo de *gap* indica que o algoritmo proposto conseguiu melhorar a solução da literatura. Nas colunas "Tempo", são apresentados os valores médios dos tempos de execução (em milisegundos). Nessas tabelas também são apresentados, na última coluna, o percentual de melhoria proporcionado pela fase de refinamento comparado com o resultado da fase de construção, tanto em relação ao melhor resultado encontrado em cada fase quanto em relação ao resultado médio.

Para as instâncias de 15 tarefas, em média, o algoritmo proposto apresentou *gap* em relação aos melhores valores de 0,63% e, em relação aos valores médios, de 2,69%. Nessas instâncias do problema, em média, a fase de refinamento (perturbação e busca local) conseguiu melhorar em 4,28% em relação às melhores soluções geradas na fase de construção, enquanto que em relação aos valores médios a melhoria foi de 5,70%.

Para as instâncias de 50 tarefas, o algoritmo proposto apresentou uma melhora de 6,25% em relação aos melhores valores da literatura e, em relação aos valores médios, a melhoria foi de 4,20%. Nesses problemas-teste, em média, a fase de refinamento conseguiu melhorar em 7,53% os resultados da fase de construção.

Dos 48 ótimos globais conhecidos para os problemas-teste de 15 tarefas, o algoritmo proposto foi capaz de encontrar 29 deles. Nos 48 problemas-teste de 50 tarefas,

Tabela 7. *Gap* do algoritmo IGS nas instâncias de 15 tarefas

	Construção			Refinamento			Melhoria	
Instância	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio
15_2_1	2,86	2,86	6,28	0,14	0,18	8743,84	1,48	1,99
15_2_3	8,56	13,85	6,37	1,08	5,53	6296,56	8,17	10,20
15_3_1	3,94	3,94	6,86	0,24	0,50	5132,84	2,83	2,79
15_3_3	10,80	15,91	5,84	1,08	4,54	5241,91	4,64	7,83
Média	6,54	9,39	-	0,63	2,69	-	4,28	5,70

Tabela 8. *Gap* do algoritmo IGS nas instâncias de 50 tarefas

	Construção			Refinamento			Melhoria	
Instância	Melhor	Médio	Tempo(ms)	Melhor	Médio	Tempo(ms)	Melhor	Médio
50_4_2	8,64	11,48	250,01	1,33	3,26	306051,88	5,86	6.55
50_4_4	5,19	8,33	221,97	**-4,59**	**-2,56**	282171,49	7,46	8.52
50_8_2	3,76	5,59	390,91	**-6,61**	**-4,57**	627214,15	8,22	8.04
50_8_4	3,01	6,15	472,03	**-15,12**	**-12,94**	612908,09	8,58	9.98
Média	5,93	7,10	-	**-6,25**	**-4,20**	-	7,53	8,27

cujas soluções ótimas não são conhecidas, o algoritmo proposto conseguiu melhorar os resultados da literatura em 39 deles.

6 Conclusões e trabalhos futuros

Este trabalho tratou o problema de sequenciamento *flowshop* híbrido e flexível, tendo como objetivo minimizar o *makespan*. Em vista de sua complexidade, ele foi resolvido por meio de um algoritmo baseado na metaheurística *Iterated Greedy Search*.

No algoritmo proposto, as soluções iniciais são geradas pela aplicação de uma adaptação da regra de avaliação NEH. O principal mecanismo de busca no espaço de soluções do algoritmo desenvolvido são as fases de destruição e reconstrução. As soluções geradas são refinadas por um procedimento de busca local baseado no método *Iterative Improvement Insertion* (III). Por fim, a solução final é refinada por um método de busca local com representação completa.

Como trabalhos futuros, propõe-se testar o desempenho do algoritmo em problemas-teste de maior dimensão, bem como desenvolver outras técnicas de exploração do espaço de busca.

7 Agradecimentos

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

Referências

Azzi, A.; Faccio, M.; Persona, A. e Sgarbossa, F. (2011). Lot splitting scheduling procedure for makespan reduction and machine capacity increase in a hybrid flow shop with batch production. *International Journal of Advanced Manufacturing Technology*, v. 59, p. 775–786.

Boudhar, M. e Meziani, N. (2010). Two-stage hybrid flow shop with recirculation. *International Transactions in Operational Research*, v. 17, p. 239–255.

Carpov, S.; Carlier, J.; Nace, D. e Sirdey, R. (2012). Two-stage hybrid flowshop with precedence constraints and parallel machines at second stage. *Computers & Operations Research*, v. 39, p. 736–745.

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.

Johnson, S.M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, v. 1, p. 61–68.

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.

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 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).

Tadayon, B. e Salmasi, N. (2012)a. A flexible flowshop scheduling problem with machine eligibility constraint and two criteria objective function. *Engineering and Technology*, v. 62, p. 103–108.

Tadayon, B. e Salmasi, N. (2012)b. A two-criteria objective function flexible flowshop scheduling problem with machine eligibility constraint. *International Journal of Advanced Manufacturing Technology*, v. 60, p. 1–15.

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.

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.