HEURÍSTICAS CONSTRUTIVAS PARA PROGRAMAÇÃO DE *FLEXIBLE FLOW LINE* COM TEMPOS DE *SETUP* INDEPENDENTES DA SEQUÊNCIA

Hélio Yochihiro Fuchigami
Departamento de Matemática / Matemática Industrial
Universidade Federal de Goiás – Campus Catalão
Av. Dr. Lamartine Pinto de Avelar, 1120, CEP 75704-020, Catalão/GO
heliofuchigami@yahoo.com.br

João Vitor Moccellin
Departamento de Engenharia Mecânica e de Produção
Universidade Federal do Ceará – Campus do Pici
Av. da Universidade, 2853, CEP 60020-181, Fortaleza/CE
jvmoccel@sc.usp.br

RESUMO

Neste trabalho foram propostas seis heurísticas construtivas para programação do sistema *flexible flow line* com tempos de setup independentes da sequência e minimização do *makespan*. Este é um ambiente de produção multiestágio em que pode haver uma ou mais máquinas paralelas em cada estágio. Além disso, as tarefas podem saltar estágios e os tempos de setup podem ser antecipados ou não, de acordo com uma probabilidade. Os resultados foram comparados com as heurísticas de Fuchigami, Moccellin e Ruiz (2007) e avaliados por meio da porcentagem de sucesso, desvio relativo médio, desvio-padrão do desvio relativo e tempo médio de computação.

PALAVARAS CHAVE. Programação da produção, *Flexible flow line*, Tempos de *setup*.

AD&GP

ABSTRACT

This paper presents six constructive heuristic algorithms for scheduling on flexible flow line with sequence-independent setup times. In this multistage production system, may there be one or more identical parallel machines at each stage. Moreover, jobs can skip stages and the setup times may be anticipatory or non-anticipatory, according a probability. Results were compared with Fuchigami, Moccellin and Ruiz (2007)'s heuristics and evaluated by percentage of success (in finding the best solution), relative deviation, standard deviation of relative deviation and CPU cost.

KEYWORDS. Scheduling. Flexible flow line. Setup time.

AD&GP

1. Introdução

A "programação da produção" (*scheduling*) é um processo de tomada de decisão utilizado em muitas indústrias de manufatura e serviços que trata da alocação de recursos a tarefas em um período de tempo considerado, objetivando otimizar uma ou mais medidas de desempenho (PINEDO, 2008).

Este trabalho trata do problema de programação em sistemas *flexible flow line* com tempos de *setup* separados dos tempos de processamento das tarefas e independentes da sua sequência de execução. A medida de desempenho considerada é o *makespan*, ou seja, a duração total da programação.

O ambiente *flexible flow line* é uma generalização do *flow shop* híbrido ou *flow shop* com múltiplas máquinas, removendo a restrição de que as tarefas precisam passar por todos os estágios. Assim, o problema tratado é multiestágio com fluxo unidirecional onde as tarefas podem saltar estágios. Em cada estágio pode haver uma ou mais máquinas paralelas idênticas.

A configuração do *flexible flow line* pode ser encontrada em um vasto número de indústrias: química, eletrônica, de empacotamento, farmacêutica, automotiva, fabricação de embalagens de vidro, madeireira, têxtil, herbicidas, alimentícia, cosméticos e de semicondutores (QUADT; KUHN, 2007).

A Figura 1 ilustra um *flexible flow line* com g estágios de produção e m_k máquinas no estágio k. Tipicamente, estoques intermediários (*buffers*) são representados entre os estágios para armazenamento temporário das tarefas.

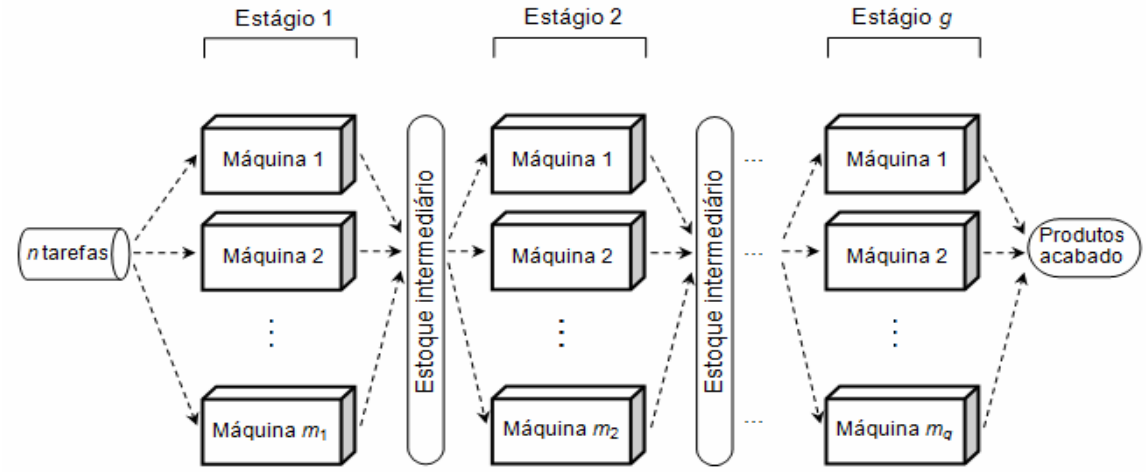

FIGURA 1 – Esquema ilustrativo de um *flexible flow line*

As operações de *setup* incluem todo trabalho de preparação da máquina ou da oficina para a fabricação dos produtos, como a obtenção e ajuste de ferramentas, inspeção e posicionamento de materiais e processos de limpeza.

Existem dois tipos de problemas com o tempo de *setup* separado do tempo de processamento das tarefas. No primeiro, o *setup* depende somente da tarefa a ser processada e é chamado "independente da sequência". E no segundo caso, o *setup* depende tanto da tarefa a ser processada como também da que foi executada imediatamente antes na mesma máquina, sendo chamado "dependente da sequência" (ALLAHVERDI, GUPTA e ALDOWAISAN, 1999). Neste trabalho, foi considerado o ambiente em que os tempos de *setup* são independentes da sequência de execução das tarefas.

Além disso, é mais realista considerar que alguns tipos de *setup* podem ser realizados antecipadamente, ou seja, antes da liberação da tarefa no estágio anterior. Como outros tipos de *setup* podem requer que o produto esteja presente na máquina onde será processado, como por exemplo operações de ajuste da peça, ambas as possibilidades foram consideradas neste trabalho:

setup antecipado e não antecipado.

Segundo a conhecida notação de três campos e as adaptações de Vignier, Billaut e Proust (1999), Allahverdi, Gupta e Aldowaisan (1999) e Lin e Cheng (2005), o problema estudado pode ser representado por $FFSg,((PM^{(k)})_{k=1}^{g}) \mid s_{jk}, as, ns \mid C_{\max}$.

O problema de programação em ambientes *flexible flow line* é *NP-hard* para todos os critérios de otimização tradicionais, mesmo quando o *setup* não é considerado explicitamente (QUADT; KUHN, 2007). Pela complexidade intrínseca ao problema, um dos principais objetivos deste trabalho foi desenvolver um conjunto de heurísticas construtivas eficazes e computacionalmente eficientes.

2. Revisão bibliográfica

Muitas publicações já abordaram o problema da programação em *flow shops* híbridos. Linn e Zhang (1999) propuseram uma classificação das pesquisas neste ambiente em três categorias: problemas com dois estágios, três estágios e mais de estágios. Vignier, Billaut e Proust (1999) apresentaram para aquele momento o estado da arte de *flow shops* híbridos. Em geral as pesquisas se restringiram a situações ou configurações específicas de máquinas nos estágios, ou sem a presença de tempos de *setup* separados dos tempos de processamento.

Kis e Pesch (2005) atualizaram o estado da arte com trabalhos posteriores a 1999, enfocando métodos de solução exata para minimização do *makespan* e tempo médio de fluxo. Wang (2005) fez uma revisão da literatura, classificando em métodos de solução ótima, heurística e de inteligência artificial. Quadt e Khun (2007) publicaram uma taxonomia para *flow shop* híbridos, porém denominando-o de *flexible flow line*.

Como observaram Quadt e Kuhn (2007), o sistema *flow shop* híbrido pode ser encontrado em um vasto número de indústrias, como química, eletrônica, de empacotamento, farmacêutica, automotiva, fabricação de embalagens de vidro, madeireira, têxtil, de herbicidas, alimentícia, de cosméticos e de semicondutores.

Estudos envolvendo tempos de *setup* dependentes da sequência de tarefas foram realizados por Kurz e Askin (2003), Lin e Liao (2003), Kurz e Askin (2004), Fuchigami e Moccellin (2005), Ruiz e Maroto (2006) e Ruiz, Şerifoğlu e Urlings (2008).

Os trabalhos a seguir consideraram os tempos de *setup* independentes da sequência de execução das tarefas, que é o foco do presente trabalho.

O ambiente *flow shop* híbrido com dois estágios, sendo uma única máquina no primeiro e várias máquinas paralelas idênticas no segundo, com o critério de minimização do *makespan*, foi estudado por Gupta e Tunc (1994), Li (1997) e Huang e Li (1998).

Botta-Genoulaz (2000) estudou o problema de minimização do atraso máximo das tarefas sujeito ao mínimo tempo de transporte. Há também a presença de tempos de remoção das tarefas. O autor propôs e avaliou o desempenho de seis heurísticas. Allaoui e Artiba (2004) analisaram o problema *flexible flow shop* com várias medidas de desempenho, entre elas a minimização do *makespan* e do atraso máximo, considerando tempos de transporte.

O problema de programação em indústria de móveis foi estudado por Wilson, King e Hodgson (2004). Em cada estágio há várias máquinas paralelas idênticas. As heurísticas desenvolvidas baseiam-se em um algoritmo genético e demonstraram eficiência na minimização do tempo de *setup* independente da sequência e do *makespan*.

Low (2005) enfocou a minimização do tempo total de fluxo em problemas em que os estágios possuíam várias máquinas paralelas não relacionadas e tempos de remoção. Heurísticas construtivas para minimização do *makespan* foram apresentadas por Logendran, Carson e Hanson (2005). As tarefas foram agrupadas em famílias. Foram considerados os tempos de *setup* dependentes das máquinas e independentes das famílias de tarefas.

Fuchigami, Moccellin e Ruiz (2007) analisaram o desempenho de regras de prioridade em sistemas *flexible flow line*. Os tempos de *setup* independentes da sequência podem ser antecipados ou não com determinada probabilidade. Fuchigami (2010) abordou dois problemas relacionados (*flexible flow line* com *setup* dependente e independente da sequência), propondo para cada um deles dois grupos de métodos de solução: regras de prioridade e heurísticas

construtivas. O estudo destaca também as características do ambiente de produção mais influentes nos resultados dos métodos.

Como pode ser observado, embora o ambiente *flow shop* híbrido tenha sido extensivamente estudado há muitos anos, existem poucas publicações com os sistemas *flexible flow line*. O único trabalho que aborda o mesmo ambiente desta pesquisa foi o de Fuchigami, Moccellin e Ruiz (2007), que será utilizado para comparação.

3. Métodos heurísticos construtivos propostos

O problema consiste em programar um conjunto de n tarefas, definido como $J = \{1, ..., n\}$, que será processado em um conjunto de g estágios. Em cada estágio k, com $k = 1,..., g$, existe um conjunto de M_k $(1, \ldots, m_k)$ máquinas paralelas idênticas, onde $m_k \geq 1$, que estão disponíveis e podem processar quaisquer tarefas, desde que liberadas. Pelo menos um dos estágios possui mais de uma máquina.

Todas as tarefas têm o mesmo fluxo, passando pelos estágios na mesma ordem e sendo processadas por no máximo uma máquina em cada estágio. As tarefas podem saltar estágios com uma determinada probabilidade. O tempo de processamento da tarefa j, $j \in J$, em cada estágio k é indicado por p_{jk}. O tempo de *setup* das máquinas do estágio k é representado por s_{jk}, antecipado ou não de acordo com um intervalo de probabilidade. A medida de desempenho utilizada é a minimização do *makespan*.

Foram propostos seis métodos heurísticos construtivos, apresentados a seguir. Com exceção da heurística Hi4, todos seguem a classificação de métodos de decomposição por estágio apresentada por Quadt e Kuhn (2007). Cada subdivisão do problema é composta por um único estágio, ou seja, um subproblema de máquina única ou de máquinas paralelas. Assim há uma evidente redução da complexidade em relação ao problema original.

Cada estágio é composto por um subconjunto de tarefas que o visitam ($V_k \subseteq J$) e é programado individual e consecutivamente. A interdependência entre os estágios está no fato da data de término das tarefas de um estágio (C_{jk}) equivaler à data de liberação (r_{jk}) das mesmas no estágio seguinte ($C_{jk} = r_{j(k+1)}$). Assim, do primeiro ao penúltimo estágio, a medida de desempenho sugerida em algumas heurísticas é a de minimização da soma das datas de término das tarefas do estágio, que corresponde ao esforço para antecipar as datas de liberação das tarefas no estágio seguinte ($\min \Sigma C_{jk} \Leftrightarrow \min \Sigma r_{j(k+1)}$), visando minimizar o *makespan* do problema original.

Por outro lado, a heurística Hi4 é classificada como método de decomposição por tarefa, pois a cada iteração uma tarefa é selecionada e associada a uma máquina em cada um dos estágios, ou seja, o sequenciamento e a alocação são feitos simultaneamente.

◆ HEURÍSTICA Hi1

A ideia intrínseca à heurística Hi1 é priorizar a cada estágio as tarefas que terminam sua execução mais cedo, para assim terem a menor data de liberação possível no estágio seguinte, conforme descrito anteriormente. No primeiro estágio, como todas as tarefas já estão liberadas (r_{j1}=0, $\forall j$) e não há *setup* antecipado, alocam-se na máquina de menor carga as tarefas sequenciadas pela regra *SPT* (*Shortest Processing Time* – menor tempo de processamento) considerando a soma $s_{j1}+p_{j1}$. Esta regra poderia ser denotada por *SPST* (*Shortest Processing + Setup Time*).

Com o *setup* independente da sequência e considerando-o somado ao tempo de processamento, o **primeiro estágio** equivale ao problema clássico de máquinas paralelas com minimização do tempo médio de fluxo (denotado por $P \,||\, \overline{F}$ com r_j=0, $\forall j$). O algoritmo de solução ótima para este problema consta em Baker (1974, p.119) e é bem conhecido. Consiste em ordenar as tarefas pela regra *SPT* e em seguida alocar uma tarefa de cada vez à máquina de menor carga.

Nos **estágios posteriores ao primeiro**, como as datas de liberação das tarefas são diferentes de zero, considerar apenas a máquina de menor carga não garante uma boa programação. Portanto, dentre as tarefas que visitam o estágio e que ainda não foram programadas nele, o algoritmo verifica todas as possibilidades de alocação tarefa-máquina e

escolhe a que leve à menor data de término. Tal procedimento constitui uma forma mais abrangente de utilizar a regra *ECT (Earliest Completion Time*). Assim, o Algoritmo Hi1 foi construído da seguinte forma:

Algoritmo Hi1

PASSO 1. No estágio k=1, sequencie as tarefas pela ordem não decrescente da soma (s_{j1}+p_{j1}) e aloque uma tarefa de cada vez à máquina de menor carga. Faça k=2.

PASSO 2. Atualize as datas de liberação do estágio k como as datas de término do estágio k–1.

PASSO 3. Analise todas as possibilidades de alocação tarefa-máquina e escolha a opção com a menor data de término.

PASSO 4. Repita o PASSO 3 até que todas as tarefas que visitam o estágio k estejam programadas.

PASSO 5. Se for o último estágio (k=g), PARE; senão, faça k=k+1 e vá para o PASSO 2.

◆ **HEURÍSTICA Hi2**

A heurística Hi2 consiste numa modificação da Hi1, alterando apenas a programação do último estágio, em que é aplicada uma adaptação do método de inserção denominado *Multiple Insertion*, apresentado por Kurz e Askin (2001). Tal método de inserção é transcrito a seguir.

Algoritmo *Multiple Insertion* (*MI*)

PASSO 1. Crie os tempos de processamento modificados: $p'_{ik} = p_{ik} + \min_{j \in V_k, i \neq j} s_{ijk}$.

PASSO 2. Sequencie as tarefas pela ordem não crescente dos tempos p'_{ik}.

PASSO 3. Para cada tarefa i da sequência:

(3*a*) Insira a tarefa i em toda posição possível em cada máquina

(3*b*) Utilizando os tempos de *setup* e os tempos de processamento originais, calcule o *makespan* parcial para cada inserção da tarefa i

(3*c*) Aloque a tarefa i na posição e na máquina com o menor *makespan* parcial.

O *makespan* parcial consiste na duração total da programação de um subconjunto de tarefas já alocadas até um estágio genérico k, com $1 \leq k \leq g$. Ou seja, é a diferença entre a maior data de término das tarefas já programadas e a menor data de liberação no primeiro estágio (que é adotada igual a zero).

Os autores aplicaram este algoritmo ao problema com *setup* dependente da sequência. Assim, para adaptá-lo ao presente trabalho foram utilizados como "tempos de processamento modificados" a soma dos tempos de processamento e de *setup* do último estágio (s_{jg}+p_{jg}), ou seja, do estágio em que o algoritmo *MI* é aplicado.

Algoritmo Hi2

PASSO 1. No estágio k=1, sequencie as tarefas pela ordem não decrescente da soma (s_{j1}+p_{j1}) e aloque uma tarefa de cada vez à máquina de menor carga. Faça k=2.

PASSO 2. Atualize as datas de liberação do estágio k como as datas de término do estágio k–1.

PASSO 3. Analise todas as possibilidades de alocação tarefa-máquina e escolha a opção com a menor data de término.

PASSO 4. Repita o PASSO 3 até que todas as tarefas que visitam o estágio k estejam programadas.

PASSO 5. Se k<g, faça k=k+1 e vá para o PASSO 2; senão (k=g), vá para o PASSO 6.

PASSO 6. Sequencie as tarefas pela ordem não crescente da soma (s_{jg}+p_{jg}) e para cada tarefa j da sequência:

(6*a*) Insira a tarefa j em toda posição possível em cada máquina do estágio g

(6*b*) Considerando as datas de liberação e os tempos de *setup* e de processamento, calcule o *makespan* parcial para cada inserção da tarefa j

(6*c*) Aloque a tarefa j na posição e na máquina com o menor *makespan* parcial.

◆ **HEURÍSTICA Hi3**

A heurística Hi3 aplica o método de inserção *MI* em todos os estágios, evidentemente com as devidas adaptações. A primeira adequação é a mesma efetuada na heurística Hi2 em relação aos tempos de processamento modificados, considerando a soma dos tempos de processamento e de *setup* (s_{jk}+p_{jk}) de cada estágio *k* programado.

Uma outra importante adaptação é o fato de se considerar a soma das datas de término do estágio como medida de desempenho dos estágios anteriores ao último ($k<g$). Como já salientado, esta função objetivo envida esforços no sentido de liberar o quanto antes as tarefas no estágio seguinte.

O último estágio tem implícita a vantagem de se manter a função objetivo original do problema, ou seja, a duração total da programação (*makespan*).

Algoritmo Hi3

PASSO 1. Faça k=1.

PASSO 2. No estágio *k*, sequencie as tarefas pela ordem não crescente da soma (s_{jk}+p_{jk}) e para cada tarefa *j* da sequência:
- (1*a*) Insira a tarefa *j* em toda posição possível em cada máquina do estágio *k*
- (1*b*) Considerando as datas de liberação e os tempos de *setup* e de processamento, calcule o soma das datas de término para cada inserção da tarefa *j*
- (1*c*) Aloque a tarefa *j* na posição e na máquina com a menor soma das datas de término.

PASSO 3. Faça k=k+1. Atualize as datas de liberação do estágio *k* como as datas de término do estágio k–1.

PASSO 4. Se $k<g$, vá para o PASSO 2; senão (k=g), vá para o PASSO 5.

PASSO 5. Sequencie as tarefas pela ordem não crescente da soma (s_{jg}+p_{jg}) e para cada tarefa *j* da sequência:
- (4*a*) Insira a tarefa *j* em toda posição possível em cada máquina do estágio *g*
- (4*b*) Considerando as datas de liberação e os tempos de *setup* e de processamento, calcule o *makespan* parcial para cada inserção da tarefa *j*
- (4*c*) Aloque a tarefa *j* na posição e na máquina com o menor *makespan* parcial.

◆ **HEURÍSTICA Hi4**

A heurística Hi4 possui um diferencial em relação às demais: a sua definição como método de decomposição por tarefa, conforme já citado. Cada tarefa selecionada é alocada em todos os estágios antes da seleção da próxima tarefa, diferentemente da programação de todas as tarefas em um estágio de cada vez. Ou seja, a programação é feita *por tarefa* e não *por estágio*.

Os valores utilizados no sequenciamento das tarefas são as somas de todos os estágios dos tempos de processamento e de *setup* quando não antecipados: $\sum_{k=1}^{g}\left[\left(1-A_{jk}\right)s_{jk}+p_{jk}\right]$, para cada tarefa *j*, onde A_{jk} = 1 se o *setup* s_{jk} é antecipado e A_{jk} = 0 caso o *setup* s_{jk} não for antecipado.

Algoritmo Hi4

PASSO 1. Sequencie as tarefas pela ordem não crescente do valor $\sum_{k=1}^{g}\left[\left(1-A_{jk}\right)s_{jk}+p_{jk}\right]$ e aloque a primeira tarefa da sequência em cada um dos estágios iniciando na data mais cedo possível. Faça k=1.

PASSO 2. Insira a próxima tarefa da sequência em toda posição possível em cada máquina do estágio *k*.

PASSO 3. Considerando as datas de liberação e os tempos de *setup* e de processamento, para cada inserção da tarefa, calcule o *makespan* parcial. Caso haja mais de uma inserção com o mesmo valor do *makespan* parcial, o desempate será feito pela menor soma das datas de término do estágio. Guarde a posição e a máquina com o menor valor encontrado.

PASSO 4. Aloque a tarefa na posição e na máquina do estágio k com o menor valor encontrado.
PASSO 5. Se $k<g$, faça $k=k+1$ e insira a mesma tarefa em toda posição possível em cada máquina do novo estágio k e vá para o PASSO 3; senão ($k=g$), vá para o PASSO 6.
PASSO 6. Se todas as tarefas já estiverem programadas, PARE; senão, faça $k=1$ e vá para o PASSO 2.

◆ **HEURÍSTICA Hi5**

Analogamente à heurística Hi1, que considera o primeiro estágio do ambiente tratado como um problema de máquinas paralelas com minimização do tempo médio de fluxo ($P \,||\, \overline{F}$ com $r_j=0,\ \forall j$), existe também um conhecido algoritmo para minimização do tempo médio de fluxo dado que o *makespan* seja ótimo ($P \,||\, \overline{F}$ *com* $C^*_{\max}$ e $r_j=0,\ \forall j$). O algoritmo heurístico para este problema consta em Morton e Pentico (1993, p.252) e será apresentado a seguir.

Algoritmo de solução heurística para o problema $P \,||\, \overline{F}$ *com* $C^*_{\max}$ e $r_j=0,\ \forall j$

PASSO 1. Ordene as tarefas pela regra *LPT* (*Longest Processing Time* – maior tempo de processamento).
PASSO 2. Aloque uma tarefa de cada vez à máquina de menor carga.
PASSO 3. Reordene as tarefas de cada máquina segundo a regra *SPT*.

A heurística Hi5 utiliza esta mesma ideia considerando na ordenação a soma $s_{jk}+p_{jk}$ para cada tarefa j em cada estágio k programado. Nos estágios $k \geq 2$, as datas de liberação são desconsideradas na alocação das tarefas visando equilibrar a carga de trabalho das máquinas do estágio, criando assim uma programação provisória. As datas de liberação são consideradas apenas na etapa de reordenação das tarefas de cada máquina, quando ocorre a programação definitiva.

Algoritmo Hi5

PASSO 1. No estágio $k=1$, sequencie as tarefas pela ordem não crescente da soma ($s_{j1}+p_{j1}$) e aloque uma tarefa de cada vez à máquina de menor carga.
PASSO 2. Reordene as tarefas de cada máquina do estágio 1 pela ordem não decrescente da soma ($s_{j1}+p_{j1}$). Faça $k=2$.
PASSO 3. No estágio k, sequencie as tarefas pela ordem não crescente da soma ($s_{jk}+p_{jk}$) e distribua uma tarefa de cada vez à máquina de menor carga do estágio k desconsiderando as datas de liberação das tarefas (como se fosse $r_{jk} = 0,\ \forall j$).
PASSO 4. Reordene as tarefas de cada máquina do estágio k pela ordem não decrescente do valor de $[r_{jk}+(1-A_{jk})s_{jk}+p_{jk}]$ e realoque as tarefas respeitando as suas datas de liberação e a carga das máquinas.
PASSO 5. Se for o último estágio ($k=g$), PARE; senão, faça $k=k+1$ e vá para o PASSO 3.

◆ **HEURÍSTICA Hi6**

Neste estudo foi utilizado também um método de programação aleatório que serve como base para comparar as heurísticas propostas, assim como foi feito por Moursli (1999).

As heurísticas aleatórias são rápidas e fáceis de implementar e são denominadas "heurísticas de lista" (*list heuristics*). Elas procedem em duas etapas: construção de uma lista ordenada de tarefas e associação das tarefas às máquinas (MOURSLI, 1999).

Algoritmo Hi6

PASSO 1.Faça $k=1$.
PASSO 2. No estágio k, associe números aleatórios às tarefas que o visitam e sequencie-as pela ordem não decrescente destes valores.

PASSO 3. Aloque sequencialmente cada tarefa à máquina selecionada por meio da geração de um número aleatório compreendido no intervalo dos índices das máquinas do estágio *k*.
PASSO 4. Repita o PASSO 3 até que todas as tarefas estejam programadas no estágio *k*.
PASSO 5. Se for o último estágio (*k*=*g*), PARE; senão, faça *k*=*k*+1 e vá para o PASSO 2.

4. Metodologia da experimentação computacional

Os parâmetros da experimentação computacional foram definidos com base em trabalhos reportados na literatura, como Kurz e Askin (2004), Logendran, Carson e Hanson (2005) e Ruiz, Şerifoğlu e Urlings (2008): 10, 30 e 100 tarefas; 3, 5 e 7 estágios de produção, dois intervalos de *setup* *U*[25, 74] e *U*[75, 125], dois intervalos de probabilidade do *setup* ser antecipado *U*[0,50]% e *U*[50, 100]% e duas opções de probabilidade de uma tarefa saltar um estágio, 0% e 50%.

O três níveis de flexibilidade determinados para o número de máquinas por estágio são: baixo, em que 1/3 dos estágios possuem máquinas paralelas; médio, com 2/3 dos estágios; e alto, em que todos os estágios possuem máquinas paralelas (LOGENDRAN; CARSON; HANSON, 2005).

Para o cálculo do número de estágios com máquinas paralelas, foi utilizado o arredondamento. Por exemplo, para o nível médio de flexibilidade em um sistema com 5 estágios, o número de estágios com máquinas paralelas é (2/3)*5 = 3,33. Assim, arredondando, tem-se 3 estágios com máquinas paralelas e 2 estágios com uma máquina. Para definir quais estágios terão as máquinas paralelas, foram utilizados valores inteiros uniformemente distribuídos. Em tais estágios, o número de máquinas paralelas é de 2 a 5.

Foi considerado um intervalo fixo para os tempos de processamento, com valores inteiros uniformemente distribuídos entre 1 e 99. Para cada classe de problemas, foram gerados aleatoriamente 100 problemas-teste visando reduzir o erro amostral. Isto perfaz um total de 216 classes de problemas e 21.600 problemas analisados.

De acordo com esses parâmetros, todos os problemas foram gerados aleatoriamente e resolvidos por meio de um software construído especificamente para esta finalidade. Foi utilizado o sistema operacional Windows e a linguagem de programação Delphi. As configurações da máquina são as seguintes: processador AMD Turion com 1.8 GHz de frequência e 512 MB de memória RAM.

Além das seis heurísticas propostas, para efeito de comparação, foram incluídas na experimentação computacional as duas melhores regras de prioridade de Fuchigami, Moccellin e Ruiz (2007), denominadas LPT3_ERD e SPT1_ERD, totalizando oito métodos de solução analisados.

A regra LPT3_ERD considera para o sequenciamento no primeiro estágio a ordem não crescente da soma dos tempos de *setup* e de processamento em todos os estágios e aplica a ordenação *ERD* (*Earliest Release Date* – menor data de liberação) nos estágios seguintes. Já a SPT1_ERD utiliza no primeiro estágio a ordem não decrescente da soma dos tempos de *setup* e de processamento do primeiro estágio e a regra *ERD* nos estágios seguintes. Em ambas as regras, cada tarefa é alocada sequencialmente à máquina de menor carga do estágio, levando à menor data de término.

As regras de prioridade de Fuchigami, Moccellin e Ruiz (2007) são também métodos heurísticos construtivos, uma vez que, além do critério de ordenação, elas incluem em sua estrutura a política de alocação adotada nos estágios.

4. Resultados obtidos

Os resultados obtidos na experimentação computacional foram analisados por meio da porcentagem de sucesso, desvio relativo médio, desvio-padrão do desvio relativo e tempo médio de computação dos oito métodos considerados.

A porcentagem de sucesso é calculada pelo número de vezes que o método forneceu a melhor solução (empatando ou não), dividido pelo número de problemas da classe.

O desvio relativo mede a variação correspondente à melhor solução obtida pelos

métodos, ou seja, a qualidade da solução. O desvio relativo (R_h) de um método *h* para um determinado problema é assim calculado: $R_h = (C^h_{\max} - C^b_{\max}) / C^b_{\max}$, onde $C^h_{\max}$ é o *makespan* fornecido pelo método *h* e $C^b_{\max}$ é o melhor *makespan* fornecido pelo grupo de métodos analisado.

O desvio-padrão do desvio relativo é o valor da variação dos desvios relativos de uma classe de problemas em torno do desvio relativo médio, ou seja, mede a estabilidade da solução. O desvio-padrão do desvio relativo (S_h) de um método *h* é calculado da seguinte forma: $S_h = \sqrt{\left[\sum_{i=1}^{L} (R_h^i - \overline{R})^2\right] / (L-1)}$, onde *L* é o número de problemas da classe, R_h^i é o desvio relativo da solução do problema *i* fornecida pelo método *h* e $\overline{R}$ é o desvio relativo médio do método *h* para a classe de problemas.

E o tempo médio de computação de um método é a média do tempo de CPU medido em milissegundos (ms).

O gráfico da Figura 2 mostra a comparação de desempenho dos métodos de solução na análise global dos problemas resolvidos.

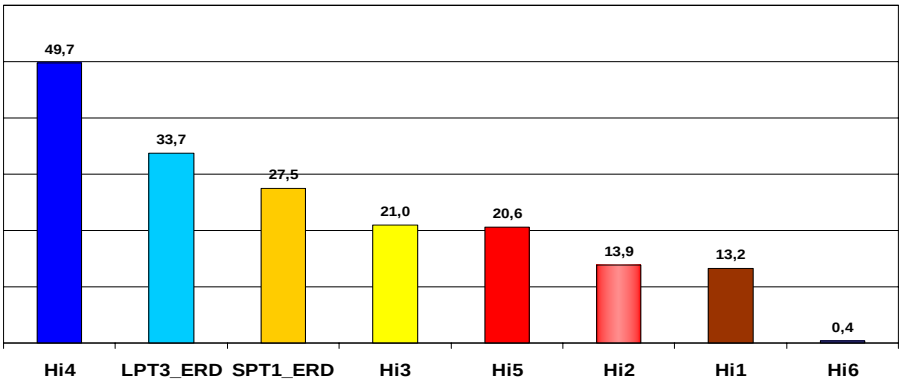

FIGURA 2 – Comparação de desempenho dos métodos heurísticos (em % de sucesso)

Dentre os 21.600 problemas, a heurística Hi4 ficou em primeiro lugar com 49,7% de sucesso. Em seguida, os melhores desempenhos foram obtidos pelas duas regras de prioridade: LPT3_ERD, com 33,7% de sucesso, e SPT1_ERD, com 27,5%. Isto mostra a vantagem de se utilizar os procedimentos mais simples, como as regras de prioridade, na programação do *flexible flow line*, quando comparados à maioria das heurísticas.

Entre as heurísticas propostas, a Hi4 é a única que faz a programação por tarefa e não por estágio. Além disso, ela utiliza na ordenação inicial a soma dos tempos de processamento e de *setup* não antecipados de todos os estágios, priorizando as tarefas com maior carga de trabalho e alocando em seguida as tarefas menores, mais fáceis de "encaixar" na programação.

Embora tanto a heurística Hi4 como a regra LPT3_ERD utilizem a ordem não crescente com tempos de todos os estágios, elas diferem levemente, pois a LPT3_ERD soma inclusive os tempos de *setup* antecipados (o que não ocorre na Hi4). Além disso, as políticas de alocação de ambas são completamente distintas.

Em quarto e quinto lugares ficaram as heurísticas Hi3 e Hi5, com resultados bastante próximos: respectivamente, 21% e 20,6% de sucesso. As heurísticas Hi1 e Hi2 também obtiveram resultados muito próximos, com 13,2% e 13,9% de sucesso. Ambas utilizam a regra *SPT* na ordenação inicial, tal como a regra SPT1_ERD. Porém, esta última, que atingiu 27,5% de sucesso em relação às heurísticas, indica que a alocação *ERD* é mais vantajosa do que a escolha do par tarefa-máquina com menor data de término (*ECT*), utilizada na Hi1 e Hi2.

A heurística Hi2 foi criada como tentativa de melhoria da Hi1, tendo como única diferença a programação do último estágio, em que aplica o método de inserção. Contudo, o resultado melhorou em apenas 0,7% em média.

A heurística aleatória Hi6 ficou em último lugar, com apenas 0,4% de sucesso.

Para a visualização dos resultados com um nível de detalhamento maior, a Figura 3 apresenta a comparação de desempenho das heurísticas para cada opção de porte do problema. Mesmo nesta análise detalhada, prevalece o desempenho superior da heurística Hi4. A regra LPT3_ERD também ficou em segundo lugar em todas as opções de número de estágios e de tarefas.

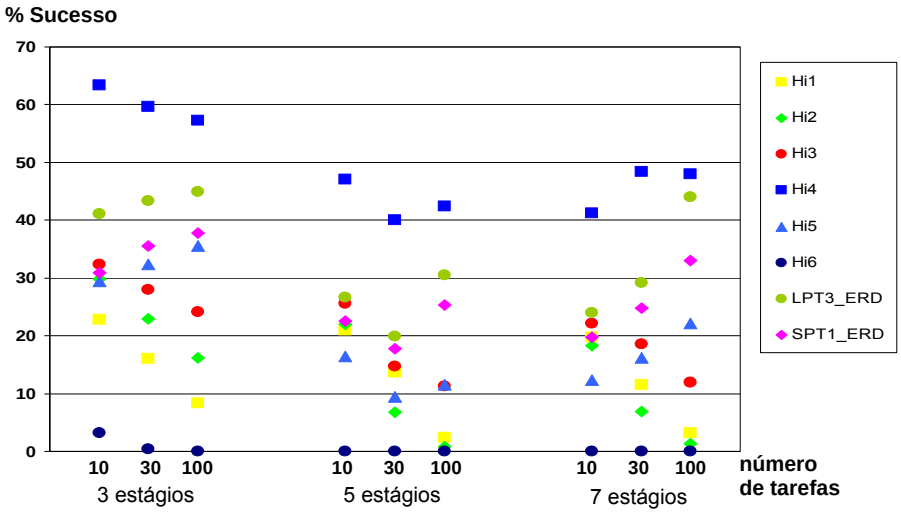

FIGURA 3 – Desempenho dos métodos heurísticos por porte do problema (em % de sucesso)

Para problemas com 3 estágios, as heurísticas Hi1, Hi2, Hi3, Hi4 e Hi6 reduzem o desempenho com o aumento do número de tarefas. Por outro lado, a heurística Hi5 e as duas regras de prioridade melhoram o desempenho com o aumento do número de tarefas.

Com 5 estágios, as heurísticas sempre apresentaram desempenho inferior em problemas com 30 tarefas quando comparado aos problemas com 10 e 100 tarefas. Em problemas com 7 estágios, a heurística Hi4 passa a melhorar o desempenho com o aumento do número de tarefas. Ou seja, ela apresenta desempenho superior em problemas de médio e grande porte.

Em todos os casos, o resultado da heurística Hi6 ficou sempre próximo de zero, exceto em problemas com 10 tarefas e 3 estágios, em que obteve 3% de sucesso.

Os dados da Tabela 1 confirmam a análise do gráfico da Figura 3, explicitando a ordem de superioridade das quatro melhores heurísticas e suas porcentagens de sucesso. Pode-se perceber que o terceiro lugar é disputado pelas heurísticas Hi3 e SPT1_ERD e o quarto lugar é relativamente alternado.

TABELA 1 – Melhores métodos heurísticos por porte do problema e suas porcentagens de sucesso

g	*n*	1º lugar	2º lugar	3º lugar	4º lugar
3	10	**Hi4** 63,3%	**LPT3_ERD** 41,0%	**Hi3** 32,3%	**SPT1_ERD** 30,9%
	30	**Hi4** 59,7,%	**LPT3_ERD** 43,3%	**SPT1_ERD** 35,5%	**Hi5** 32,4%
	100	**Hi4** 57,3,%	**LPT3_ERD** 44,9%	**SPT1_ERD** 37,8%	**Hi5** 35,5%
5	10	**Hi4** 47,0%	**LPT3_ERD** 26,7%	**Hi3** 25,6%	**SPT1_ERD** 22,5%
	30	**Hi4** 40,0%	**LPT3_ERD** 19,9%	**SPT1_ERD** 17,8%	**Hi3** 14,7%
	100	**Hi4** 42,5%	**LPT3_ERD** 30,5%	**SPT1_ERD** 25,3%	**Hi5** 11,5%
7	10	**Hi4** 41,2%	**LPT3_ERD** 24,0%	**Hi3** 22,2%	**Hi1** 19,8%
	30	**Hi4** 48,4%	**LPT3_ERD** 29,2%	**SPT1_ERD** 24,8%	**Hi3** 18,6%
	100	**Hi4** 48,0%	**LPT3_ERD** 44,0%	**SPT1_ERD** 33,0%	**Hi5** 22,2%

Na análise separada por opção de flexibilidade, a heurística Hi4 também fornece os melhores resultados. Já para cada opção de probabilidade de salto, o desempenho de cada método

foi substancialmente diferente, principalmente para as duas melhores heurísticas Hi4 e LPT3_ERD. Para probabilidade de 0% de salto, ou seja, quando as tarefas não saltam nenhum estágio, a Hi4 obteve de 10% a 39% de sucesso, enquanto que com 50% de probabilidade de salto, o seu desempenho aumentou expressivamente para 64% a 88%. Com 0% probabilidade de salto, a LPT3_ERD teve desempenho de 10% a 18% de sucesso e com 50% de probabilidade subiu para 26% a 80%.

Os intervalos de *setup* e as probabilidades do *setup* ser antecipado não influenciaram de forma significativa o desempenho dos métodos.

Os valores dos desvios relativos confirmaram as conclusões da análise feita para a porcentagem de sucesso. O melhor método obteve o menor desvio relativo: a Hi4 com 0,02. E o pior método teve o maior desvio relativo, o que já era esperado por se tratar de um procedimento aleatório: a Hi6 com 1,43. O desvio relativo das duas regras foi bem pequeno: LPT3_ERD com 0,03 e SPT1_ERD com 0,05. O resultado das outras heurísticas foi o seguinte: Hi1 com 0,06; Hi3 e Hi5 com 0,07; e Hi2 com 0,08.

Com exceção da heurística aleatória Hi6, as médias dos valores do desvio-padrão do desvio relativo ficaram entre 0,03 e 0,07, uma amplitude pouco significativa, indicando a estabilidade do desempenho dos métodos. O desvio-padrão da pior heurística foi de 0,30.

As duas regras de prioridade tiveram o menor tempo de CPU, de 0,1 ms, conforme já era previsto, por demandar menor complexidade no código computacional. As heurísticas Hi1, Hi5 e Hi6 levaram em média 0,5 ms para o processamento, a Hi2 3,5 ms e a Hi3 20,1 ms.

A heurística Hi4 teve o tempo médio de CPU de 179,1 ms ou aproximadamente 0,2 segundos, o que indica que deve-se ter cautela na sua utilização dependendo do porte do problema. Esse tempo deve-se ao código demandado pelo método de inserção em todos os estágios. Ainda assim, por ter obtido os expressivos resultados de quase 50% de sucesso, sua eficácia justifica a sua pequena perda de eficiência computacional, o que ocorre basicamente nos problemas com 100 tarefas.

5. Considerações finais

Neste trabalho foram propostos seis métodos de solução heurística para programação no ambiente *flexible flow line* com tempos de *setup* independentes da sequência de tarefas, com a minimização do *makespan*, incluindo um procedimento aleatório (Hi6). O resultado destas heurísticas foi comparado com os dois melhores métodos de Fuchigami, Moccellin e Ruiz (2007).

A heurística que forneceu os melhores resultados foi a Hi4, com 49,7% de sucesso, sendo a única que constitui um método de composição por tarefa (e não por estágio). As heurísticas baseadas em regras de prioridade também forneceram bons resultados: a LPT3_ERD com 33,7% de sucesso e a SPT1_ERD com 27,5%. Já a heurística aleatória obteve o pior resultado,com apenas 0,4% de sucesso. O desempenho da heurística SPT1_ERD indica que é mais vantajoso empregar a alocação pela regra *ERD* do que a *ECT*, como é o caso da Hi1 e Hi2.

Tanto o layout, no que se refere ao número de estágios e à flexibilidade (número de estágios com máquinas paralelas), como as características das tarefas, basicamente em relação à probabilidade de saltar estágios, influenciam de forma significativa no resultado dos métodos de solução propostos. Já os fatores relacionados ao setup (intervalos e probabilidade de antecipação) não interferiram de forma relevante no desempenho das heurísticas.

Como sugestão para futuros trabalhos, podem ser desenvolvidas heurísticas de melhoria ou então diferentes procedimentos de solução, como as meta-heurísticas. Além disso, para este problema de programação, podem ser consideradas outras medidas de desempenho como tempo médio de fluxo, pontualidade média (*lateness*), atraso máximo, ou mesmo dois critérios em conjunto (problema bicritério).

Referências

Allahverdi, A., Gupta, J.N.D. e Aldowaisan, T. (1999), A review of scheduling research involving setup considerations, *Omega - The International Journal of Management Science*, 27, 219-239.

Allaoui, H. e Artiba, A. (2004), Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints, *Computers and Industrial Engineering*, 47, 431-450.
Baker, K.R., *Introduction to sequencing and scheduling*, John Wiley & Sons, New York, 1974.
Botta-Genoulaz, V. (2000), Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness, *International Journal of Production Economics*, 64, 101-111.
Fuchigami, H.Y., *Flexible flow line com tempos de setup: métodos heurísticos*, Tese (Doutorado), Universidade de São Paulo, São Carlos, 2010.
Fuchigami, H.Y. e Moccellin, J.V. Métodos heurísticos construtivos para programação da produção em sistemas *flow shop* híbridos com tempos de preparação das máquinas assimétricos e dependentes da sequência, *Anais do 37º Simpósio Brasileiro de Pesquisa Operacional*, 2005.
Fuchigami, H.Y., Moccellin, J.V. e Ruiz, R., Análise comparativa do desempenho de regras de prioridade em sistemas *flexible flow line* com múltiplas máquinas e tempos de *setup* independentes da sequência, *Anais do 39º Simpósio Brasileiro de Pesquisa Operacional*, 2007.
Gupta, J.N.D. e Tunc, E.A. (1994), Scheduling a two-stage hybrid flowshop with separable setup and removal times, *European Journal of Operational Research*, 77, 415-428.
Huang, W. e Li, S. (1998), A two-stage hybrid flowshop with uniform machines and setup times, *Mathematical and Computer Modeling*, 27, 27-45.
Kis, T. e Pesch, E. (2005), A review of exact solution methods for the non-preemptive multiprocessor flowshop problem, *European Journal of Operational Research*, 164, 592-608.
Kurz, M.E. e Askin, R.G. (2001), Heuristic scheduling of parallel machines with sequence-dependent set-up times, *International Journal of Production Research*, 39, 3747-3769.
Kurz, M.E. e Askin, R.G. (2003), Comparing scheduling rules for flexible flow lines, *International Journal of Production Economics*, 85, 371-388.
Kurz, M.E. e Askin, R.G. (2004), Scheduling flexible flow lines with sequence-dependent setup times, *International Journal of Operational Research*, 159, 66-82.
Li, S. (1997), A hybrid two-stage flowshop with part family, batch production, and major and minor set-ups, *European Journal of Operational Research*, 102, 142-156.
Lin, B.M.T. e Cheng, T.C.E. (2005), Two-machine flowshop batching and scheduling, *Annals of Operations Research*, 133, 149-161.
Lin, H.T. e Liao, C.J. (2003), A case study in a two-stage hybrid flow shop with setup time and dedicated machines, *International Journal of Production Economics*, 86, 2, 133-143.
Linn, R. e Zhang, W. (1999), Hybrid flow shop scheduling: a survey, *Computers & Industrial Engineering*, 37, 1-2, 57-61.
Logendran, R., Carson, S. e Hanson, E. (2005), Group scheduling in flexible flow shops, *International Journal of Production Economics*, 96, 143-155.
Low, C. (2005), Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines, *Computers and Operations Research*, 32, 2013-2025.
Moursli, O., *Scheduling the hybrid flowshop*: branch and bound algorithms. 198p. Tese (Doutorado), Faculte des Sciences Economiques, Sociales et Politiques, Universite Catholique de Louvain, Louvain, Bélgica, 1999.
Morton, T.E. e Pentico, D.W., *Heuristic scheduling systems*, John Wiley & Sons, New York, 1993.
Naderi, B., Ruiz, R. e Zandieh, M. (2010), Algorithms for a realistic variant of flowshop scheduling, *Computers and Operations Research*, 37, 2, 236-246.
Pinedo, M.L, *Scheduling: Theory, Algorithms, and Systems*, Springer, New York, 2003, 3rd. ed.
Quadt, D. e Kuhn, H. (2007), A taxonomy of flexible flow line scheduling procedures, *European Journal of Operational Research*, 178, 686-698.
Ruiz, R. e Maroto, C. (2006), A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, *European Journal of Operational Research*, 169, 781-800.
Ruiz, R., Sivrikaya-Şerifğlu, F. e Urlings, T. (2008), Modeling realistic hybrid flexible flowshop scheduling problems, *Computers and Operations Research*, 35, 1151-1175.
Vignier, A., Billaut, J.C. e Proust, C. (1999), Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art, *RAIRO – Recherche Opérationnelle*, 33, 2, 117-183.
Wang, H. (2005), Flexible flow shop scheduling: optimum, heuristic and artificial intelligence solutions, *Expert Systems*, 22, 2, 78-85.
Wilson, A.D., King, R.E. e Hodgson, T.J. (2004), Scheduling non-similar groups on a flow line: multiple group setups, *Robotics and Computer-Integrated Manufacturing*, 20, 505-515.

A pesquisa relatada neste artigo teve o apoio da CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior.