# MÉTODOS HEURÍSTICOS PARA MAXIMIZAÇÃO DO NÚMERO DE TAREFAS *JUST-IN-TIME* EM *FLOW SHOP* PERMUTACIONAL

**Hélio Yochihiro Fuchigami**
Universidade Federal de Goiás (UFG) – Regional Goiânia
Campus Aparecida de Goiânia – Engenharia de Produção
R. Mucuri, s/n, Área 3, Setor Conde dos Arcos, CEP 74.968-755, Aparecida de Goiânia/GO

Pós-doutorando do PNPD/CAPES
Universidade Estadual Paulista (UNESP) – IBILCE
Departamento de Matemática Aplicada
R. Cristóvão Colombo, 2265, CEP 15.054-000, S.J.Rio Preto/SP
heliofuchigami@ufg.br

**Socorro Rangel**
Universidade Estadual Paulista (UNESP) – IBILCE
Departamento de Matemática Aplicada
R. Cristóvão Colombo, 2265, CEP 15.054-000, S.J.Rio Preto/SP
socorro@ibilce.unesp.br

## RESUMO

Este trabalho trata do problema de sequenciamento em *flow shop* permutacional com o objetivo de maximizar o número de tarefas *just-in-time*, ou seja, tarefas que concluem seu processamento pontualmente. Foram propostas e implementadas computacionalmente dez heurísticas construtivas para resolver o problema. As heurísticas foram avaliadas quanto à eficácia da solução (qualidade) por meio do desvio relativo percentual e quanto à eficiência computacional (tempo de CPU). Os resultados comprovaram a aplicabilidade dos métodos de solução. A melhor heurística obteve soluções com desvio médio de apenas 0,2% da melhor solução encontrada para problemas com até 10 tarefas e 5 máquinas e tempo de execução aceitável mesmo para problemas de grande porte.

**PALAVRAS CHAVE. Programação da produção, *Flow shop*, Heurísticas, Tarefas *just-in-time*.**

**AD&GP**

## ABSTRACT

This paper aims at maximizing the number of just-in-time jobs in a permutation flow shop scheduling problem. Ten constructive heuristics were proposed and computationally implemented to solve the problem. The heuristic methods were evaluated for efficacy of the solution (quality) by relative percentage deviation and for computational efficiency (CPU time). The results show the applicability of the solution methods. The best heuristic found solutions with an average deviation of only 0.2% of the best found solution for problems with up to 10 jobs and 5 machines within an acceptable running time even for large problems.

**KEYWORDS. Scheduling. Flow shop. Heuristics. Just-in-time jobs.**

**AD&GP**

## 1. Introdução

A programação da produção (*scheduling*) em *flow shop* permutacional, um sistema produtivo em que as tarefas seguem o mesmo fluxo por todas as máquinas na mesma ordem, é um dos mais importantes problemas de planejamento a produção. Este tipo de manufatura é encontrado com muita frequência nos ambientes industriais de produção intermitente.

Na literatura científica, as medidas de desempenho mais comuns são aquelas relacionadas ao fluxo de tarefas, como a duração total da programação e o tempo médio de fluxo, ou então as derivadas do atraso, como o atraso total, atraso máximo e número de tarefas atrasadas. Entretanto, esta ênfase começou a mudar com o crescente interesse pela produção *just-in-time*.

Surgida no Japão após a Segunda Guerra Mundial, a filosofia *just-in-time*, em uma definição simplista, consiste num planejamento para se adquirir os materiais e iniciar a produção apenas no momento em que se garanta a entrega "no momento exato". O seu advento suscitou posteriormente a aplicação na área de programação da produção. As tarefas que são concluídas exatamente nos seus prazos, ou seja, pontualmente, são denominadas "tarefas *just-in-time*".

A filosofia envolve amplos conceitos relacionados à redução de desperdícios, como redução de estoques, dos lotes de produção, dos tempos de espera e dos atrasos internos e também o estabelecimento de prazos de entregas maiores, melhoria na qualidade dos produtos e dos processos e dos programas de produção. Maiores informações sobre a filosofia *just-in-time* podem ser encontradas em Fernandes e Godinho Filho (2010) e em Krajewski, Ritzman e Malhotra (2009); e especificamente sobre *just-in-time scheduling* em Józefowska (2007) e Ríos-Mercado e Ríos-Solís (2012).

A programação da produção *just-in-time* consiste em obter uma solução que minimize funções de custo associado aos adiantamentos e atrasos das tarefas. Por este motivo, é mais comum encontrar pesquisas abordando a soma dos adiantamentos e atrasos das tarefas ou das penalidades ocasionadas por este desvio na finalização do processamento ou da entrega do produto em relação aos prazos estabelecidos previamente.

Na prática, estas medidas de desempenho são muito importantes para as empresas, pois tanto adiantamentos no término das tarefas como atrasos acarretam maiores custos à produção, respectivamente, com aumentos nos níveis de estoques e com incidência de multas, cancelamentos de pedidos ou até mesmo perda de clientes. Shabtay e Steiner (2012) exemplificam algumas aplicações: indústrias químicas e de alta tecnologia, programação de aeronaves espaciais, produção de itens perecíveis sob demanda determinística e ambientes produtivos sem capacidade de armazenamento.

Na clássica notação de três campos, o problema de maximização do número de tarefas *just-in-time* em *flow shop* permutacional pode ser denotado por $Fm|prmu,d_j|n_{JIT}$, onde "*Fm*" indica o ambiente *flow shop* genérico com "*m*" máquinas, "*prmu*" é a restrição permutacional, "$d_j$" enfatiza a existência de prazos para as tarefas e "$n_{JIT}$" representa a função-objetivo que maximiza o número de tarefas concluídas *just-in-time*.

Neste problema, além do sequenciamento, a solução será composta também pelo cronograma, a definição do instante de início de cada operação, pois a programação permite a inserção de tempo ocioso.

Este trabalho objetiva propor heurísticas construtivas, caracterizadas pela simplicidade na concepção e implementação, pela fundamentação em critérios de decisão lógicos e por fornecerem boas soluções em relativamente curto tempo computacional. A pesquisa se justifica pela importância prática do problema tratado, por resultados bastante promissores em áreas correlacionadas e por ser um tema pouquíssimo explorado na literatura específica.

## 2. Maximização do número de tarefas *just-in-time* em *flow shop*

Conforme salientado, embora a filosofia *just-in-time* envolva conceitos mais abrangentes, os problemas de programação da produção nesta área eram considerados até as últimas décadas como variações da minimização dos adiantamentos e atrasos, como mostra a revisão de Baker e

Scudder (1990). Uma profusa fundamentação destes problemas pode ser consultada em Józefowska (2007) e em Ríos-Mercado e Ríos-Solís (2012).

Recentemente, Shabtay e Steiner (2012) publicaram uma revisão da literatura abordando o problema de programação da produção com maximização do número de tarefas *just-in-time*, demonstrando que o tema tem sido pouquíssimo explorado. A obra de Ríos-Mercado e Ríos-Solís (2012), que aborda vários tipos de problemas de programação *just-in-time*, contém apenas um trabalho considerando a medida do número de tarefas *just-in-time*, que é exatamente este *survey* desenvolvido por Shabtay e Steiner (2012) dos ambientes de máquina única, máquinas paralelas e *flow shop* com duas máquinas.

As funções-objetivos *just-in-time* mais comumente encontradas nas pesquisas, conforme Baker e Trietsch (2009), são a soma (ponderada ou com pesos iguais) dos adiantamentos e atrasos das tarefas e a soma do adiantamento e do atraso máximos. Embora existam alguns trabalhos na literatura específica abordando o critério de maximização do número de tarefas *just-in-time* nos diversos sistemas de produção, especificamente com *flow shop* foram encontradas poucas publicações, que serão descritas a seguir.

Choi e Yoon (2007) mostraram que mesmo o problema ponderado com duas máquinas é classificado como NP-completo e o problema com pesos idênticos com três máquinas é NP-hard. Os autores propuseram e demonstraram várias condições de dominância para o problema com pesos iguais e, com base nelas, apresentaram um algoritmo para o problema com duas máquinas. Além disso, os autores provaram que existe uma solução ótima para o problema com duas máquinas dada pela regra de ordenação EDD (*Earliest Due Date*).

Diversos problemas ponderados com duas máquinas, tanto *flow shop* como *job shop* e *open shop*, foram analisados por Shabtay e Bensoussan (2012), que propuseram algoritmos de tempos pseudopolinomiais e também uma forma de convertê-los em esquemas de aproximação em tempo polinomial. Elalouf Levner e Tang (2013) sugeriram uma melhoria na redução da complexidade do algoritmo de Shabtay e Bensoussan (2012) para *flow shop* com duas máquinas.

Shabtay (2012) analisou quatro diferentes cenários para o problema ponderado: o primeiro, com duas máquinas e pesos idênticos para as tarefas *just-in-time*; o segundo, com um *flow shop* proporcional, em que os tempos de processamento das operações é o mesmo em todas as máquinas; no terceiro, um conjunto de tarefas idênticas precisa ser produzido para diferentes clientes (os tempos de processamento das tarefas são iguais, porém os prazos são diferentes para cada cliente); e o último, com a restrição *no-wait*, ou seja, sem tempo de espera entre as operações de uma mesma tarefa.

O problema de *flow shop* proporcional também foi tratado por Gerstl, Mor e Mosheiov (2015), que consideraram as possibilidades com e sem a restrição *no-wait*, e propuseram algoritmos de programação dinâmica, que requerem tempo computacional na ordem de $O(n^2)$.

Um problema bicritério, com duas máquinas, considerando conjuntamente a maximização do número ponderado de tarefas *just-in-time* e a minimização do custo de consumo de recursos, foi abordado por Shabtay, Bensoussan e Kaspi (2012). É interessante observar que este foi único trabalho que explicitou uma implementação computacional dos métodos propostos. Diferentemente da maioria dos trabalhos na área de programação da produção, em que geralmente se emprega a metodologia de pesquisa experimental (com implementação computacional), todos os outros estudos examinados foram teóricos, com demonstrações de resultados que atestam a validade dos métodos propostos, incluindo a análise da complexidade dos algoritmos.

Além disso, dado o grau de dificuldade de se resolver diretamente esta classe de problemas de maximização de tarefas *just-in-time*, em alguns trabalhos optou-se pela conversão em outro tipo de problema de otimização (como por exemplo a partição do problema, em Choi e Yoon, 2007, e a modelagem de grafos orientados acíclicos, em Shabtay, 2012).

Em geral, as pesquisas se limitaram ao problema de *flow shop* com apenas duas máquinas, não sendo encontrado nenhum trabalho propondo heurísticas construtivas para o *flow shop* clássico (sem restrições adicionais) com qualquer número de máquinas, incluindo experimentação computacional que avalie a eficácia do algoritmo em termos de qualidade de

solução e também a eficiência computacional. Aqui se encontra, portanto, a principal contribuição do presente trabalho, ao se apresentar métodos de solução eficientes e eficazes, que se demonstram aplicáveis em problemas práticos de *flow shop* permutacional com o objetivo de maximizar o número de tarefas *just-in-time* em um ambiente com várias máquinas.

## 3. Métodos heurísticos propostos

O problema tratado pode ser formulado genericamente da seguinte forma. Considere um conjunto de $n$ tarefas $J = \{J_1, J_2, ..., J_n\}$, que são independentes, possuem o mesmo peso ou prioridade, não podem ser interrompidas e estão todas disponíveis para processamento no instante zero da programação. Todas as tarefas devem ser executadas em $m$ máquinas, $M = \{M_1, M_2, ..., M_m\}$, dispostas fisicamente de forma a respeitar um fluxo linear unidirecional. O fluxo de todas as tarefas nas máquinas é idêntico. Cada tarefa $J_j$ requer um tempo de processamento $p_{jk}$ em cada uma das máquinas $M_k$ e possui o seu prazo representado por $d_j$, ambos considerados conhecidos previamente e fixos. Deseja-se encontrar uma programação que maximize o número de tarefas que são concluídas exatamente no seu respectivo prazo.

Para resolver o problema descrito, foram concebidos métodos heurísticos construtivos com base na investigação da estrutura do problema e inspirados em algoritmos clássicos da área, como a heurística NEH de Nawaz, Enscore Jr. e Ham (1983) e a de Hodgson (1977), além de regras de prioridade bem conhecidas, como a EDD, que faz a ordenação crescente pelos prazos das tarefas, e a MST (*Minimum Slack Time*), que sequencia as tarefas pela menor folga ($\square_{\square} - \sum_{\square=1}^{\square} \square_{\square\square}$). Sabe-se que no problema de máquina única, a regra EDD minimiza o atraso máximo e a MST, o adiantamento máximo.

Todas as heurísticas propostas requerem uma etapa de ajuste de cronograma, que consiste em verificar o melhor instante para se iniciar as operações de forma que, para uma dada sequência, um maior número de tarefas sejam concluídas pontualmente. O simples deslocamento de uma tarefa antecipada com folga para que coincida com seu prazo poderá não provocar uma melhoria global na solução.

***Procedimento de ajuste de cronograma***

(1) Para a sequência dada, considerando o início das operações de cada tarefa no instante mais cedo possível, contabilize o número de tarefas *just-in-time* ($\square_{\square\square\square}$) e considere $J_{inicial} = J_{[1]}$, onde $J_{[j]}$ é a tarefa que ocupa a posição $j$ da sequência.

(2) A partir de $J_{inicial}$, identifique a primeira tarefa adiantada $J_E$ da sequência e vá para o passo (3). Se não houver tarefas adiantadas (a partir de $J_{inicial}$), FINALIZE.

(3) Desloque a última operação da tarefa $J_E$ de forma a eliminar o adiantamento, fazendo sua conclusão coincidir com o seu prazo. Reprograme as últimas operações das tarefas seguintes à $J_E$, respeitando o seu deslocamento.

(4) Contabilize o novo número de tarefas *just-in-time* ($\square'_{\square\square\square}$).

Se $\square'_{\square\square\square} < \square_{\square\square\square}$ (a nova solução piorou), volte a última operação de $J_E$ à sua posição anterior, bem como as demais operações das tarefas seguintes, e faça $J_{inicial} = J_{[inicial+1]}$.

Se $\square'_{\square\square\square} \geq \square_{\square\square\square}$ (a nova solução é melhor ou igual), mantenha a nova programação, faça $J_{inicial} = J_{[E+1]}$ e volte ao passo (2).

É importante observar que optou-se pelo deslocamento apenas na última operação da tarefa pois manter as operações anteriores iniciando na sua data mais cedo possível pode constituir numa vantagem, proporcionando a antecipação de tarefas que poderiam se atrasar por causa das suas primeiras operações. Além disso, note que o deslocamento só é mantido quando houver de fato uma melhora na solução; quando há empate com a nova solução, o deslocamento é revertido para que se antecipem as operações e assim contribua para eliminar algum possível atraso das tarefas posteriores.

As primeiras quatro heurísticas propostas são adaptações do método de inserção do algoritmo NEH. As heurísticas H1 e H2 utilizam a regra EDD na ordenação inicial, enquanto a H3 e a H4 consideram a regra MST. Outra característica deste conjunto de algoritmos é que as

heurísticas pares (H2 e H4) constituem-se do emprego de métodos de melhoria (de busca em vizinhança da sequência parcial em construção) nas heurísticas ímpares (H1 e H3, respectivamente).

***Heurística H1***

(1) Ordene as tarefas pela regra EDD (desempate pela menor $\sum p_{jk}$).
(2) Com as duas primeiras tarefas da ordenação, aplicando o *Procedimento de ajuste do cronograma*, encontre a subsequência (entre as duas possíveis) com o melhor $n_{JIT}$.
(3) Para $h = 3$ a $n$, faça:
Sem alterar as posições relativas das tarefas já programadas, insira a tarefa que ocupa a $h$-ésima posição da ordenação em todas as posições possíveis da subsequência e, executando o *Procedimento de ajuste de cronograma*, considere aquela com melhor $n_{JIT}$ (desempate pela maior posição).

***Heurística H2***

(1) Ordene as tarefas pela regra EDD (desempate pela menor $\sum p_{jk}$).
(2) Com as duas primeiras tarefas da ordenação, aplicando o *Procedimento de ajuste do cronograma*, encontre a subsequência (entre as duas possíveis) com o melhor $n_{JIT}$.
(3) Para $h = 3$ a $n$, faça:
Acrescente à subsequência a tarefa que ocupa a $h$-ésima posição da ordenação.
Considerando toda a Vizinhança de Inserção da sequência com $(h-1)^2$ soluções e executando o *Procedimento de ajuste de cronograma*, determine aquela com melhor $n_{JIT}$.
Considerando toda a Vizinhança de Permutação da sequência com $h(h-1)/2$ soluções e executando o *Procedimento de ajuste de cronograma*, determine aquela com melhor $n_{JIT}$.

***Heurística H3***

(1) Ordene as tarefas pela regra MST (desempate pela menor $\sum p_{jk}$).
(2) e (3) Idem à heurística H1.

***Heurística H4***

(1) Ordene as tarefas pela regra MST (desempate pela menor $\sum p_{jk}$).
(2) e (3) Idem à heurística H2.

As próximas quatro heurísticas, H5, H6, H7 e H8, utilizam ideias advindas do clássico algoritmo de Hodgson (1977), que fornece a solução ótima para o problema de minimização do número de tarefas atrasadas no problema de máquina única. Novamente, as duas primeiras consideram a regra EDD e duas últimas, a MST. E também as heurísticas pares diferenciam-se das ímpares por empregarem métodos de melhoria (busca em vizinhança ao final da execução).

***Heurística H5***

(1) Ordene as tarefas pela regra EDD (desempate pela menor $\sum p_{jk}$).
(2) Execute o *Procedimento de ajuste do cronograma*.
(3) Identifique a primeira tarefa atrasada $J_T$ da sequência e vá para o passo (4). Se não há tarefas atrasadas, FINALIZE.
(4) Desloque a tarefa $J_T$ para o final da sequência e volte ao passo (2).

***Heurística H6***

(1) Ordene as tarefas pela regra EDD (desempate pela menor $\sum p_{jk}$).
(2) Execute o *Procedimento de ajuste do cronograma*.
(3) Identifique a primeira tarefa atrasada $J_T$ da sequência e vá para o passo (4). Se não há tarefas atrasadas, FINALIZE.

(4) Desloque a tarefa $J_T$ para o final da sequência e volte ao passo (2).
(5) Considerando toda a Vizinhança de Inserção da sequência com $(n–1)^2$ soluções, determine aquela com melhor $n_{JIT}$.
Considerando toda a Vizinhança de Permutação da sequência com $n(n–1)/2$ soluções, determine aquela com melhor $n_{JIT}$.

***Heurística H7***

(1) Ordene as tarefas pela regra MST (desempate pela menor $\sum p_{jk}$).
(2) a (4) Idem à heurística H5.

***Heurística H8***

(1) Ordene as tarefas pela regra MST (desempate pela menor $\sum p_{jk}$).
(2) a (5) Idem à heurística H6.

Finalmente, nas duas últimas heurísticas, a H9 utiliza a EDD como regra de ordenação inicial enquanto a H10 considera a MST, e em ambas foi adotada uma forma diferente de busca em vizinhança. Ao invés do esquema de vizinhança de inserção e de permutação, as heurísticas H9 e H10 reinserem uma tarefa de cada vez na última posição, considerando a melhor sequência, e em seguida reinserem iterativamente a última tarefa em todas as demais posições, mantendo aquela com melhor $n_{JIT}$.

***Heurística H9***

(1) Ordene as tarefas pela regra EDD (desempate pela menor $\sum p_{jk}$).
(2) Execute o *Procedimento de ajuste do cronograma*.
(3) Faça $h$=1. Enquanto $h < n$, desloque a $h$-ésima tarefa para a última posição da sequência e execute o *Procedimento de ajuste de cronograma*. Se o valor de $n_{JIT}$ da nova solução for melhor que o da anterior, mantenha a nova sequência (e o valor de $h$), senão volte à anterior e faça $h=h+1$.
(4) Faça $h=n$. Enquanto $h \geq 1$, insira a $h$-ésima tarefa em todas as posições anteriores e, executando o *Procedimento de ajuste de cronograma*, considere a melhor solução encontrada. Se não houve deslocamento da última tarefa, ou seja, a melhor posição é a que estava originalmente, faça $h=h–1$.

***Heurística H10***

(1) Ordene as tarefas pela regra MST (desempate pela menor $\sum p_{jk}$).
(2) a (4) Idem à heurística H9.

Na seção 4 será apresentada a averiguação da qualidade da solução e da eficiência computacional das heurísticas construtivas propostas.

## 4. Experimentação computacional e resultados

Esta seção detalha o planejamento da experimentação computacional e a análise dos resultados alcançados.

### 4.1 Planejamento do experimento

Na experimentação computacional foram testados e avaliados 15.600 problemas, divididos em dois grupos: Grupo 1, com problemas de pequeno porte e Grupo 2, com problemas de médio e grande portes. As classes de problemas foram definidas pelo número de tarefas ($n$), número de máquinas ($m$) e pelo cenário relativo aos prazos das tarefas. Para cada classe, foram gerados aleatoriamente 100 problemas visando reduzir o erro amostral.

Conforme a maioria dos trabalhos de sequenciamento da produção (e.g. Li et al., 2015; e Vallada, Ruiz e Minella, 2008), os tempos de processamento foram gerados no intervalo $U$[1,99]. No Grupo 1, os parâmetros foram: $n \in \{5, 6, 7, 8, 10\}$ e $m \in \{2, 3, 5\}$. E no Grupo 2, os

parâmetros consistiram de: $n \square\{15, 20, 30, 50, 80, 100\}$ e $m \square\{5, 10, 15, 20\}$. Estes valores foram escolhidos de forma a abranger uma significativa gama de problemas de diversos tamanhos.

Devido à praticamente inexistência de experimentação computacional nas publicações abordando tarefas *just-in-time*, optou-se por gerar os prazos das tarefas seguindo o método utilizado por Armentano e Ronconi (2000) e Ronconi e Birgin (2012), que utiliza a distribuição uniforme no intervalo $[P(1–T–R/2), P(1–T+R/2)]$, onde $T$ e $R$ são dois parâmetros denominados fator de atraso e faixa de prazos, respectivamente, e $P$ é o limitante inferior de Taillard (1993) para o *makespan*, definido como:

$$P = \max\left\{\max_{1\le k\le m}\left\{\sum_{j=1}^{n} p_{jk} + \min_{j}\sum_{q=1}^{k-1} p_{jq} + \min_{j}\sum_{q=k+1}^{m} p_{jq}\right\}, \max_{j}\sum_{k=1}^{m} p_{jk}\right\}.$$

A partir da variação de $T$ e $R$, foram obtidos os seguintes cenários:

- Cenário 1: baixo fator de atraso ($T$=0,2) e pequena faixa de prazos ($R$=0,6);
- Cenário 2: baixo fator de atraso ($T$=0,2) e ampla faixa de prazos ($R$=1,2);
- Cenário 3: alto fator de atraso ($T$=0,4) e pequena faixa de prazos ($R$=0,6);
- Cenário 4: alto fator de atraso ($T$=0,4) e ampla faixa de prazos ($R$=1,2).

Com estes parâmetros, obteve-se 6.000 problemas do Grupo 1: 5 opções de número de tarefas, 3 opções de número de máquinas, 4 cenários e 100 problemas por classe (5*3*4*100=6.000). E foram gerados 9.600 problemas do Grupo 2: 6 opções de número de tarefas, 4 opções de número de máquinas, 4 cenários e 100 problemas por classe (6*4*4*100=9.600). Ambos os grupos totalizam os 15.600 problemas resolvidos.

Foi utilizado o sistema operacional Windows e o ambiente de programação Delphi. As configurações da máquina são as seguintes: processador Pentium Dual-Core com 2.0 GHz de frequência e 3.0 GB de memória RAM.

**4.2 Análise dos resultados**

Os resultados obtidos na experimentação computacional foram avaliados em termos da eficácia, ou seja, a qualidade da solução dos métodos, e também da eficiência computacional, verificada por meio do tempo médio de computação medido em milissegundos (ms).

Para a comparação do desempenho dos problemas do Grupo 1, a melhor solução foi obtida por um método de enumeração, tal como Laha e Sarin (2009), enquanto no Grupo 2, foi considerada a melhor solução fornecida pelos métodos implementados.

É importante esclarecer que o método de enumeração fornece uma solução de referência para o problema tratado e, embora enumere as $n!$ permutações de tarefas possíveis, não garante a solução ótima, uma vez que a programação é composta pelo sequenciamento e o cronograma. Ou seja, para uma mesma sequência é possível estabelecer diversas programações possíveis, com diferentes instantes de início para cada operação, acarretando várias possibilidades de valores do número de tarefas *just-in-time*. O método de enumeração executa o *Procedimento de ajuste de cronograma* definido a cada uma das $n!$ sequências geradas e considera a melhor sequência encontrada, porém não examina todas as possibilidades de ajuste. Este procedimento foi aplicado apenas nos problemas do Grupo 1, porque o tempo que seria consumido no Grupo 2 seria inviável.

A medida mais comumente usada na literatura (e.g. Vallada, Ruiz e Minella, 2008; e Laha e Sarin, 2009) para comparar o desempenho dos métodos de solução é o desvio relativo percentual (*relative percentage deviation* – RPD) que, adaptado ao problema de maximização tratado, é calculado da seguinte forma:

$$\square\square\square = \left(\frac{\square_{\square\square\square\square} - \square_{\square\square\square}}{\square_{\square\square\square\square}}\right).100, \qquad (4.1)$$

onde $\square_{\square\square\square}^{\square\square\square\square}$ é a melhor solução obtida para o número de tarefas *just-in-time* e $\square_{\square\square\square}^{\square}$ o número de tarefas *just-in-time* obtido pelo método de solução que está sendo avaliado. Quanto menor o RPD de um método, melhor é o seu desempenho; e RPD zero indica que o método forneceu a melhor solução encontrada.

A análise global dos resultados dos problemas resolvidos mostrou que a heurística H6 tem desempenho claramente superior às demais, com RPD de 0,2%, tanto no Grupo 1 como no Grupo 2, como pode ser constatado na Tabela 1. Isto significa que a solução fornecida pelo método H6 teve um desvio médio de 0,2% da melhor solução encontrada, um desempenho muito significativo em se tratando de uma heurística construtiva.

Tabela 1 – *Ranking* do desempenho global (RPD) dos métodos de solução propostos

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Grupo 1** | **H6** | **H5** | **H2** | **H1** | **H4** | **H3** | **H9** | **H8** | **H10** | **H7** |
| | 0.2 | 0.4 | 1.1 | 1.2 | 3.7 | 7.0 | 9.6 | 16.2 | 21.1 | 41.7 |
| **Grupo 2** | **H6** | **H5** | **H2** | **H1** | **H4** | **H3** | **H9** | **H10** | **H8** | **H7** |
| | 0.2 | 0.3 | 0.6 | 2.4 | 7.0 | 11.9 | 31.4 | 61.4 | 68.1 | 76.6 |

O resultado da heurística H6 é bastante próximo do obtido pela H5, que ficou em segundo lugar no desempenho global, com RPD de 0,4% no Grupo 1 e de 0,3%, no Grupo 2, ainda conforme a Tabela 1. O funcionamento de ambas as heurísticas é muito similar, considerando a regra EDD na ordenação inicial e deslocando iterativamente a primeira tarefa atrasada para o final da sequência. A heurística H6 se diferencia da H5 por empregar ao final da execução as buscas nas vizinhanças de inserção e de permutação. Percebe-se portanto que embora esta etapa de melhoria tenha conferido algum progresso no desempenho, as heurísticas já forneceram resultados muito bons mesmo antes destas buscas.

Seguindo a ordem de superioridade dos métodos, as heurísticas H2 e H1 ficaram respectivamente em terceiro e quarto lugares, as duas também semelhantes no seu procedimento, considerando a regra EDD na ordenação inicial e em seguida empregando o método de inserção na construção da sequência. A diferença é que a heurística H2 realiza a busca nas vizinhanças a cada tentativa de inserção da nova tarefa na sequência.

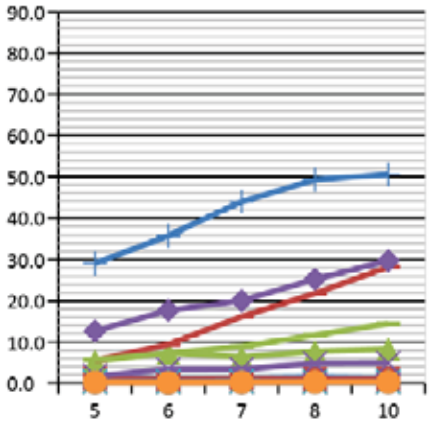
Pode-se notar ainda na Tabela 1 que a ordem de superioridade dos métodos permaneceu quase a mesma nos Grupos 1 e 2, com a única exceção da inversão entre as heurísticas H8 e H10 na oitava e nona colocações. Outra observação é que nos cinco piores métodos, os desvios no Grupo 1 foram bem menores do que os do Grupo 2, indicando maior dispersão na solução dos métodos nos problemas de médio e grande portes.

O pior método foi a heurística H7, que faz a ordenação inicial pela regra MST e em seguida desloca iterativamente a primeira tarefa atrasada para o final da sequência. Ela é muito parecida com a heurística H5, diferenciando apenas na ordenação inicial (que é feita pela regra EDD na H5). Entretanto, a discrepância nos resultados é bem notória. Isto mostra que, dentre as duas opções utilizadas de regra de ordenação inicial, a EDD é sempre mais vantajosa que a MST, como também ocorreu com todos os pares de heurísticas que se diferenciam apenas por essas regras: H1 e H3, H2 e H4, H5 e H7, H6 e H8, H9 e H10.

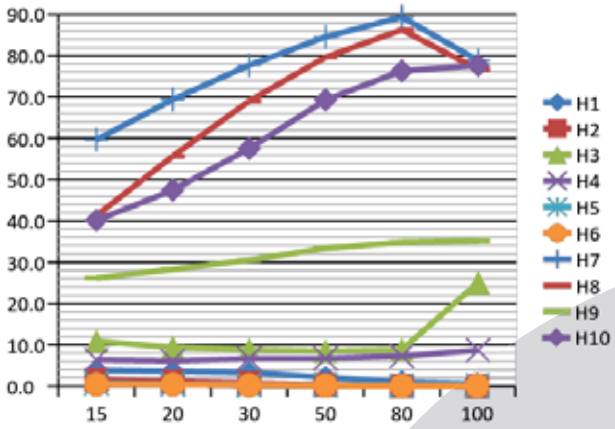
Uma análise detalhada por grupo e porte de problema pode ser feita nos gráficos das Figuras 1 e 2, que apresentam o desempenho dos métodos em relação ao RPD para cada opção do número de tarefas e de máquinas, respectivamente.

**RPD (%)**

**RPD (%)**




**Número de tarefas (Grupo 1)**




**Número de tarefas (Grupo 2)**

Figura 1 – Comparação do desempenho (RPD) dos métodos por grupo e número de tarefas

Como se observa nos dois gráficos da Figura 1, os melhores métodos, H6, H5, H2 e H1, tiveram desempenho considerado estável tanto no Grupo 1 como no 2, com valores de RPD bem baixos, próximos de zero. A exceção é com a heurística H1 no Grupo 2, que tem desvios relativos decrescentes, indo de 3,9% com 15 tarefas a 0,6% com 100 tarefas.

Já as demais heurísticas tiveram valores de RPD em geral crescentes com o aumento do número de tarefas, exceto a H7 e H8 no Grupo 2, que tiveram uma queda ao atingir problemas com 100 tarefas. Além disso, o método H3 mostrou um leve decrescimento em problemas de 15 a 50 tarefas, e em seguida um aumento no RPD para 80 e 100 tarefas.

Os gráficos da Figura 2, que apresentam resultados específicos para cada opção do número de máquinas em cada um dos grupos, mostram grande estabilidade com variações relativamente pequenas nas amplitudes dos RPD. Isto pode indicar que o número de máquinas não seja um fator relevante no desempenho dos métodos propostos e no problema tratado.

**RPD (%)**

**RPD (%)**

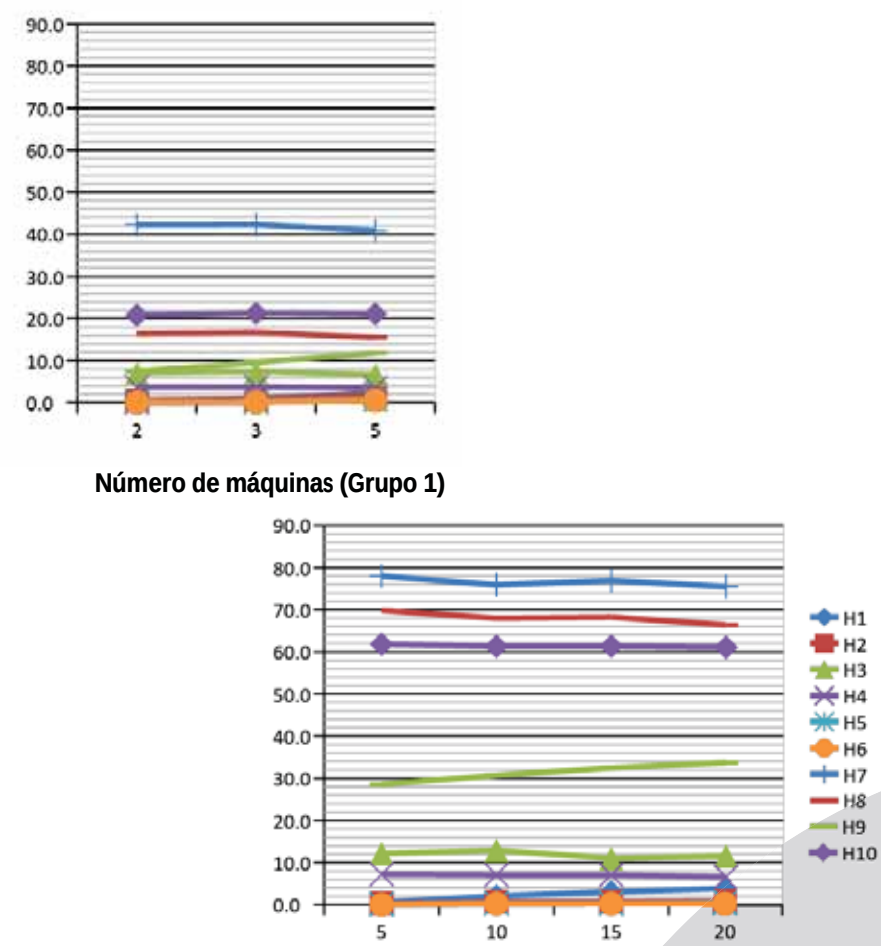



Figura 2 – Comparação do desempenho (RPD) dos métodos por grupo e número de máquinas

A Tabela 2 apresenta os resultados de cada grupo para cada um dos quatro cenários definidos, que são bastante consistentes com as análises apresentadas até agora em termos de superioridade dos métodos. Não há diferenças significativas entre os resultados dos diferentes cenários, sugerindo que o fator de atraso e a faixa de prazos não exercem influência relevante.

Uma única revelação interessante é que as duas melhores heurísticas, H6 e H5, apresentam resultados idênticos em ambos os grupos nos cenários 2 e 4, caracterizados pela ampla faixa de prazos. Ou seja, quando o intervalo dos prazos é maior, as buscas vizinhanças não oferecem melhorias.

Tabela 2 – Desempenho (RPD) dos métodos por grupo e cenário

| | Cenário | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Grupo 1** | **1** | 0.9 | 0.8 | 8.7 | 4.6 | 0.4 | **0.3** | 42.2 | 16.5 | 10.0 | 20.8 |
| | **2** | 1.0 | 0.7 | 4.7 | 2.2 | **0.2** | **0.2** | 45.7 | 19.2 | 4.3 | 20.0 |
| | **3** | 1.9 | 1.7 | 9.3 | 5.1 | 0.8 | **0.5** | 39.3 | 14.7 | 15.0 | 22.1 |
| | **4** | 1.0 | 1.0 | 5.5 | 2.7 | **0.1** | **0.1** | 39.8 | 14.5 | 9.2 | 21.5 |
| **Grupo 2** | **1** | 1.7 | 0.7 | 15.0 | 8.9 | 0.4 | **0.2** | 76.3 | 68.1 | 39.4 | 61.6 |
| | **2** | 1.6 | 0.5 | 7.8 | 4.5 | **0.1** | **0.1** | 78.6 | 71.3 | 13.8 | 62.0 |
| | **3** | 3.7 | 1.0 | 15.2 | 9.4 | 0.7 | **0.4** | 75.1 | 65.4 | 43.0 | 60.9 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **4** | 2.5 | 0.3 | 9.7 | 5.3 | **0.1** | **0.1** | 76.3 | 67.6 | 29.3 | 61.0 |

A Tabela 3 mostra o consumo médio de CPU, medido em milissegundos, de cada heurística em ambos os grupos e também do método de enumeração (ME) para os problemas de pequeno porte.

Tabela 3 – Eficiência computacional (tempo médio de CPU, em milissegundos) dos métodos por grupo

| | **H1** | **H2** | **H3** | **H4** | **H5** | **H6** | **H7** | **H8** | **H9** | **H10** | **ME** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Grupo 1** | 0.1 | 0.3 | 0.0 | 0.3 | 0.0 | 0.1 | 0.0 | 0.1 | 0.1 | 0.0 | 1483.7 |
| **Grupo 2** | 68.0 | 7895.5 | 72.6 | 7820.2 | 1.2 | 391.5 | 1.3 | 190.5 | 79.0 | 56.3 | - |

Nos problemas de pequeno porte, o tempo de computação foi praticamente zero. E nos problemas de médio e grande portes, os maiores tempos foram consumidos pelas heurísticas H2 e H4, com quase 8 segundos em média cada, que é um tempo relativamente alto comparado aos demais métodos. Todas as demais heurísticas tiveram tempos bem menores do que um segundo, demonstrando a viabilidade da sua utilização na prática. Vale salientar que a heurística H5, que ficou em segundo lugar, com resultado muito próximo da H6, que ficou em primeiro, consumiu pouco mais do que um segundo em média, indicando elevada eficiência computacional.

## 5. Considerações finais

Este estudo atingiu seu objetivo de desenvolver e implementar métodos heurísticos construtivos eficazes e eficientes para resolver o problema de maximização do número de tarefas *just-in-time* em *flow shop* permutacional, conforme demonstrado por experimentação computacional. O tempo de execução de todos os métodos foi aceitável, mesmo em problemas de grande porte.

As melhores heurísticas, H6 e H5, tiveram desvios médios da melhor solução encontrada próximos de zero (em torno de 0,2% e 0,4%, respectivamente) para problemas com até 10 tarefas e 5 máquinas. Nos problemas de pequeno porte, conjuntamente elas fornecem a melhor solução encontrada em 99,7% dos problemas-teste. Os métodos H6 e H5 consideram como regra de prioridade inicial a ordenação dada pela EDD e em seguida deslocam iterativamente a primeira tarefa atrasada para o final da sequência. Embora com resultados bastante próximos, o método H6 se constitui numa melhoria do H5 por utilizar buscas em vizinhanças.

Nesta pesquisa, o foco da solução do problema foi a redução do intervalo entre a data de término da última operação da tarefa e o seu prazo, visando otimizar o número de tarefas que estes dois instantes coincidem. Isto permitiu inclusive um ajuste no cronograma das tarefas, acarretando a possibilidade de inserção de tempo ocioso entre as operações. Assim, não houve preocupação com as primeiras operações de cada tarefa, ou seja, em aproximar a execução dessas operações. Uma reprogramação com essas aproximações reduziria o tempo ocioso entre as operações e possivelmente minimizaria também a duração total da programação (*makespan*). Portanto, sugere-se para trabalhos futuros a consideração de funções-objetivo multicritério, incluindo medidas de fluxo (e não apenas de atraso e/ou adiantamento), como *makespan* e/ou *flowtime*.

**Agradecimentos**: Este trabalho teve apoio parcial da FAPEG, FAPESP, CAPES e CNPq.
Agradecemos aos revisores anônimos pela leitura cuidadosa do texto.

## Referências

**Armentano, V.A. e Ronconi, D.P.** (2000), Minimização do tempo total de atraso no problema de *flow shop* com *buffer* zero através de busca tabu, *Gestão & Produção*, 7(3), 352-363.
**Baker, K.R. e Scudder, G.D.** (1990), Sequencing with earliness and tardiness penalties: a review, *Operations Research*, 38(1), 22-36.

**Baker, K.R. e Trietsch, D.**, *Principles of sequencing and scheduling*, John Wiley & Sons, New York, 2009.
**Choi, B.-C. e Yoon, S.-H.** (2007), Maximizing the weighted number of just-in-time Jobs in flow shop scheduling, *Journal of Scheduling*, 10, 237-243.
**Elalouf, A., Levner, E. e Tang, H.** (2013), An improved FPTAS for maximizing the weighted number of just-in-time jobs in a two-machine flow shop problem, *Journal of Scheduling*, 16, 429-435.
**Fernandes, F.C.F. e Godinho Filho, M.**, *Planejamento e controle da produção*: dos fundamentos ao essencial, Atlas, São Paulo, 2010.
**Gerstl, E., Mor, B. e Mosheiov, G.** (2015), A note: maximizing the weighted number of just-in-time jobs on a proportionate flowshop, *Information Processing Letters*, 115, 159-162.
**Hodgson, T.J.** (1977), A note on single machine sequencing with random processing times, *Management Science*, 23, 1144-1146.
**Jung, C.F.**, *Metodologia para pesquisa & desenvolvimento:* aplicada a novas tecnologias, produtos e processos, Rio de Janeiro, Axcel Books, 2004.
**Józefowska, J.**, *Just-in-time scheduling*: models and algorithms for computer and manufacturing systems, Springer Science, New York, 2007.
**Krajewski, L., Ritzman, L. e Malhotra, M.**, *Administração da produção e operações*, Pearson Prentice Hall, São Paulo, 2009.
**Laha, D. e Sarin, S.C.** (2009), A heuristic to minimize total flow time in permutation flow shop, *Omega – The International Journal of Management Science*, 37, 734-739.
**Li, X., Chen, L., Xu, H. e Gupta, J.N.D.** (2015), Trajectory scheduling methods for minimizing total tardiness in a flowshop, *Operations Research Perspectives*, 2, 13-23.
**Nawaz, M., Enscore Jr., E.E. e Ham, I.** (1983), A heuristic algorithm for the *m*-machine *n*-job flow-shop sequencing problem, *OMEGA – The International Journal of Management Science*, 11(1), 91-95.
**Ríos-Solís, Y.A. e Ríos-Mercado, R.Z.** (Eds.) *Just-in-Time Systems*, Springer Sciences, New York, 2012.
**Ronconi, D.P. e Birgin, E.G.**, Mixed-integer programming models for flow shop scheduling problems minimizing the total earliness and tardiness. *In:* Ríos-Solís, Y.A. e Ríos-Mercado, R.Z. (Eds.) *Just-in-Time Systems*, Springer Sciences, New York, 2012.
**Shabtay, D.** (2012), The just-in-time scheduling problem in a flow-shop scheduling system, *European Journal of Operational Research*, 216, 521-532.
**Shabtay, D. e Steiner, G.** (2012), Scheduling to maximize the number of just-in-time jobs: a survey *In:* Ríos-Solís, Y.A. e Ríos-Mercado, R.Z. (Eds.) *Just-in-Time Systems*, Springer Sciences, New York, 2012.
**Shabtay, D. e Bensoussan, Y.** (2012), Maximizing the weighted number of just-in-time jobs in several two-machine scheduling systems, *Journal of Scheduling*, 15, 39-47.
**Shabtay, D., Bensoussan, Y. e Kaspi, M.** (2012), A bicriteria approach to maximize the weighted number of just-in-time jobs and to minimize the total resource consumption cost in a two-machine flow-shop scheduling system, *International Journal of Production Economics*, 136, 67-74.
**Taillard, E.** (1993), Benchmarcks for basic scheduling problems, *European Journal of Operational Research*, 64, 278-285.
**Vallada, E., Ruiz, R. e Minella, G.** (2008), Minimizing total tardiness in the *m*-machine flowshop problem: a review and evaluation of heuristics and metaheuristics, *Computers & Operations Research*, 35, 1350-1373.