ANÁLISE COMPARATIVA DE HEURÍSTICAS PARA MINIMIZAÇÃO DE ADIANTAMENTOS E ATRASOS EM *FLOW SHOP* COM TEMPOS DE *SETUP*

John Lennon Damas David
UFG/Campus Catalão
johnlennon_13@yahoo.com.br

Hélio Yochihiro Fuchigami
UFG/Campus Catalão
helio@catalao.ufg.br

RESUMO

Neste trabalho foram propostas seis heurísticas construtivas para programação de *flow shop* com tempos de *setup* independentes da sequência e minimização da soma dos adiantamentos e atrasos das tarefas. O algoritmo com o procedimento mais simples, que apenas sequencia pela regra de prioridade EDD (*Earliest Due Date*) e aplica uma etapa de melhoria, forneceu os melhores resultados, com mais de 45% de sucesso.

PALAVARAS-CHAVE: Programação da produção, *Flow shop*, Heurísticas, Atrasos, Adiantamentos.

1. Introdução

Este estudo aborda o problema de programação da produção em sistemas *flow shop* com tempos de *setup* explícitos, ou seja, separados dos tempos de processamento das tarefas, e independentes da sequência de execução. A medida de desempenho considerada neste problema é a minimização da soma dos adiantamentos e atrasos das tarefas. Esta é uma abordagem da filosofia *Just-in-Time*, que considera que o estoque gerado pela conclusão antecipada da tarefa é um desperdício, por ocupar espaço e representar capital parado, e que os atrasos geram punições e/ou perda de oportunidade.

Os problemas de programação da produção são alvo de incontáveis trabalhos de Pesquisa Operacional, desde a publicação de Johnson (1954), que aborda *flow shop* com duas e três máquinas. Inúmeras pesquisas têm sido realizadas na busca de métodos exatos e heurísticos para programação de *flow shop*.

Conforme Linn e Zhang (1999) e Quadt e Kuhn (2007), a maioria dos autores trata o problema de *flow shop* com medidas de desempenho relacionadas à passagem das tarefas pelo sistema, tais como minimização do *makespan* (duração total da programação) ou do tempo médio de fluxo. Recentemente, os trabalhos passaram a abordar este ambiente com medidas relacionadas ao atraso e/ou adiantamento, que devido a sua complexidade, geralmente são abordados por meio de métodos heurísticos.

Este objetivo está sendo cada vez mais alvo dos temas de estudos de problemas de programação da produção, principalmente em sistemas de produção *Just-in-Time*, que visam reduzir, entre outras medidas, os adiantamentos e atrasos.

2. Descrição do problema

O problema em estudo consiste no ambiente de *flow shop* com *m* máquinas e *n* tarefas com diferentes datas de entrega (d_j) e tempos de *setup* em cada máquina *k* independentes da sequência (s_{jk}). O objetivo é a minimização da soma dos adiantamentos (E_j) e atrasos (T_j) das tarefas. Na conhecida notação de três campos, o problema é representado por $\boldsymbol{Fm|s_{jk},\ d_j|\sum(E_j+T_j)}$. O objetivo é reduzir o intervalo de tempo entre o término da execução de cada tarefa e o seu respectivo prazo de entrega.

No *flow shop*, as tarefas devem ter um fluxo de processamento unidirecional e passar por todas as máquinas na mesma sequência. Cada tarefa tem *m* operações e cada operação op_{jk} de cada tarefa *j* é realizada na máquina *k* ($k = 1, ..., m$). Os tempos de processamento de cada tarefa nas diferentes máquinas (p_{jk}) são conhecidos previamente e considerados fixos.

Quando, em cada máquina, a ordem de processamento em das tarefas for a mesma, tem-se o ambiente de produção *flow shop permutacional*, onde o número de programações possíveis para *n* tarefas é $n!$ No caso no *flow shop* não permutacional, o número de programações é $(n!)^m$.

No problema tratado foram consideradas as seguintes premissas: cada máquina está disponível continuamente, não havendo interrupções, e pode processar apenas uma tarefa de cada vez; cada tarefa pode ser processada por apenas uma máquina de cada vez; os tempos de processamento das tarefas nas diversas máquinas são determinados e fixos; as tarefas têm diferentes datas de entrega (prazos); os tempos de preparação (*setup*) das operações nas diversas máquinas são separados dos tempos de processamento, independentes da sequência e podem ser antecipados (iniciar antes do término do processamento da tarefa na máquina anterior); as operações nas diversas máquinas, uma vez iniciadas não devem ser interrompidas; as tarefas devem ser processadas em todas as máquinas; as datas de entrega de cada tarefa são diferentes de zero e conhecidas previamente; a execução de uma tarefa em uma máquina só pode começar após o término da sua operação no estágio anterior e desde que a máquina já esteja preparada; o problema é estático, ou seja, as datas de liberação são iguais e consideradas iguais a zero.

3. Métodos de solução

Neste estudo, foram propostos cinco heurísticas e também um método de solução aleatório (Algoritmo 6), para servir como base de comparação para os demais métodos propostos, assim como foi feito por Moursli (1999), conforme apresentado a seguir.

ALGORITMO 1

Passo 1 - Ordene as tarefas de acordo com a regra EDD (*Earliest Due Date*).

Passo 2 - Programe sequencialmente as tarefas nas máquinas e calcule os adiantamentos e atrasos.

Passo 3 (Melhoria)

Passo 3a) Se a última tarefa da última máquina estiver adiantada, atrase-a fazendo sua data de término coincidir com a sua data de entrega, ou seja, $C_j = d_j$.

Passo 3b) A partir da penúltima tarefa até a primeira, se houver adiantamento e tempo ocioso, atrase cada tarefa, reduzindo ao mínimo possível esse adiantamento, em relação ao seu prazo ou o intervalo ocioso.

ALGORITMO 2

Passo 1 - Calcule para cada tarefa *j*, o valor do índice I_j:

$$I_j = d_j - \left(\max\{ s_{j1} + p_{j1}; s_{j2} \} + \sum_{k=2}^{m} p_{jk} \right)$$

Passo 2 - Ordene as tarefas de acordo com os índices I_j de forma crescente.

Passo 3 - Programe sequencialmente as tarefas nas máquinas e calcule os adiantamentos e atrasos.

Passo 4 (Melhoria)

Passo 4a) Se a última tarefa da última máquina estiver adiantada, atrase-a fazendo sua data de término coincidir com a sua data de entrega, ou seja, $C_j = d_j$.

Passo 4b) A partir da penúltima tarefa até a primeira, se houver adiantamento e tempo ocioso, atrase cada tarefa, reduzindo ao mínimo possível esse adiantamento, em relação ao seu prazo ou o intervalo ocioso.

O índice I_j objetiva priorizar as tarefas que contenham os menores valores de folga, ou seja, que apresentem a menor diferença entre a sua data de entrega e a sua carga total. A carga é calculada considerando a possibilidade de antecipação do *setup* na segunda máquina, ou seja, o maior valor entre o *setup* mais o processamento da tarefa na primeira máquina ($s_{j1}+p_{j1}$) e o *setup* na segunda máquina (s_{j2}), somando-se os tempos de processamento das máquinas 2 a *n*, sob a hipótese otimista de todos os *setups* nestas máquinas serem antecipados.

ALGORITMO 3

Passo 1 - Ordene as tarefas de acordo com a regra EDD.

Passo 2 - Programe sequencialmente as tarefas nas máquinas e calcule os adiantamentos e atrasos.

Passo 3 - Selecione a tarefa com o maior adiantamento e insira-a em todas as posições possíveis da sequência, mantendo a programação com o melhor valor da função objetivo.

Passo 4 - Se uma nova programação foi encontrada, repita o *Passo 3*. Caso contrário, *Pare*.

ALGORITMO 4

Passo 1 - Ordene as tarefas de acordo com a regra EDD.

Passo 2 - Programe sequencialmente as tarefas nas máquinas e calcule os adiantamentos e atrasos.

Passo 3 (Inserção com melhoria)

Passo 3a) Selecione a tarefa com o maior adiantamento e insira-a em todas as posições possíveis da sequência.

Passo 3b) Para cada posição verificada, se a última tarefa da última máquina estiver adiantada, atrase-a fazendo sua data de término coincidir com a sua data de entrega, ou seja, $C_j = d_j$.

Passo 3c) A partir da penúltima tarefa até a primeira, se houver adiantamento e tempo ocioso, atrase cada tarefa, reduzindo ao mínimo possível esse adiantamento em relação ao seu prazo ou ao intervalo ocioso.

Passo 3d) Mantenha a programação com o melhor valor da função objetivo.

Passo 4 - Se uma nova programação foi encontrada, repita o *Passo 3*. Caso contrário, *Pare*.

ALGORITMO 5

Passo 1 - Ordene as tarefas de acordo com a regra EDD.

Passo 2 - Programe sequencialmente as tarefas nas máquinas e calcule os adiantamentos e atrasos.

Passo 3 - Selecione a tarefa com o maior adiantamento e insira-a em todas as posições possíveis da sequência, mantendo a sequência com o melhor valor da função objetivo.

Passo 4 - Se uma nova programação foi encontrada, repita o *Passo 3*. Caso contrário, vá

para o *Passo 5*.

Passo 5 (Melhoria)

Passo 5a) Se a última tarefa da última máquina estiver adiantada, atrase-a fazendo sua data de término coincidir com a sua data de entrega, ou seja, $C_j = d_j$.

Passo 5b) A partir da penúltima tarefa até a primeira, se houver adiantamento e tempo ocioso, atrase cada tarefa, reduzindo ao mínimo possível esse adiantamento, em relação ao seu prazo ou o intervalo ocioso.

ALGORITMO 6

Passo 1 - Associe um número aleatório a cada tarefa e ordene-as de forma crescente.

Passo 2 - Programe sequencialmente as tarefas nas máquinas.

4. Resultados

Na experimentação computacional foram considerados problemas com n = 6, 8, 10, 20, 50 e 100 tarefas, m = 2, 3, 5, 10 e 20 máquinas, tempos de processamento no intervalo U[1,99] e tempos de *setup* nos intervalos U[1,49], U[1,124] e U[50,99]. As datas de entrega foram geradas conforme Ronconi e Birgin (2012), 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 dispersão, respectivamente, e P é um limitante inferior para o *makespan*, em que neste trabalho foi utilizada uma adaptação para *setup* explícito daquele proposto por Taillard (1993). Os valores de T foram 0,2 e 0,4 e os de R 0,6 e 1,2. Para cada classe dos parâmetros descritos foram gerados 100 problemas, totalizando 36.000 problemas resolvidos.

Os resultados foram analisados por meio da porcentagem de sucesso (percentual de vezes em que o método forneceu a melhor solução), desvio relativo e tempo médio de computação (medido em milissegundos). A geração dos problemas e a implementação dos métodos foram feitas em linguagem Delphi em uma máquina com as seguintes configurações: processador Intel Core i5 com 2,67 GHz de frequência, memória RAM de 14 Gb e sistema operacional Windows.

As Tabelas 4.1 e 4.2 apresentam os resultados gerais dos métodos, ordenados de acordo com o desempenho.

TABELA 4.1 – Porcentagem de sucesso dos métodos de solução

Método	**Algoritmo 1**	**Algoritmo 4**	**Algoritmo 5**	**Algoritmo 2**	**Algoritmo 3**	**Algoritmo 6**
% Sucesso	45,73	42,56	40,40	17,43	15,79	2,16

TABELA 4.2 – Desvio relativo médio dos métodos de solução

Método	**Algoritmo 1**	**Algoritmo 4**	**Algoritmo 2**	**Algoritmo 5**	**Algoritmo 3**	**Algoritmo 6**
% DRM	18,87	29,68	30,51	52,11	184,66	601,22

Os Algoritmos 1, 4 e 5 obtiveram os melhores resultados, com média acima de 45%, 42% e 40% de sucesso, respectivamente. Além disso, os Algoritmos 1 e 4 obtiveram os menores desvios relativos médios. É interessante notar que o Algoritmo 1 é o que possui o procedimento mais simples e obteve o melhor desempenho. Desconsiderando os empates, o percentual de sucesso conjunto destes três melhores algoritmos foi de 73,2%, o que é bastante significativo.

Pode ser verificado na Tabela 4.1 que a soma do percentual de sucesso perfaz 164,07%, indicando que houve um grande número de empates no melhor resultado dos métodos.

Conforme esperado, o Algoritmo 3 teve desempenho inferior aos Algoritmos 4 e 5, que

possuem a etapa de melhoria, comprovante que esse passo trouxe um efeito positivo na resolução do problema.

A ordem de superioridade dos métodos foi quase a mesma considerando a porcentagem de sucesso e o desvio relativo médio. A única exceção foi com a inversão dos Algoritmos 5 e 2, como pode ser observado nas Tabelas 4.1 e 4.2.

Os resultados do Algoritmo aleatório 6 evidenciam a viabilidade de aplicação dos cinco primeiros algoritmos, pelo seu pior desempenho, com pouco mais de 2% de sucesso e desvio relativo de mais de 600%, extremamente distante dos demais. Para efeito de comparação, dentre os cinco melhores, o Algoritmo 3 foi o que obteve o pior resultado (cerca de 15% de sucesso e 183% de desvio relativo).

Houve variabilidade nos resultados dos métodos em relação aos cenários configurados pelo fator de atraso e pela faixa de dispersão, assim como nos diferentes portes de problema (número de tarefas e de máquinas). Já os intervalos de *setup* considerado não influenciaram significativamente.

O custo computacional mostrou-se desprezível, pois o tempo máximo de CPU consumido por um problema foi de 63 ms.

5. Conclusões e perspectivas futuras

Os resultados da experimentação computacional elucidaram a viabilidade de aplicação das heurísticas apresentadas na solução do problema tratado. Uma vez que o tempo computacional não constitui fator restritivo, uma proposta viável seria a criação de uma heurística híbrida, que verifica os resultados dos três melhores métodos – Algoritmos 1, 4 e 5, e forneça a melhor solução encontrada, atingindo assim mais de 73% de sucesso.

Para futuros trabalhos, sugere-se a consideração de novas restrições, como tarefas com prazos de entrega comuns ou então janelas de entrega, e também a inclusão de outras medidas na função objetivo, como tempo médio de fluxo ou duração total da programação. Além disso, as soluções de problemas de pequeno porte poderiam ser comparados com a fornecida por algum método de solução exata.

Referências

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

Linn, R.; Zhang, W. (1999). Hybrid flow shop scheduling: a survey. *Computers and Industrial Engineering*, 37, 57-61.

Moursli, O. (1999). *Scheduling the hybrid flowshop: branch and bound algorithms.* Tese (Doutorado) – Faculte des Sciences Economiques, Sociales et Politiques, Universite Catholique de Louvain, Louvain. Bélgica.

Quadt, D.; Kuhn, H. (2007). A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research*, 178, 686-698.

Ronconi, D. P.; Birgin, E. G. (2012). Mixed-integer programming models for flow shop scheduling problems minimizing the total earliness and tardiness. In: *Just-in-Time Systems*, Y.A. Ríos-Solís and R.Z. Ríos-Mercado (Eds.), Springer Series on Optimization and Its Applications, P.M. Pardalos and Ding-Zhu Du (Series eds.).

Taillard, E. (1993). Benchmarcks for basic scheduling problems. *European Journal of Operational Research*, 64, 278-285.