https://eventos.utfpr.edu.br//sicite/sicite2020

Comparação entre soluções iniciais heurísticas para o método *Iterated Greedy*

Comparison between initial heuristic solutions for the Iterated Greedy method

Pedro Eduardo Hernandes Natal
pedroehn@hotmail.com
Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil

Mauricio Iwama Takano
takano@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil

Cassio Henrique dos Santos Amador
cassioamador@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil

RESUMO

O sequenciamento é um método que organiza as operações, de modo que ele agende tarefas para alcançar um objetivo que pode ser, por exemplo, diminuir o tempo de processamento da última tarefa na última máquina. Observando a necessidade de resolver o problema de sequenciamento, muitos métodos diferentes foram criados e, neste trabalho, será utilizado o *Iterated Greedy* (*IG*). A qualidade da solução inicial pode melhorar a qualidade da solução final do algoritmo *IG*. Os métodos heurísticos usados para fornecer as soluções iniciais são: MinMax (*MM*), profile fitting (*PF*), Nawaz-Enscore-Ham (*NEH*), *PW*, weighted profile fitting (*wPF*) e aleatório (a fim de comparação). A eficiência dos algoritmos será avaliada usando estas soluções iniciais diferentes.

PALAVRAS-CHAVE: Pesquisa operacional. Programação heurística. Algoritmos computacionais.

ABSTRACT

Scheduling is a method that arranges operations, so it schedules jobs to achieve an objective that can be, for example, to decrease the processing time of the last job in the last machine. Observing the need of solving the scheduling problem, many different methods were created and, in this work, it'll be used the *iterated greedy* (*IG*). The quality of the initial solution may improve the quality of the final solution of the *IG* algorithm. The heuristic methods used to provide the initial solutions are: minmax (*MM*), profile fitting (*PF*), Nawaz-Enscore-ham (*NEH*), *PW*, weighted profile fitting (*wPF*) e random (for comparison). The efficiency of the *IG* algorithm will be evaluated using those different initial solutions.

KEYWORDS: Operational research. Heuristic programation. Computational algorithms.

Recebido: 19 ago. 2020.
Aprovado: 01 out. 2020.

INTRODUÇÃO

Para otimizar um ou mais objetivos, o sequenciamento busca uma forma eficiente de selecionar e aplicar a melhor sequência de tarefas definindo as prioridades de ocorrências das mesmas em sistemas de manufatura.

Neste artigo, o Método *Iterated Greedy* foi escolhido como método de solução principal, como este método necessita de uma solução inicial, foram testados 6 métodos diferentes para se obter essa solução inicial, os quais serão explicadas posteriormente na seção de metodologia.

Os problemas ocorrem em um ambiente de *flowshop* em que há *m* máquinas e *n* tarefas e todos as tarefas devem seguir o mesmo padrão de fluxo de máquinas, por exemplo, todos as tarefas devem ser processadas primeiro na máquina um, depois na máquina dois até a última máquina.

Outra característica do problema é o bloqueio, se uma tarefa *j* é processada na máquina *k* e a próxima máquina *k + 1* não terminou de processar sua tarefa, ela permanece na máquina, bloqueando o início do processo da tarefa *j*+1.

Por fim, a última restrição considerada foi o setup variável em função da sequência, por exemplo, em um ambiente que exista 6 tarefas para serem processadas a máquina *k* demora t_1 de setup para sair do ajuste da tarefa 1 e receber a tarefa 4 enquanto a mesma máquina demora $t_2 \neq t_1$ para sair do ajuste da mesma tarefa 1 e receber a tarefa 6, ou seja a sequência de tarefas influencia diretamente no tempo de setup e da conclusão.

MATERIAL E MÉTODOS

De acordo com Pan e Ruiz (2014), um algoritmo *Iterated Greedy* (*IG*) consiste basicamente de algumas etapas. Primeiramente, uma solução inicial é criada, por meio da utilização de uma heurística construtiva de alta performance. Então um *loop* principal é rodado até que um critério de parada seja atingido, o critério utilizado neste artigo foi o limite do tempo computacional, que foi definido em *m*n*15* ms. Dentro deste *loop*, duas operações são iterativamente aplicadas. O primeiro operador é o *random destruction*, no qual alguns elementos da solução são removidos. O segundo operador é o *greedy reconstruction* que reinsere os elementos removidos de forma a criar uma nova solução completa. A reconstrução também utiliza uma heurística de alta performance. Depois que uma nova solução completa é obtida, um critério de aceitação é aplicado de forma a decidir se a solução substituirá a incumbente.

Opcionalmente, um procedimento de busca local pode ser aplicado, normalmente após a construção da solução inicial e antes do critério de aceitação em cada passagem do *loop* principal. Neste artigo, embora o método perca desempenho, a fim de reduzir o tempo computacional, a operação de inserção aprimorada foi retirada do algoritmo.

A ideia de avaliar e comparar os algoritmos foi baseada no estudo desenvolvido por Sanches et al. (2015), onde ele verifica métodos heurísticos, porém para o algoritmo branch and bound.

Outro estudo que serviu como base foi o de Takano e Nagano (2017), que foi um dos pioneiros a estudar o ambiente utilizado neste artigo com todas as restrições de setup e bloqueio aqui consideradas.

O primeiro algoritmo de solução inicial implementado foi o algoritmo *NEH*, através do método LPT (maior tempo de processamento), uma sequência parcial é gerada, e a primeira tarefa é definida, a segunda tarefa da sequência parcial é testada em todas as posições e fixada na que resulte menor valor da função objetivo, posteriormente a terceira tarefa da sequência parcial é testada em todas as posições e o ciclo se repete até que todas as *n* tarefas sejam testadas e alocadas.

Para o *NEH*, o pseudocódigo é mostrado a seguir:

Gere uma sequência inicial $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ usando a regra LPT (maior tempo de processamento)

$\pi := (\beta_1)$.

for k:=2 até *n* faça %

Pegue a tarefa β_k de β e teste em todas as *k* posições possíveis de π.

Insira β_k em π na posição que resulta no menor valor da função objetivo do problema

Return π

O *Profile Fitting* (*PF*) foi o segundo algoritmo heurístico implementado, ele consiste na determinação de uma sequência parcial através do método LPT. Começa os laços iterativos, primeiramente ele vai determinar a primeira tarefa como a inicial e calcular o tempo de partida de todas as máquinas até a última tarefa alocada (*K*) que inicialmente será apenas a primeira, depois vai testar cada uma das tarefas não alocadas como a tarefa *K+1* de forma a calcular o tempo de partida e a soma da ociosidade e do tempo de bloqueio. Através de uma equação que será apresentada no pseudocódigo a tarefa não alocada que tiver o menor valor de δ, se tornará a tarefa *k+1.* A seguir está o pseudocódigo.

Selecione a tarefa com a menor soma de tempo de processamento como a primeira tarefa de uma sequência parcial e denote a sequência parcial como $\beta = \beta_1$.

Defina $U = j - \{\beta_1\}$.

For k:=1 até n-2 faça:

Calcule o tempo de partida $d_{(k),i}, i = 1,2, \dots, m$ até a última tarefa β_k na sequência parcial $\beta = (\beta_1, \beta_2, \dots, \beta_k)$.

Para cada tarefa $j \in U$, como se fosse anexado e se tornasse o (k+1)ésima tarefa de β, calcule o tempo de partida $d_{(k+1),i}, for\ i = 1,2, \dots, m$, e a soma da ociosidade e do tempo de bloqueio $\delta_{j,k} = \sum_{k=1}^{m} (d_{(k+1),i} - d_{(k),i} - p_{j,i})$.

Selecione a tarefa que resulte no menor valor de $\delta_{j,k}$ como a (K+1)ésima tarefa de β, e remova a tarefa selecionada de U.

Endfor

A única tarefa restante de U é acrescida em β e se torna a enésima tarefa de β.

Return.

O MM é o terceiro método implementado, nele a primeira tarefa é aquela que tem o menor tempo de processamento na primeira máquina e a última tarefa é a que tem o menor tempo na última máquina, então os outros trabalhos consideram o caminho crítico que é composto pelas somas de valores máximos entre os tempos de processamento de máquinas consecutivas e pela soma de *m* tempos de processamento, o próximo trabalho é estabelecido para minimizar *S* na Eq. (1), sugerida por Ronconi (2004).

$$S = \alpha \sum_{l=1}^{m-1} \left| P_{cl} - P_{i,l+1} \right| + (1-\alpha) \sum_{k=1}^{m} P_{ck} \quad (1)$$

O algoritmo seleciona uma tarefa *c* como a tarefa diretamente consecutiva da tarefa já posicionada *i*, *Pxy* representa o tempo de processamento da tarefa *x* na máquina *y.* Por fim, *α* foi definido como 0.6, este valor foi tratado como a melhor opção de escolha segundo o autor do artigo.

O algoritmo *weighted PF* (*wPF*) também foi implementado, ele tem a mesma construção do *PF*, porém considera um peso diferente para cada máquina e diferentes posições na sequência. Os pesos são calculados pela Eq. (3) multiplicando a Eq. (2).

$$\delta_{j,k} = \sum_{i=1}^{m} w_i (d_{(k+1),i} - d_{(k),i} - p_{j,i}) \quad (2)$$

$$w_i = m/(i + \frac{k(m-i)}{n-2}) \quad (3)$$

Na equação acima o i representa a máquina variando de 1 até m, enquanto o k representa as tarefas que iniciam em 2 e vão até n-1, pois a única tarefa restante das não alocadas entrará na última posição das tarefas alocadas. D representa a data de início e p representa o tempo de processamento.

O procedimento de *PW* é explicado a seguir, com o mesmo propósito de diminuir o bloqueio e a ociosidade que o *PF*, ele determina a sequência de tarefas dividido em 2 etapas, a primeira é calcular a soma ponderada dos tempos de ociosidade e de bloqueio da tarefa que está sendo testada, similar ao método *wPF*

Selecione a tarefa com o menor valor de $\mathrm{f_{j,0}}$ como a primeira tarefa da sequência parcial $\beta = (\beta_1)$. Se as tarefas tiverem o mesmo valor de $\mathrm{f_{j,0}}$, selecione aquela com o mínimo valor de $x_{j,0}$.

Defina $\mathrm{U} = \mathrm{j} - \{\beta_1\}$.

For k:=1 até n-2 faça

Calcule o tempo de partida $d_{(k),i}, i = 1,2,\dots,m$ para a última tarefa β_k na sequência parcial $\beta = (\beta_1, \beta_2, \dots, \beta_k)$.

Para cada tarefa $j \in U$, como se fosse adicionada e se tornasse a (k+1)ésima tarefa de β, calcule o tempo de processamento $p_{v,i}$ para a tarefa artificial *v*, o tempo de partida $d_{(k+1),i}$ da tarefa j e $d_{(k+2),i}$ da tarefa artificial *v*, $for\ i = 1,2,\dots,m$. Calcule a função índice $f_{j,k}$.

Selecione a tarefa resultante do valor de $f_{j,k}$ como a (k+1)ésima tarefa de β (quebrar laços de acordo com o valor mínimo $x_{j,k}$), e remova a tarefa selecionada de U.

Endfor

A única tarefa restante de U é acrescida em β e se torna a enésima tarefa de β.

Return.

Uma solução aleatória foi criada por um algoritmo de permutação aleatória. Esta solução foi gerada para verificação da efetividade dos métodos utilizados. O algoritmo e procedimento *NEH* foi proposto por Nawaz et al. (1983), o *PF* foi proposto por McCormick (1989), o *MM* por Ronconi (2004) *wPF* e *PW* por Pan e Wang (2012). Todas as soluções geradas por esses seis algoritmos foram utilizadas como solução inicial e a melhor foi escolhida pelo método do desvio médio relativo.

O método de desvio médio relativo foi usado para ajudar nas conclusões, o método foi utilizado por Takano e Nagano (2019). O desvio relativo médio é um índice em porcentagem que indica a variação do resultado analisado em relação ao melhor resultado encontrado. Foi calculado cada desvio relativo médio de cada problema pertencente a cada classe, depois foi feita a média do desvio relativo de cada método em cada determinada classe, por exemplo, na classe 20x5 foi tirado o desvio relativo médio dos 10 problemas pertencentes a esta classe para cada método de solução, depois foi feito a média do desvio dos 10 problemas para cada problema para quantificar suas eficácias naquela classe.

Para comparação foi utilizado a base de dados Taillard (1993). Foram usadas todas as classes de problema exceto a 500 x 20, devido à falta de tempo hábil para execução. Para o tempo de setup, a base de dados proposta por Takano e Nagano (2019) foi utilizada.

RESULTADOS E DISCUSSÃO

Todas as quatro heurísticas foram utilizadas para resolver os 70 primeiros problemas do banco de dados proposto por Taillard (1993). Os problemas que foram resolvidos variam no número de tarefas e máquinas com 20, 50, 100 e 200 tarefas e 5,10 e 20 máquinas. As classes de problemas dessa base de dados são:

$$(n,m) = \begin{Bmatrix} (20{,}5);\ (20{,}10);\ (20{,}20);\ (50{,}5);\ (50{,}10);\ (50{,}20);\ (100{,}5); \\ (100{,}10);\ (100{,}20);\ (200{,}10);\ (200{,}20) \end{Bmatrix}$$

Os tempos de processamento neste banco de dados são distribuídos uniformemente entre 1 e 99. Os tempos de setup usados para os testes computacionais foram propostos por Takano e Nagano (2019) e os valores também estão uniformemente distribuídos entre 1 e 99. Todos os algoritmos heurísticos foram codificados em Python. O experimento foi realizado em um Intel® core i7 7700HQ com 2,80 GHz, 8 Gb DDR4 RAM e sistema operacional Ubuntu 18.10.

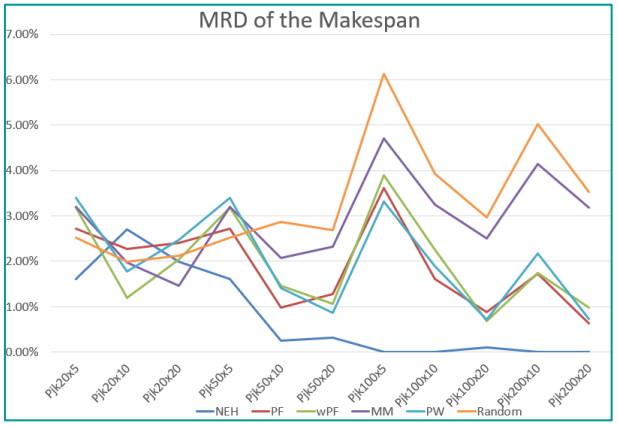
Em seguida todos os dados foram processados utilizando como critério de parada para o loop inicial do algoritmo *IG* um limite de tempo computacional (tempo_parada) *IG*ual a 30ms * m * n / 2. Em seguida os desvios médios relativos do makespan de cada classe de problema foram calculados e são apresentados na Tabela 1 e representados graficamente na Figura 1. Na Figura 2 o tempo médio computacional é mostrado. Devido ao prazo deste artigo, não foi possível executar o *NEH* 500 x 20, então todos os dados foram obtidos de problemas de tamanho 20 x 5 a 200 x 20.

Tabela 1 – Desvio relativo médio do makespan (%)

n x m	NEH	PF	wPF	MM	PW	RAND.
20 x 5	1.61	2.72	3.18	3.19	3.40	2.51
20 x 10	2.70	2.27	1.19	1.98	1.78	2.00
20 x 20	1.98	2.41	2.04	1.47	2.47	2.12
50 x 5	1.61	2.72	3.18	3.19	3.40	2.51
50 x 10	0.25	0.98	1.46	2.07	1.41	2.87
50 x 20	0.31	1.27	1.07	2.32	0.86	2.69
100 x 5	0.00	3.61	3.89	4.70	3.32	6.14
100 x 10	0.00	1.61	2.25	3.25	1.89	3.92
100 x 20	0.10	0.89	0.68	2.51	0.71	2.97
200 x 10	0.00	1.73	1.74	4.15	2.18	5.02
200 x 20	0.00	0.63	0.98	3.19	0.72	3.53
Mean	0.78	1.89	1.97	2.91	2.01	3.30

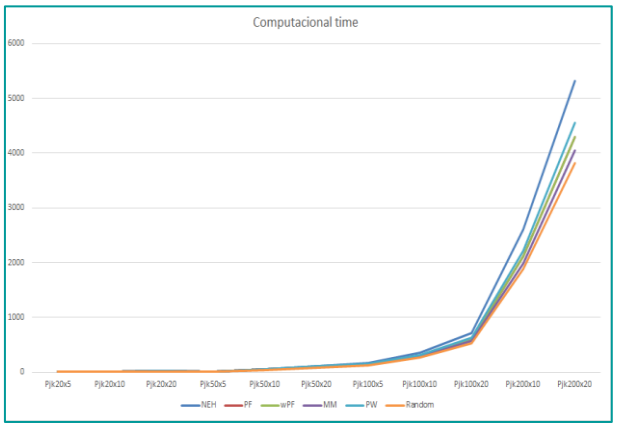
Fonte: Autoria própria (2019).

Figura 1 – Gráfico do desvio relativo médio do makespan



Fonte: Autoria própria (2019).

Figura 2 – Gráfico do tempo computacional



Fonte: Autoria própria (2019).

CONCLUSÃO

O *NEH* é um algoritmo de II fase de acordo com a classificação proposta por Framinan et al. (2004), por isso tende a dar melhores resultados que as outras heurísticas testadas (algoritmos de I fase), portanto, pode-se verificar que a qualidade dos resultados do *IG* tende a ser influenciado pela qualidade da solução inicial, mesmo que o tempo de obtenção dessa solução inicial seja maior. Todos os tempos computacionais avaliados neste trabalho não apresentam diferença efetiva entre si. Além disso, pela Figura 1, pode-se notar que a influência da solução inicial é maior para problemas maiores. A Tabela 1 mostra que o método *NEH* é 58,73% menor que *PF*, o segundo melhor método é 76,36% menor que a solução aleatória (pior método). Em relação aos demais métodos: o *NEH* é 60,40 % menor que o *wPF*, 73,19% menor que o *MM* e 61,19% menor que o *PW*.

AGRADECIMENTOS

Gostaria de agradecer a UTFPR por proporcionar a oportunidade de pesquisar e desenvolver este artigo, assim como meu orientador Mauricio Iwama Takano e coorientador Cassio Henrique dos Santos Amador por prestarem todo o apoio necessário.

REFERÊNCIAS

FRAMINAN, J. M.; GUPTA, J. N. D.; LEISTEN, R. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. **Journal of the Operational Research Society**, v. 55, n. 12, p. 1243-1255, 2004.

MCCORMICK, S. T. et al. **Sequencing in an assembly line with blocking to minimize cycle time**. Operations Research, v. 37, p. 925–936, 1989.

NAWAZ, M.; ENSCORE, E. E. J.; HAM, I. **A heuristic algorithm for the m-machine, n -job flow shop sequencing problem**. OMEGA-International Journal of Management Science, v. 11, p. 91–95, 1983.

PAN, Q.; WANG. L. **Effective heuristics for the blocking flowshop scheduling problem with makespan minimization**. ELSEVIER, v. 40, p. 218-229, 2012.

PAN, Q.; RUIZ, R. **An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem**. ELSEVIER, v. 44, p. 41-50, 2014.

RONCONI, D. P. A note on constructive heuristics for the flowshop problem with blocking. **Int Journal Production Economics**, v. 87, p. 39-48, 2004.

SANCHES, F. B.; TAKANO, M. I.; NAGANO, M. S. **Evaluation of heuristics for a branch and bound algorithm to minimize the makespan in a flowshop with blocking**. Acta Scientiarum, v. 38, n. 3, p. 321-326, 2016.

TAILLARD E., "Benchmarks for basic scheduling problems". **European Journal of Operational Research**, v. 64, n. 2, p. 278-285, 1993.

TAKANO, M. I.; NAGANO, M. S. A branch-and-bound method to minimize the makespan in a permutation flow shop with blocking and setup times. **Cogent OA**, 2017. Disponível em: https://www.cogentoa.com/article/10.1080/23311916.2017.1389638. Acesso em: 13 ago. 2017.

TAKANO, M. I.; NAGANO, M. S. **Evaluating the performance of constructive heuristics for the blocking flow shop scheduling problem with setup times.** Growing Science Ltd., v. 10, p. 37-50, 2019.