

Proposta de melhoria da heurística RAES para o problema de sequenciamento permutacional em *flow shops*

Rafael Henrique Palma Lima (UTFPR-LD) rafaelhlma@utfpr.edu.br

Resumo:

Diversos sistemas produtivos podem ser caracterizados como *flow shops*, ou seja, com todas as tarefas tendo que passar por uma mesma sequência de equipamentos. Devido à alta complexidade computacional do problema, a literatura científica tem se dedicado a propor diversas heurísticas e metaheurísticas que buscam boas soluções para o problema em um curto espaço de tempo. O objetivo deste trabalho é propor melhorias à heurística RAES (*Rapid Access with Extensive Search*), proposta na década de 1970, e testá-las utilizando uma base de instâncias de *benchmark* utilizada por outros pesquisadores na literatura. Ao todo foram propostas cinco modificações do algoritmo original obtidas a partir do uso de novas estruturas de vizinhança e estratégias de busca local. Após testar as seis versões do algoritmo RAES, ou seja, a versão original mais as cinco modificações propostas, concluiu-se que a estrutura de vizinhança mais promissora consiste na troca de todos os pares de tarefa. A adoção de estratégias de busca local do tipo “maior melhoria” ou “primeira melhoria” tiveram desempenho similar. Além disso, o algoritmo proposto é capaz de rodar usando pouco tempo computacional, mesmo nas maiores instâncias da base de *benchmark*, com *gaps* baixos com relação às melhores soluções conhecidas para as instâncias.

Palavras chave: Sequenciamento de produção, *Flow shop*, Heurísticas.

Proposal of improvements to the RAES heuristic to solve the permutation flow shop scheduling problem

Abstract

Several production systems can be characterized as flow shops, in which all jobs have to go through the same sequence of machines to be completed. Due to the high computational complexity of the problem, the scientific literature has proposed a variety of heuristics and metaheuristics that seek good solutions to the problem consuming short computational time. The objective of this work is to propose improvements to the Rapid Access with Extensive Search (RAES) heuristic, proposed in the 1970s, and to test them using a database of benchmark instances used by other researchers in the literature. In all, five modifications of the original algorithm were proposed, based on the use of new neighborhood structures and local search strategies. After testing the six versions of the RAES algorithm, i.e. the original version plus the five proposed modifications, it was possible to conclude that the most promising neighborhood structure consists in exchanging all possible pairs of jobs in a given solution. The adoption of the local search strategies "greatest improvement" and "first improvement" showed similar performance. In addition, the proposed algorithm consumes little computational time, even in the largest instances of the benchmark database, with low gaps in relation to the best-known solutions.

Key-words: Production scheduling, Flow shop, Heuristics.

1. Introdução

O sequenciamento da produção é uma atividade que ocorre no nível operacional das organizações e tem como objetivo alocar recursos às tarefas e determinar a ordem em que tais tarefas serão realizadas para cumprir prazos e quantidades de entrega. Conforme explica Guerrini, Belhot e Azzolini Jr. (2014), após definir as quantidades a serem produzidas e em quais períodos deverão estar prontas, a produção deve ser programada e deve-se decidir a alocação de cargas aos equipamentos, a sequência em que cada lote será fabricado e as datas de início e término de cada lote.

Se considerarmos que os recursos são sempre limitados, o sequenciamento da produção se torna crucial para que as empresas consigam utilizar insumos e equipamentos da maneira mais eficiente possível, visando o cumprimento de objetivos específicos. De acordo com Sule (2008), as técnicas de sequenciamento de produção são geralmente complexas para serem compreendidas por todos os colaboradores de uma organização, o que tem desmotivado diversos profissionais a usá-las em suas operações. Ainda segundo o autor, há heurísticas disponíveis de fácil entendimento e implementação capazes de encontrar boas soluções em pouco tempo, que podem ser utilizadas em diversos sistemas produtivos.

Um exemplo de sistema produtivo comumente encontrado em empresas é o *flow shop*, que é caracterizado por máquinas que são arranjadas sequencialmente. Nesse sistema, as tarefas (*jobs*) fluem sempre a partir de uma máquina inicial, passando por máquinas intermediárias até chegar à última máquina, quando a tarefa é então considerada completa. No problema clássico de sequenciamento em *flow shop* há apenas uma máquina disponível por etapa, as operações devem ser executadas sem preempção e a próxima etapa de uma tarefa só pode ser iniciada quando a anterior tiver sido finalizada (BAKER; TRIETSCH, 2009).

Diversas heurísticas têm sido propostas para minimizar o *makespan* em *flow shops*, ou seja, o tempo para que todas as tarefas tenham sido finalizadas (FERNANDEZ-VIAGAS; RUIZ; FRAMINAN, 2017). Um exemplo é o algoritmo RAES (*Rapid Access with Extensive Search*) proposta por Dannenbring (1977). À época em que esse algoritmo foi criado, os computadores não possuíam grande poder de processamento, o que limitou a complexidade computacional dos procedimentos de melhoria local utilizados pelo autor.

Por isso, o objetivo deste trabalho é propor melhorias ao algoritmo RAES original e testá-lo usando a base de instâncias de *benchmark* desenvolvida por Vallada, Ruiz e Framinan (2015), compor 240 instâncias de diversos tamanhos. Tais melhorias consistem na adoção de estruturas de vizinhança e estratégias de busca local computacionalmente mais complexas, com a finalidade de obter boas soluções para as instâncias em um curto espaço de tempo.

O restante do artigo está organizado da seguinte forma. A Seção 2 descreve o problema de sequenciamento em *flow shops* e o funcionamento do algoritmo RAES original. A Seção 3 apresenta as melhorias propostas para o algoritmo e a Seção 4 discute os resultados numéricos da aplicação das novas versões da heurística RAES. Por fim, a Seção 5 apresenta as considerações finais desta pesquisa.

2. Sequenciamento em *flow shops*

Em alguns ambientes produtivos todas as ordens de produção (OPs) devem passar por uma série de operações seguindo um mesmo roteiro de fabricação. Tais ambientes são conhecidos

como *flow shops* e as operações podem ser arranjadas sequencialmente no projeto da instalação física, pois sempre são executadas na mesma ordem em todas as OPs (PINEDO, 2016).

De forma mais detalhada, podemos considerar que o problema de sequenciamento de produção em *flow shops* (FSSP – *Flow Shop Scheduling Problem*) consiste em determinar a sequência em que n tarefas (*jobs*) deverão ser processadas em m máquinas dispostas em série. O tempo de processamento de cada tarefa $i = \{1, \dots, n\}$ em cada máquina $j = \{1, \dots, m\}$ é representado por p_{ij} . Tal valor é não negativo, determinístico e previamente conhecido (VALLADA; RUIZ; FRAMINAN, 2015).

Conforme explica Sule (2008), o problema básico de sequenciamento em *flow shops* assume que todas as tarefas estão disponíveis para processamento no tempo zero e que não há preempção durante o processamento das tarefas. Além disso, não há limitação no espaço de armazenagem entre duas máquinas sucessivas, ou seja, nenhuma máquina ficará bloqueada ao finalizar seu trabalho.

Qualquer permutação das n tarefas pode ser vista como uma solução candidata para o problema, a partir da qual é possível calcular indicadores de desempenho como o *makespan* ou o tempo total para finalizar todas as tarefas. Se a sequência de processamento das tarefas tiver que ser a mesma para todas as máquinas, o problema é chamado de Problema de Sequenciamento Permutacional em *Flow Shops*. Contudo, se for possível modificar a sequência de processamento em diferentes máquinas, o problema passa a ser classificado como não-permutacional, porém torna-se muito mais complexo devido ao aumento no tamanho do espaço de soluções possíveis (PINEDO, 2016; ROSSIT; TOHMÉ; FRUTOS, 2018).

A Tabela 1 apresenta um exemplo de instância com $n = 6$ tarefas e $m = 3$ máquinas, indicando todos os tempos de processamento p_{ij} .

Tarefa	Máquinas		
	M_1	M_2	M_3
J1	7	8	11
J2	5	12	9
J3	10	8	5
J4	14	10	13
J5	8	9	5
J6	6	9	10

Fonte: Elaborado pelo autor

Tabela 1 – Exemplo de instância com 6 tarefas e 3 máquinas

Se considerarmos o problema permutacional, uma solução arbitrária seria a sequência J_4, J_2, J_5, J_1, J_6 e J_3 . A Tabela 2 apresenta os tempos de início e término das tarefas em cada máquina, o que permite concluir que o *makespan* dessa solução seria 73, pois esse é o tempo que levaria para que todas as operações fossem finalizadas.

Máquina 1			Máquina 2			Máquina 3		
Tarefa	Início	Fim	Job	Início	Fim	Job	Início	Fim
J4	0	14	J4	14	18	J4	18	29
J2	14	19	J2	19	23	J2	29	38
J5	19	27	J5	27	35	J5	38	43
J1	27	34	J1	35	45	J1	45	58
J6	34	40	J6	45	54	J6	58	63
J3	40	50	J3	54	63	J3	63	73

Fonte: Elaborado pelo autor

Tabela 2 – Cálculo dos tempos de início e término de cada operação

O FSSP é um problema NP-Difícil, o que tem levado pesquisadores a desenvolverem heurísticas e metaheurísticas que buscam boas soluções em um curto espaço de tempo. Tradicionalmente as heurísticas podem ser classificadas como construtivas ou de melhoria. No primeiro caso o foco está na construção de uma solução, adicionando uma tarefa a cada iteração. Já o segundo tipo foca na execução de iterações que visam melhorar uma dada solução (FERNANDEZ-VIAGAS; RUIZ; FRAMINAN, 2017).

Um dos primeiros algoritmos propostos para o FSSP foi a “Regra de Johnson”, capaz de obter a solução ótima em tempo polinomial para problemas com 2 máquinas (JOHNSON, 1954). Apesar de não poder ser generalizada para problemas com mais de 3 máquinas, autores como Campbell, Dudek e Smith (1970) propuseram extensões da regra de Johnson que permitem achar boas soluções para problemas com qualquer quantidade de máquinas. Outra heurística construtiva comumente utilizada na literatura foi proposta por Palmer (1965), que se baseia em um “índice de inclinação” calculado a partir dos tempos de processamento de cada tarefa e que posteriormente é usado para obter o sequenciamento final.

De acordo com Fernandez-Viagas, Ruiz e Framinan (2017), a heurísticas construtiva mais eficiente é o algoritmo NEH, cujo nome é inspirado nas iniciais de seus criadores (NAWAZ; ENSCORE; HAM, 1983). A heurística NEH é dividida em duas fases: (i) ordenar todos os jobs de maneira decrescente com relação ao somatório de seus tempos de processamento; (ii) construção sequencial da solução processando as tarefas na mesma ordem da etapa anterior. A cada nova tarefa a ser processada, deve-se determinar a melhor posição a ser inserida na solução parcial até o momento.

As heurísticas de melhoria geralmente são compostas por um procedimento para construção de uma solução inicial seguida de uma estratégia de melhoria local. Um exemplo de algoritmo que se enquadra nessa categoria foi proposto por Dannenbring (1977) e é conhecido como RAES (*Rapid Access with Extensive Search*). A etapa de construção do algoritmo é baseada em uma versão generalizada da regra de Johnson, que reduz um problema de m máquinas a um problema de 2 máquinas. Os valores p_{i1} e p_{i2} para todas as tarefas $i = \{1, \dots, n\}$ correspondentes às máquinas fictícias $\{1, 2\}$ no problema reduzido são calculados usando as Equações (1) e (2). Vale ressaltar que o problema original possui $j = \{1, \dots, m\}$ máquinas.

$$p_{i1} = \sum_{j=1}^m (m - j + 1)p_{ij} \quad (1)$$

$$p_{i2} = \sum_{j=1}^m (j)p_{ij} \quad (2)$$

Em seguida é aplicada a regra de Johnson por meio do cálculo de um índice g_i para cada tarefa do problema reduzido utilizando a Equação (3). A solução inicial será obtida a partir da ordenação crescente dos índices g_i de cada tarefa.

$$g_i = \frac{\text{sgn}(p_{i1} - p_{i2})}{\min(p_{i1}, p_{i2})} \quad (3)$$

Após obter o sequenciamento inicial, inicia-se a fase de melhoria do algoritmo, que é baseada em uma abordagem Hill-Climbing. A partir da solução inicial, são geradas soluções vizinhas trocando todos os pares de tarefas adjacentes. Dessa maneira serão gerados n soluções vizinhas, dentre as quais a melhor de todas será selecionada como a nova solução corrente. O procedimento se repete enquanto vizinhos melhores estiverem sendo obtidos.

3. Propostas de melhoria ao algoritmo RAES

Este artigo propõe melhorias ao algoritmo RAES descrito na Seção 2. Em sua versão original, o algoritmo primeiro aplica uma heurística simples de sequenciamento e em seguida executa uma busca local com as seguintes características:

- A estrutura da vizinhança consiste na geração de todas as soluções possíveis invertendo tarefas subsequentes na solução atual;
- A escolha do vizinho que substituirá a solução atual utiliza a estratégia “maior melhoria”, ou seja, gera-se todos os vizinhos e escolhe-se o melhor dentre eles.

Uma possível crítica ao algoritmo original é a simplicidade da estrutura da vizinhança, que não permite uma boa exploração do espaço de soluções. Para um problema com n tarefas, haveria apenas n vizinhos possíveis. Os autores possivelmente usaram essa estratégia no algoritmo original devido às limitações de processamento computacional na época, uma vez que o algoritmo foi proposto na década de 1970. Porém o poder de processamento dos computadores modernos permite a adoção de estruturas de vizinhança mais complexas, que podem levar a soluções melhores para o problema sem prejudicar significativamente o tempo de execução.

Portanto, este trabalho testará o algoritmo RAES usando três estruturas de vizinhança:

- **Estrutura 1 (E1):** Consiste na estrutura de vizinhança original do algoritmo, com a troca de tarefas adjacentes na solução corrente;
- **Estrutura 2 (E2):** A vizinhança de uma solução consiste em todas as trocas possíveis de duas tarefas na solução;
- **Estrutura 3 (E3):** Consiste na implementação da estrutura 2-opt, que consiste em todas as inversões possíveis de sub-trechos da solução corrente.

Se considerarmos que qualquer solução do problema pode ser representada como um vetor contendo a sequência em que as tarefas serão processadas em todas as máquinas, então a Figura 1 ilustra o funcionamento das três estruturas de vizinhança.

Estrutura de Vizinhaça E1

Solução original:

Vizinhos

1	2	3	4	5	6	7
2	1	3	4	5	6	7
1	3	2	4	5	6	7
1	2	4	3	5	6	7
1	2	3	5	4	6	7
(...)						

Estrutura de Vizinhaça E2

Solução original:

Vizinhos

1	2	3	4	5	6	7
2	1	3	4	5	6	7
3	2	1	4	5	6	7
4	2	3	1	5	6	7
5	2	3	4	1	6	7
(...)						

Estrutura de Vizinhaça E3

Solução original:

Vizinhos

1	2	3	4	5	6	7
2	1	3	4	5	6	7
3	2	1	4	5	6	7
4	3	2	1	5	6	7
5	4	3	2	1	6	7
(...)						
1	3	2	4	5	6	7
1	4	3	2	5	6	7
1	5	4	3	2	6	7
(...)						

Figura 1 – Estruturas de vizinhaça propostas para o algoritmo RAES

Outra alteração que poderia ser testada no algoritmo é a adoção da estratégia “primeira melhoria” durante a busca local. Assim, não seria necessário gerar todos os possíveis vizinhos, pois quando é encontrado um primeiro vizinho que já representa uma melhoria, este já é retornado ao procedimento de busca local. Vale ressaltar que o caminho de busca seria, portanto, diferente do caminho resultante da estratégia “maior melhoria”.

Ao combinarmos as três estruturas de vizinhaça com as duas estratégias de busca local, temos ao todo seis versões do algoritmo que serão testadas neste artigo, conforme mostra a Tabela 3.

Algoritmo	Estrutura de Vizinhaça	Estratégia de Busca Local
V1 (Original)	E1	Maior melhoria
V2	E2	Maior melhoria
V3	E3	Maior melhoria
V4	E1	Primeira melhoria
V5	E2	Primeira melhoria
V6	E3	Primeira melhoria

Fonte: Elaborado pelo autor

Tabela 3 – Versões do algoritmo RAES a serem testadas

É importante destacar que o algoritmo V1 consiste no RAES originalmente proposto por Dannenbring (1977). As demais 5 versões são modificações propostas neste artigo e que serão testadas em instâncias de *benchmark* encontradas na literatura.

4. Resultados numéricos

Para testar o desempenho dos algoritmos propostos foram utilizadas as instâncias de *benchmark* propostas por Vallada, Ruiz e Framinan (2015). Os autores formularam 240 instâncias que variam de acordo com a quantidade de tarefas e quantidade de máquinas no *flow shop*. Foram criadas instâncias com 10, 20, 30, 40, 50 e 60 tarefas e 5, 10, 15 e 20 máquinas. Para cada combinação de quantidade de tarefas e máquinas, foram geradas 10 instâncias. Assim, há 10 instâncias de tamanho 10 x 5, ou seja, 10 tarefas e 5 máquinas, mais 10 instâncias de tamanho 10 x 10 e assim por diante. As informações contidas nas instâncias correspondem aos tempos de processamento de cada tarefa em cada máquina. Dessa maneira, as instâncias mais complexas são as de tamanho 60 x 20, ou seja, 60 tarefas e 20 máquinas. Os autores também fornecem em seu *website* uma planilha com os melhores resultados obtidos para todas as instâncias.

Os algoritmos foram implementados usando a linguagem Excel VBA e executadas utilizando o software MS Excel em um computador com processador Intel® i7-6500U com 8 GB de memória RAM e usando o sistema operacional Windows 10 64 bits.

A Tabela 4 apresenta os resultados da aplicação das 6 versões do algoritmo RAES a todas as 240 instâncias, agregando os resultados por tamanho da instância.

Tamanho	Maior Melhoria			Primeira Melhoria			Vmin
	V1	V2	V3	V4	V5	V6	
10 x 5	2,77%	0,66%	1,77%	2,83%	2,44%	2,82%	0,61%
10 x 10	3,71%	1,24%	2,55%	4,15%	1,40%	2,91%	0,99%
10 x 15	4,36%	1,90%	2,54%	5,04%	2,02%	2,93%	1,26%
10 x 20	2,06%	1,42%	2,29%	3,03%	1,79%	2,35%	1,03%
20 x 5	5,44%	1,71%	2,10%	5,32%	2,58%	2,79%	1,06%
20 x 10	7,79%	4,02%	5,73%	7,85%	4,95%	5,38%	3,54%
20 x 15	8,44%	4,48%	4,74%	8,43%	3,52%	5,60%	3,19%
20 x 20	7,93%	3,89%	4,27%	7,09%	3,27%	5,62%	2,82%
30 x 5	2,88%	1,01%	1,07%	2,89%	1,27%	1,49%	0,91%
30 x 10	9,52%	4,41%	5,78%	9,36%	4,92%	5,89%	3,67%

30 x 15	10,78%	6,48%	7,41%	9,73%	5,97%	7,76%	5,11%
30 x 20	9,93%	4,49%	6,52%	9,71%	4,93%	6,43%	4,18%
40 x 5	3,57%	1,06%	1,58%	3,49%	1,15%	1,41%	0,70%
40 x 10	10,98%	4,51%	5,79%	10,43%	4,29%	5,53%	3,53%
40 x 15	10,55%	4,95%	7,49%	10,12%	5,30%	7,96%	4,77%
40 x 20	11,11%	5,37%	6,83%	9,85%	5,03%	6,65%	4,47%
50 x 5	2,49%	0,49%	0,86%	2,58%	0,55%	1,29%	0,20%
50 x 10	8,82%	3,58%	5,55%	8,37%	3,53%	5,15%	2,83%
50 x 15	11,30%	6,26%	7,49%	10,79%	5,53%	7,16%	5,16%
50 x 20	11,64%	5,81%	7,19%	11,49%	5,85%	7,10%	4,92%
60 x 5	2,68%	0,43%	0,77%	2,61%	0,86%	1,29%	0,32%
60 x 10	9,27%	3,37%	4,76%	9,37%	3,37%	5,87%	2,70%
60 x 15	12,49%	4,73%	8,37%	12,32%	4,57%	7,38%	4,16%
60 x 20	11,75%	5,61%	7,65%	11,57%	5,45%	8,05%	4,85%

Tabela 4 – Gaps médios obtidos pelos algoritmos para cada tamanho de instância

Os valores na Tabela 4 expressam o *gap* médio entre os melhores valores conhecidos para as instâncias e os valores obtidos por cada versão do algoritmo. Vale ressaltar que os valores da Tabela 4 são agregados, ou seja, representam a média do *gap* considerando as 10 instâncias existentes para cada tamanho. Em negrito é indicada a versão do algoritmo que obteve o menor *gap* médio.

A coluna V_{min} considera como se houvesse uma sétima versão do algoritmo, em que para cada instância as 6 versões do algoritmo são executadas e a melhor solução obtida dentre as 6 é retornada. Assim, coluna V_{min} calcula o *gap* médio dessa sétima versão do algoritmo. É importante notar que como a melhor solução de cada instância pode ser obtida por algoritmos diferentes, então V_{min} não corresponderá ao menor valor dentre os *gaps* médios dos 6 algoritmos para o mesmo tamanho de instância.

Com base nos dados da Tabela 4, podemos verificar que os algoritmos que usam as estruturas de vizinhança E2 e E3 apresentam um desempenho superior às versões que usam a estrutura original E1. Isso pode ser explicado pela maior quantidade de vizinhos gerados por essas estruturas, o que possibilita uma busca mais ampla no espaço de soluções. Também é possível notar que todos os melhores desempenhos médios foram obtidos pela estrutura E2, porém em alguns casos usando a estratégia de “maior melhoria” e em outros casos com a estratégia de “primeira melhoria”. Há uma concentração maior de melhores desempenhos do algoritmo V2 para pequenas instâncias, o que parece ser um indício de que essa seria a combinação mais adequada para tais casos. Para grandes instâncias não parece haver uma diferença significativa de desempenho entre os algoritmos V2 e V5.

Outra conclusão que pode ser obtida da Tabela 4 é que a quantidade de máquinas da instância parece ser um determinante do desempenho dos algoritmos. Por exemplo, se considerarmos as instâncias com 50 e 60 tarefas, o desempenho muda significativamente entre as instâncias com 5 e 20 máquinas. Por exemplo, o *gap* médio de problemas 60 x 5 foi de 0,43% usando V2, enquanto que o mesmo algoritmo obteve um *gap* médio de 5,61% para problemas de tamanho 60 x 20. A Tabela 5 apresenta o desempenho do algoritmo V2 de acordo com a quantidade de tarefas e de máquinas. Pela tabela é possível notar que o desempenho do algoritmo em instâncias com 5 máquinas muda pouco ao aumentarmos a quantidade de tarefas. A partir de 10 máquinas o desempenho parece cair significativamente, sendo esse um indício de que a

quantidade de máquinas é um fator que aumenta a complexidade do problema de *flow shop*.

Qtd. Tarefas	Quantidade de Máquinas			
	5	10	15	20
10	0,66%	1,24%	1,90%	1,42%
20	1,71%	4,02%	4,48%	3,89%
30	1,01%	4,41%	6,48%	4,49%
40	1,06%	4,51%	4,95%	5,37%
50	0,49%	3,58%	6,26%	5,81%
60	0,43%	3,37%	4,73%	5,61%

Tabela 5 – Gaps médios do algoritmo V2 por quantidade de tarefas e máquinas

A Tabela 6 propicia uma análise do desempenho de cada algoritmo de maneira desagregada, considerando quantas vezes cada algoritmo encontrou a melhor solução de cada instância.

Tamanho	Maior Melhoria			Primeira Melhoria		
	V1	V2	V3	V4	V5	V6
10 x 5	3	8	4	4	2	5
10 x 10	0	6	2	0	6	3
10 x 15	0	8	6	0	2	3
10 x 20	3	4	2	1	3	4
20 x 5	0	7	5	0	3	3
20 x 10	1	5	2	1	2	2
20 x 15	0	2	1	0	6	1
20 x 20	0	2	1	0	7	0
30 x 5	3	7	4	3	6	4
30 x 10	0	4	2	0	2	2
30 x 15	0	3	1	0	6	0
30 x 20	0	7	0	0	3	0
40 x 5	2	6	4	2	6	4
40 x 10	0	3	1	0	4	2
40 x 15	0	6	1	0	4	0
40 x 20	0	2	1	0	7	0
50 x 5	2	6	3	2	6	2
50 x 10	0	4	0	0	5	1
50 x 15	0	3	1	0	6	1
50 x 20	0	3	0	0	5	2
60 x 5	0	5	4	0	4	2
60 x 10	0	4	1	0	4	1
60 x 15	0	4	0	0	6	0
60 x 20	0	3	0	0	7	0
Total Geral	14	112	46	13	112	42

Tabela 6 – Quantidade de instâncias em que cada algoritmo obteve a melhor solução

Para compreender os dados da Tabela 6, tome como exemplo as instâncias de tamanho 60 x 5.

Das 10 instâncias desse tamanho, 5 delas tiveram as melhores soluções encontradas por V2, 4 foram encontradas por V3, 4 por V5 e outras 2 encontradas por V6. O total é superior a 10 pois é possível ocorrer empates entre os algoritmos.

A partir dos resultados da Tabela 6 podemos notar o desempenho superior dos algoritmos V2 e V5, com cada um encontrando 112 melhores soluções. Isso é um forte indício de que a estrutura de vizinhança E2 é a mais adequada para esse tipo de problema. Porém, não há diferença significativa entre usar as duas estratégias de busca local. Para algumas instâncias a melhor estratégia é a “maior melhoria” e para outras instâncias a estratégia “primeira melhoria” resulta em melhores soluções.

Com relação aos algoritmos V3 e V6 que utilizam a estrutura de vizinhança E3 baseada no método 2-opt, deve-se destacar que tal abordagem conseguiu ser a melhor para várias instâncias. Apesar de ter obtido um desempenho médio inferior à estrutura E2, os resultados indicam que tal estrutura pode ser promissora em certos casos, não devendo portanto ser descartada para o estudo de problemas de *flow shop*.

Para finalizar a análise dos algoritmos, a Tabela 7 apresenta os tempos médios de execução dos algoritmos em segundos, com quatro casas decimais de precisão.

Tamanho	Maior Melhoria			Primeira Melhoria		
	V1	V2	V3	V4	V5	V6
10 x 5	0,0004	0,0074	0,0047	0,0008	0,0031	0,0047
10 x 10	0,0016	0,0074	0,0094	0,0020	0,0109	0,0063
10 x 15	0,0020	0,0113	0,0094	0,0016	0,0125	0,0172
10 x 20	0,0047	0,0180	0,0156	0,0051	0,0137	0,0207
20 x 5	0,0063	0,0465	0,0531	0,0027	0,0438	0,0754
20 x 10	0,0227	0,1238	0,1063	0,0109	0,0977	0,1113
20 x 15	0,0164	0,1629	0,1813	0,0172	0,1809	0,2160
20 x 20	0,0250	0,2465	0,2656	0,0328	0,2637	0,3148
30 x 5	0,0090	0,1379	0,1453	0,0105	0,0992	0,1426
30 x 10	0,0332	0,6043	0,4844	0,0375	0,5078	0,7492
30 x 15	0,0594	0,8863	0,7371	0,0613	0,7027	1,0996
30 x 20	0,0918	1,3605	1,0375	0,1000	1,2750	1,6828
40 x 5	0,0188	0,3305	0,4234	0,0188	0,2629	0,4215
40 x 10	0,0641	1,4598	1,5113	0,0656	1,3406	2,2766
40 x 15	0,1391	3,2855	2,1570	0,1328	2,4863	2,9457
40 x 20	0,1797	3,7230	3,5539	0,1953	4,3609	6,5156
50 x 5	0,0488	0,7543	0,6656	0,0414	0,6887	0,9188
50 x 10	0,0980	3,1250	2,7973	0,1031	2,7863	4,1223
50 x 15	0,3441	7,3207	6,8434	0,2598	6,5109	9,9145
50 x 20	0,3754	9,2168	8,4223	0,3191	7,4625	13,9773
60 x 5	0,0586	1,7047	1,5172	0,0484	0,9805	1,4727
60 x 10	0,2387	7,3035	7,2871	0,1629	5,5113	6,5332
60 x 15	0,4098	16,7273	9,4953	0,3258	13,2574	16,9398
60 x 20	0,5258	18,5711	16,2223	0,4277	18,0535	22,0066

Tabela 7 – Tempos médios de execução dos algoritmos (em segundos)

A Tabela 7 mostra que todos os algoritmos obtêm soluções em menos de um segundo para problemas com tamanho até 30 x 15. Após esse tamanho, pode-se notar um aumento mais acentuado nos tempos de execução dos algoritmos que utilizam as estruturas de vizinhança E2 e E3. Apesar disso, mesmo para as maiores instâncias, o tempo médio para obter a solução foi de apenas 22 segundos.

5. Considerações finais

Este artigo analisou a heurística RAES para o problema de sequenciamento em *flow shops* com soluções permutacionais e propôs cinco novas versões para o algoritmo baseadas em novas estruturas de vizinhança e estratégias de busca local. As propostas foram testadas usando 240 instâncias de *benchmark* e comparadas com os melhores valores conhecidos. Os resultados obtidos pelos algoritmos propostos foram superiores à heurística RAES original, com baixos *gaps* com relação às instâncias analisadas.

Uma descoberta que merece destaque é que a estrutura de vizinhança E2, que consiste na troca de todos os pares possíveis de tarefas em uma solução, foi capaz de obter os melhores resultados, o que parece ser um indício de que essa é a estrutura de vizinhança mais promissora para este problema. A respeito das estratégias de busca local testadas, não houve diferenças significativas de desempenho entre usar a “maior melhoria” ou a “primeira melhoria”, independentemente da estrutura de vizinhança adotada.

As alterações propostas consomem pouco tempo para serem executadas. Instâncias com até 30 tarefas foram executadas em até 1 segundo, ao passo que instâncias maiores com 60 tarefas e 20 máquinas necessitaram de apenas 22 segundos para obter a solução final.

Vale ressaltar que tanto o algoritmo RAES original como as alterações propostas ainda caracterizam a heurística como sendo determinística, sem nenhuma aleatorização na busca por soluções. Da forma como os algoritmos foram propostos, as soluções obtidas sempre serão as mesmas se usarmos uma mesma instância de entrada. Por isso, uma oportunidade para pesquisas futuras é a utilização das estruturas de vizinhança e estratégias de busca local usando metaheurísticas como a ILS (*Iterated Local Search*), VNS (*Variable Neighborhood Search*) ou algoritmos genéticos.

Referências

- BAKER, K.; TRIETSCH, D. *Principles of sequencing and scheduling*. John Wiley & Sons, 2009.
- CAMPBELL, H.; DUDEK, R.; SMITH, M. A heuristic algorithm for the *n* job *m* machine sequencing problem. *Management Science*, Vol. 16, p. 630-637, 1970.
- DANNENBRING, D. An Evaluation of Flow Shop Sequencing Heuristics. *Management Science*. Vol. 23, p. 1174-1182, 1977.
- GUERRINI, F.; BELHOT, R.; AZZOLINI JR, W. *Planejamento e controle da produção: projeto e operação de sistemas*. Rio de Janeiro: Elsevier, 2014.
- NAWAZ, M.; ENSCORE, E.; HAM, I. A heuristic algorithm for the *m*-machine *n*-job flow-shop sequencing problem. *Omega*. Vol. 11, p. 91-95, 1983.
- PALMER, D. Sequencing jobs through a multi-stage process in the minimal total time: a quick method of obtaining a near optimum. *Operational Research Quarterly*. Vol. 16, p. 101-107, 1965.



VIII CONGRESSO BRASILEIRO DE ENGENHARIA DE PRODUÇÃO

Ponta Grossa, PR, Brasil, 05 a 07 de dezembro de 2018

PINEDO, M. *Scheduling: theory, algorithms and Systems*. 5^a ed. New York: Springer, 2016.

ROSSIT, D.; TOHMÉ, F.; FRUTOS, M. *The Non-Permutation Flow-Shop scheduling problem: A literature review*. Omega. Vol. 77, p. 143-153, 2018.

SULE, D. *Production planning and industrial scheduling: examples, case studies and applications*. 2^a ed. CRC Press, 2008.

VALLADA, E.; RUIZ, R.; FRAMINAN, J. *New hard benchmark for flowshop scheduling problems minimizing makespan*. European Journal of Operational Research. Vol. 240, p. 666-677, 2015.

FERNANDEZ-VIAGAS, V.; RUIZ, R.; FRAMINAN, J. *A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation*. European Journal of Operational Research. Vol. 257, p. 707-721, 2017.