EXPERIMENTAÇÃO COMPUTACIONAL DE MODELOS MATEMÁTICOS PARA O PROBLEMA DE *FLOW SHOP* COM TEMPOS DE *SETUP*

Lorrany Guilherme Santos
Discente do Curso de engenharia de produção da Universidade Federal de Goiás;
R. Mucuri, s/n, Área 3, Setor Conde dos Arcos, CEP 74.968-755, Aparecida de Goiânia/GO
lorrany.engprod@gmail.com

Hélio Yochihiro Fuchigami
Docente do Curso de engenharia de produção da Universidade Federal de Goiás;
R. Mucuri, s/n, Área 3, Setor Conde dos Arcos, CEP 74.968-755, Aparecida de Goiânia/GO
heliofuchigami@ufg.br

RESUMO

Este trabalho considera o problema de *flow shop* permutacional com tempos de *setup* independentes da sequência e minimização do tempo médio de fluxo. Com o intuito de comparar os resultados obtidos, propõe-se dois modelos de programação linear inteira mista.: um baseado em variáveis de precedência e outro em variáveis de posição. A experimentação computacional foi realizada por meio de um conjunto de 1000 problemas-testes da literatura, divididos em 40 classes, segundo a variação do número de máquinas, tarefas e distribuição dos tempos de processamento. Os resultados evidenciaram que a formulação baseada em variáveis de posição é mais eficiente do que aquela baseada em variáveis de precedência.

PALAVRAS CHAVE: Sequenciamento da produção, *Flow shop, Setup* independente da sequência

ABSTRACT

This paper addresses the permutation flow shop problem with sequence-independent setup times and minimization of mean flow time. In order to compare the results, two mixed integer linear programming models were proposed: one based on precedence variables and another on position variables. The computational experimentation was performed through a set of 1000 instances of the literature, divided into 40 classes, according to the variation of the number of machines, jobs and distribution of processing times. The results highlighted that the formulation based on position variables is more efficient than that based on precedence variables.

KEYWORDS: *Scheduling, Flow shop,* sequence-independent *setup*

https://proceedings.science/p/85453?lang=pt-br

1. Introdução

O problema de sequenciamento da produção consiste na definição de uma ordem sobre um conjunto de trabalhos e máquinas. Na programação da produção, existem inúmeras possibilidades de padrões a serem encontrados, dependendo do ambiente existente, do número de operações requisitadas para o processamento da tarefa e do número de máquinas disponíveis para cada operação [ZADIEH et al., 2006]. Para encontrar a melhor solução para as diversas medidas de desempenho propostas, vários métodos têm sido desenvolvidos., incluindo modelos matemáticos, algoritmos exatos, heurísticas e meta-heurísticas.

Dentre tantas possibilidades, essa pesquisa aborda o problema com o ambiente de programação da produção *flow shop* permutacional, um campo de pesquisa muito ativo que consiste em agendar todos os trabalhos em todas máquinas sempre na mesma ordem, sendo que existe apenas uma máquina responsável para cada operação ou estágio [PINEDO, 2016].

Muitas pesquisas em programação da produção desconsideram algumas restrições como forma de encontrar um problema aproximado aos que existem nas indústrias e no cotidiano. Um tipo de restrição comumente retirada ou incluída no tempo de processamento das tarefas são os tempos de *setup*, o tempo de *setup* inclui todo trabalho de preparação da máquina ou da oficina para a fabricação de produtos, por exemplo, a obtenção e ajuste de ferramentas, inspeção e posicionamento de materiais e processos de limpeza. [PINEDO, 2016].

Muitos problemas realistas, no entanto, exigem que o *setup* seja separado dos tempos de processamento, como por exemplo, um trabalho de preparação de uma máquina pode ser feito antes da conclusão da operação de trabalho na máquina anterior, caso haja algum tempo ocioso na máquina seguinte. Em tal situação, denominada de *setup* antecipado não é válido absorver o tempo de *setup* no tempo de processamento. Além disso, separar o tempo de *setup* pode indicar uma minimização no tempo total [SILVA, 2012]. Por esse motivo, com o objetivo de encontrar uma solução mais próxima do real, a restrição de *setup* antecipado foi considerada neste trabalho.

Em relação às medidas de desempenho, as mais comuns na literatura científica 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 [FUCHIGAMI, 2015].

A função objetivo que neste trabalho queremos minimizar será o tempo médio de fluxo, com *setup* independente da sequência, podendo ser antecipado. Portanto, na clássica notação de três campos, este problema pode ser denotado por $Fm|prmu,s_j|\ \bar{F}$, onde "F_m" indica o ambiente *flow shop* genérico com "*m*" máquinas, "*prmu*" é a restrição permutacional, "s_j" enfatiza a existência de *setup* para as tarefas e "$\bar{F}$" representa a função objetivo descrita.

Para alcançar os resultados para o problema proposto, existem dois tipos de soluções possíveis: uma técnica de programação linear com solução exata, que tem a grande vantagem de resolver a maioria dos casos, no entanto exige um elevado tempo de computação e memória para realizar os cálculos. Por outro lado, os algoritmos heurísticos, que não fornecem necessariamente a solução ótima para o problema, mas exige memória e tempo computacional reduzido. Portanto, o grande desafio dessa pesquisa, que optou por desenvolver propostas baseadas em solução exata consiste em encontrar um modelo que não apenas resolva o problema, mas também consiga solucionar o

problema de forma eficiente. Sendo para isso necessário estar atento a influência das variáveis de decisão, do número de restrições e dos parâmetros envolvidos.

O trabalho está estruturado da seguinte forma: a seção 2 apresenta o referencial teórico da pesquisa, na seção 3, definem-se os modelos propostos, os parâmetros envolvidos e o tamanho dos modelos em questão. Na seção 4 definem-se os elementos amostrais, bem como os dados para implementação, o software e o resolvedor utilizados e os resultados obtidos. Por fim, a quinta seção apresenta a conclusão do trabalho.

2. Referencial teórico

A programação da produção ou *scheduling* surgiu como uma área de pesquisa na década de 1950 com a publicação do trabalho de [JOHNSON, 1954]. Desde então, vem recebendo considerável atenção de estudiosos da pesquisa operacional, matemáticos e pesquisadores da área de produção e operações.

A classificação dos sistemas de produção, quanto ao posicionamento do processo de produção, corresponde ao padrão do fluxo dos produtos nas máquinas. Neste contexto, Boiko [2013] apresenta uma classificação dos Sistemas de Produção quanto ao posicionamento do processo, conforme segue:

Dentre os problemas de programação existentes Boiko [2013] afirma que suas possíveis variações estão relacionadas com o posicionamento do processo de produção, o fluxo padrão dos produtos nas máquinas e as restrições tecnológicas do ambiente de produção.

Os ambientes de produção podem ser divididos em duas vertentes, a primeira constituída de uma única etapa, onde há somente uma operação para cada tarefa; a segunda se baseia em várias etapas, nos quais uma ordem de produção é composta por várias operações em diferentes máquinas [PINEDO, 2016]. Na primeira configuração, apenas uma máquina executa todas as tarefas, ou podem existir várias máquinas com a mesma funcionalidade rodando operações em paralelo. Já no ambiente de múltiplas etapas, cada tarefa deve ser executada em máquinas distintas, com diferentes características. Esse último grupo é subdividido em *job shop, open shop e flow shop*. Em um *job shop* cada ordem é única, com rotas pré-estabelecidas e diferentes entre si. No *open shop*, um pouco mais complexo, a rotas das máquinas por qual as ordens passam pode não ser a mesma para todas as ordens. Por fim, em um *flow shop*, ilustrado na Figura 1, todas as ordens possuem as mesmas rotas, o que significa que as operações em cada tarefa são realizadas em uma mesma sequência nas máquinas, para todas as ordens de produção.

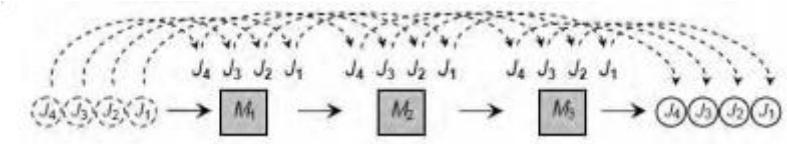


Figura 1: ilustração de um *flow shop* permutacional

Qualquer que seja o ambiente de produção, a programação visa atingir um objetivo ou critério específico. Para medir a eficácia de uma decisão utiliza -se as medidas de desempenho, que também são responsáveis por caracterizar a natureza do problema de programação [DAVIS et al., 2001].

A medida de desempenho adotado neste trabalho é o tempo médio de fluxo ou *flow time* uma vez que, como declaram Gaither e Frazier [2002], entre os diversos desafios inerentes a programação da produção, o mais comum está relacionado com a redução do tempo ocioso e máxima utilização dos recursos.

Um dos meios para alcançar essa proposta está na maneira como é tratado o *setup*. Segundo Allahverdi et al.[1999], este termo corresponde ao trabalho de preparar uma máquina ou um

processo para processar uma tarefa. Isto inclui obter ferramentas, limpeza, recolocação de ferramental, posicionamento de acessórios, ajuste de ferramentas e inspeção de materiais. Para classificar o tipo de *setup* podemos analisar critérios como à dependência da sequência adotada e quanto à "antecipabilidade".

Quanto à dependência da sequência, O tempo de *setup* pode ser definido como independente ou dependente da sequência. No primeiro caso, o *setup* depende somente da tarefa a ser processada e é chamado independente da sequência. E no segundo caso, o *setup* depende tanto da tarefa a ser processada como também da que foi executada imediatamente antes na mesma máquina, sendo chamado dependente da sequência [ALLAHVERDI et al., 1999]. Além disso, existem os problemas dependentes das máquinas, para o caso de um ambiente híbrido, em que mais de uma máquina realiza a mesma operação [JUNGWATTANAKIT et al., 2009].

Os *setups* independentes da sequência de execução das tarefas são ainda subdivididos em dois tipos: antecipado e não antecipado.

Segundo a "antecipabilidade", o setup é classificado, segundo Allahverdi et al. [2008], em:

- *Setup* antecipado significa que o *setup* requisitado por uma tarefa pode se iniciar antes mesmo que da tarefa em questão estar disponível para ser processada, como é possível visualizar na Figura 2. Geralmente, o *setup* somente pode ser antecipado completamente quando a máquina está ociosa;

- *Setup* não antecipado neste caso, o *setup* solicitado por uma tarefa, ocorre apenas a tarefa tornar-se disponível para ser processada, como ilustra a Figura 3.

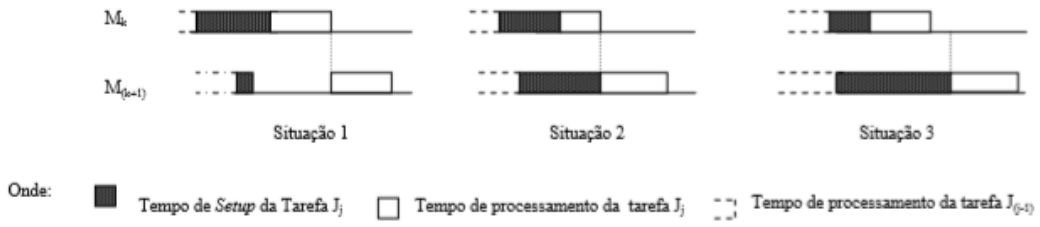


Figura 2. Programações para uma determinada tarefa J_j com tempos de *setup* antecipado

A Figura 2 representa as três possíveis situações de programação para uma determinada tarefa J_j, quando o *setup* é antecipado. Na primeira situação, o *setup* da tarefa J_j foi concluído na máquina $M_{(k+1)}$, antes mesmo da tarefa J_j estar liberada para o processamento. Na situação 2 o *setup* e liberação da tarefa ocorre no mesmo momento. E por fim, a situação 3 ocorre quando a tarefa J_j fica disponível para ser processada, o *setup* ainda não foi completado.

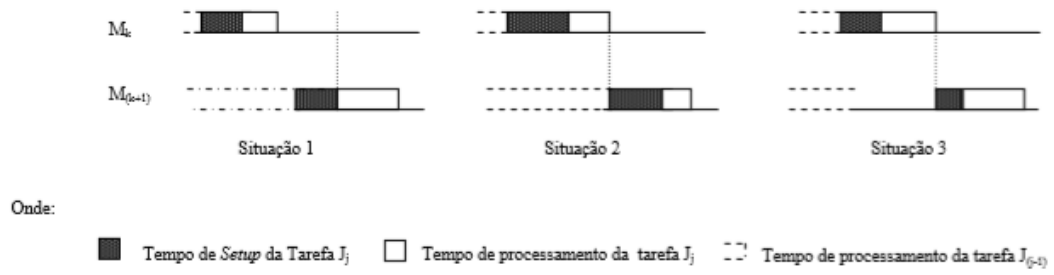


Figura 3 Programações para uma determinada tarefa J_j com tempos de *setup* não antecipado

No caso que o *setup* é não antecipado as situações podem ser explicadas da seguinte maneira, primeiro quando a tarefa J_j fica disponível para ser processada, no entanto deve aguardar a liberação da máquina pela tarefa anterior. Depois, quando a tarefa J_j fica disponível para ser processada a máquina também fica disponível. Finalmente, quando a máquina $M_{(k+1)}$ já está disponível, no entanto a tarefa ainda não foi liberada da máquina anterior.

No entanto, como afirmam Cheng et al. [2000], não fará sentido incluir no problema os tempos de *setups*, com o intuito de reduzir o tempo ocioso das máquinas, caso estes não puderem ser realizados antecipadamente para uma tarefa. Já que, neste caso, o problema de setups independentes da sequência é automaticamente simplificado considerando-se o *setup* como parte do processamento, ou seja, os tempos de *setup* são considerados como parte dos tempos de processamento, ignorando-se os tempos de *setup*. Desta forma, o problema passaria a ser formulado e resolvido com um problema de *flow shop* tradicional. Como para o objetivo do trabalho é extrema importância que seja considerado o *setup* e que o mesmo não contra para o aumento da ociosidade nas máquinas é coerente considerar o *setup* antecipado. E, portanto, os modelos devem sempre em consideração, as 3 possíveis situações ilustradas na Figura 2.

3. Modelos matemáticos propostos

O problema tratado pode ser definido genericamente da seguinte forma. Considere um conjunto de *n* tarefas $J = \{J_1, J_2, \ldots, J_n\}$, 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, $M = \{M_1, M_2, \ldots, M_n\}$, 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 um tempo de setup s_{jk} independente também em cada máquina M_k, ambos considerados conhecidos previamente e fixos. A solução consiste na elaboração de uma programação que minimize o tempo médio de fluxo. Para a solução do problema foram propostos dois modelos matemáticos: um baseado em variáveis de precedência e outro em variáveis de posição. Os parâmetros, as variáveis e os modelos propostos, considerando que existem *n* tarefas para serem processadas em m máquinas, são apresentados a seguir.

3.1 Modelo baseado em variáveis de precedência

- **Parâmetros:**

i, j: índices das tarefas	$i = 0, \ldots, n,$	$j = 0, \ldots, n$
k: índice das máquinas		$k = 1, \ldots, m$
p_{jk} : tempo de processamento da tarefa Jj na máquina M_k	$j = 1, ..., n,$	$k = 1, ..., m$
s_{jk}: tempo de *setup* da tarefa Jj na máquina M_k	$j = 1, ..., n,$	$k = 1, ..., m$
B: valor muito grande, considerado $10 \cdot \sum_k \sum_j p_{jk}$		

- **Variáveis de decisão:**

$x_{ij}\begin{cases}1\ se\ a\ tarefa\ J_i\ precede\ a\ tarefa\ J_j\\0, caso\ contrario\end{cases}$

C_{jk} – Término da tarefa *j* na máquina *k*

- **Modelo matemático:**

Min Z = $\frac{\sum_j C_{jm}}{n}$ (1)

$C_{jk} - s_{jk} \geq p_{jk}$ *j*= 1, ... , *n* *k*= 1, ... , *m* (2)

$C_{jk} - C_{j(k-1)} \geq p_{jk}$ *j*= 1, ... , *n* *k*= 2, ... , *m* (3)

$C_{jk} - C_{ik} + B(1 - x_{ij}) - s_{jk} \geq p_{jk}$ *i*= 1, ... , *n* *j*= 1, ... , *n* *k*= 1, ... , *m* *i≠j* (4)

$C_{ik} - C_{jk} + B \cdot x_{ij} - s_{ik} \geq p_{ik}$ *i*= 1, ... , *n* *j*= 1, ... , *n* *k*= 1, ... , *m* *i≠j* (5)

$x_{ij} = 0$ *i*= 1, ... , n *i=j* (6)

$x_{ij} \in \{0,1\}$ *i*= 1, ... , *n* *j*= 1, ... , *n* (7)

$C_{jk} \geq 0$ *j*= 1, ... , *n* *k*= 1, ... ,*m* (8)

A função objetivo minimiza o fluxo médio das tarefas. As expressões (2) asseguram que o término de uma tarefa em qualquer máquina seja no mínimo igual ao tempo de processamento somado ao tempo de *setup* para aquela tarefa. As restrições (3) asseguram que o início da tarefa em uma máquina deve aguardar até que a mesma tarefa seja liberada da máquina anterior e portanto, o término da *k-ésima* operação da tarefa J_j seja completada após a operação (*k*-1) mais o tempo de processamento (p_{jk}). As restrições (4) e (5) asseguram a consistência dos instantes de término das tarefas, ou seja, se a tarefa J_i precede imediatamente a tarefa J_j, ou seja, x_{ij}= 1, a diferença entre seus instantes de término em cada máquina deve ser pelo menos igual ao tempo de processamento somado ao tempo de *setup* da tarefa $J_j$na máquina considerada, caso contrário, se x_{ij} = 0, não há relação entre os instantes de término deste par de tarefas e as restrições. As restrições (6) garantem que uma mesma tarefa não pode ser precedida por ela mesma. As expressões (7) e (8) definem o domínio das variáveis do modelo.

O tamanho problema em relação ao modelo 1, em termos do número e tipo das variáveis e das restrições, é apresentado na Tabela 1.

Tabela 1: Tamanho do problema em relação ao modelo 1 em termos do número de variáveis e de restrições

Variáveis	Binárias	n^2
	Inteiras	nm
	Total	**n(n+m)**
Restrições	(2)	nm
	(3)	nm
	(4)	nm(n-1)
	(5)	nm
	(6)	n
	Total	**n^2m+2nm+n**

O modelo proposto possui n^2 variáveis binárias, nm variáveis inteiras e n^2m+2nm+n restrições. Assim, uma base de dados com n = 15 tarefas e m = 3 máquinas terá 270 variáveis e 780 restrições; já uma instância com n = 30 tarefas e m = 6 máquinas terá 1080 variáveis e 5790 restrições.

3.2 Modelo baseado em variáveis de posição

- **Parâmetros:**

i, j: índices das tarefas $i = 1, \ldots, n,$ $j = 1, \ldots, n$
k: índice das máquinas $k = 1, \ldots, m$
p_{jk} : tempo de processamento da tarefa J_j na máquina M_k $j = 1, ..., n,$ $k = 1, ..., m$
s_{jk}: tempo de *setup* da tarefa J_j na máquina M_k $j = 1, ..., n,$ $k = 1, ..., m$

- **Variáveis de decisão:**

$$x_{jh} \begin{cases} 1 \; se \; a \; tarefa \; j \; é \; programada \; na \; posição \; h \\ 0, caso \; contrario \end{cases}$$

C_{hk} – Término da tarefa na posição h na máquina k

- **Modelo matemático:**

$$\text{Min Z} = \frac{\sum_h C_{hm}}{n} \quad (9)$$

$$\sum_j x_{jh} = 1 \qquad h=1,\ldots, n \quad (10)$$

$$\sum_h x_{jh} = 1 \qquad j=1, \ldots, n \quad (11)$$

$$\sum_j (p_{j1} + s_{j1}) \cdot x_{j1} = C_{11} \quad (12)$$

$$C_{(h-1)1} + \sum_j (p_{j1} + s_{j1}) \cdot x_{jh} = C_{h1} \qquad h=2, \ldots, n \quad (13)$$

$$\sum_j (p_{jk} + s_{j1}) \cdot x_{j1} \leq C_{1k} \qquad k=2, \ldots, m \quad (14)$$

$$C_{h(k-1)} + \sum_j p_{jk} \cdot x_{jh} \leq C_{hk} \qquad h=1, \ldots, n \quad k=2, \ldots, m \quad (15)$$

$$C_{(h-1)k} + \sum_j (p_{jk} + s_{jk}) \cdot x_{jh} \leq C_{hk} \qquad h=2, \ldots, n \quad k=2, \ldots, m \quad (16)$$

$$x_{jh} \in \{0,1\} \qquad j= 1, \ldots, n \quad h=1, \ldots, n \quad (17)$$

$$C_{hk} \geq 0 \qquad h=1, \ldots, n \quad k=1, \ldots, m \quad (18)$$

A função objetivo minimiza o fluxo médio das tarefas. As expressões (10) e (11) são restrições clássicas de associação (problema da designação) e asseguram que apenas uma tarefa pode ser programada em cada posição. A restrição (12) determina que o término da tarefa programada na primeira posição na primeira máquina é igual à soma do tempo de processamento com o tempo de

setup dessa mesma tarefa na primeira máquina. As restrições (13) garantem que o término de cada tarefa na primeira máquina é sempre maior ou igual ao término da tarefa anterior somado ao tempo de processamento com o tempo de *setup* dessa mesma tarefa na primeira máquina. As expressões (14) garantem que o término da tarefa programada na primeira posição da segunda máquina em diante é maior ou igual à soma do tempo de processamento com o tempo de *setup* dessa mesma tarefa na máquina atual (*k*). As restrições (15) garantem que o término da qualquer tarefa programada em qualquer posição e da segunda máquina em diante é maior ou igual ao término da tarefa programada na máquina anterior (*k*-1) somado ao tempo de processamento dessa mesma tarefa na máquina atual (*k*). As restrições (16) asseguram que o término da qualquer tarefa programada da segunda posição em diante e da segunda máquina em diante ou maior ou igual ao término da tarefa programada na posição anterior (*h*-1) na máquina atual (*k*) somado ao tempo de processamento com o tempo de *setup* dessa mesma tarefa na máquina atual (*k*).

As expressões (17) e (18) definem o domínio das variáveis do modelo.

O tamanho do problema em relação ao modelo 2, em termos do número e tipo das variáveis e das restrições, é apresentado na Tabela 2.

Tabela 2: Tamanho do problema em relação ao modelo 2 em termos do número de variáveis e de restrições

Variáveis	Binárias	n^2
	Inteiras	nm
	Total	**n(n+m)**
Restrições	(2)	n
	(3)	n
	(4)	1
	(5)	n
	(6)	m
	(7)	n(m-1)
	(8)	(n-1)(m-1)
	Total	**2nm+n+2**

O modelo proposto possui n^2 variáveis binárias, nm variáveis inteiras e 2nm+n+2 restrições. Assim, uma base de dados com n = 15 tarefas e m = 3 máquinas terá 270 variáveis e 107 restrições; já uma instância com n = 30 tarefas e m = 6 máquinas terá 1080 variáveis e 392 restrições.

4 Experimentação computacional e resultados

4.1 Definição da amostragem

Para efeito de análise e comparação de desempenho dos dois modelos propostos, avaliou-se na experimentação computacional 1000 problemas-testes de Ruiz e Allahverdi [2007].

Os problemas foram divididos em classes, sendo cada uma formada pela combinação do número de tarefas, número de máquinas e o intervalo de distribuição dos tempos de processamento e de *setup*. Assim, 4 (alternativas de número de tarefas) x 5 (alternativas de número de máquinas) x 1 (alternativa de intervalo de *setup*) x 2 (alternativa de intervalo de processamento) = 40 classes.

Os valores para o número de tarefas e de máquina, respectivamente, foram: n = {15,20,25,30} e m = {2, 3, 4, 5, 6}. Os tempos de processamento das tarefas foram gerados no intervalo uniforme U[1,10] e U[1,100] e o tempo de *setup* em um intervalo uniforme U[0,50]. . Cada classe é constituída por 25 replicações, ou seja, 25 problemas-testes gerados através da distribuição

uniforme em questão, totalizando os 1000 problemas já citados. A Tabela 3 apresenta o detalhamento dos problemas-testes resolvidos.

Tabela 3: Parâmetros da experimentação computacional e número de problemas resolvidos por cada modelo

Parâmetros	**Valores**	
Número de tarefas (n)	15,20,25,30	15,20,25,30
Número de máquinas (m)	2,3,4,5,6	2,3,4,5,6
Faixa de distribuição dos tempos de processamento	[0;100]	[0;10]
Faixa de distribuição dos tempos de *setup*	[0;50]	[0;50]
Número de replicações para cada classe	25	25
Número de problemas-testes	500	500
Total de problemas-testes resolvidos	**1000**	

4.2 Implementação computacional

Os modelos foram implementados em "JuMP", uma linguagem de modelagem embutida na linguagem de programação "JULIA", escolhida para este trabalho por ter um código fácil, moderno, com IDE gratuito e eficiente para problemas de programação linear inteira mista. Por meio desta linguagem de modelagem, é possível realizar a execução sequencial de vários exemplares; além disso, a JuMP já inclui vários resolvedores também gratuitos. O resolvedor utilizado foi o Cbc e todos os testes foram conduzidos em um processador Intel®Core™i5 com 2.3 GHz, 6.0 GB de memória RAM e sistema operacional Windows. Os 1000 problemas-testes foram resolvidos até um tempo máximo de 600 segundos pelo algoritmo *branch-and-cut* incluído no Cbc. As medidas analisadas para cada um dos modelos foram o tempo médio de computação (CPU) em segundos e o *gap* de otimalidade.

4.3 Resultados

A Tabela 4 a seguir apresenta os resultados numéricos da experimentação computacional com os dois modelos propostos. Sendo o modelo 1 baseado em variáveis de precedência e o modelo 2 baseado em variáveis de posição.

Tabela 4: Comparativo da eficiência computacional média por porte do problema

		MODELO 1		MODELO 2	
n	**m**	**CPU (s)**	**GAP**	**CPU (s)**	**GAP**
15	2	600.44	0.54	5.53	0.00
	3	600.26	0.52	19.77	0.00
	4	600.25	0.50	36.50	0.00
	5	600.42	0.49	110.57	0.00
	6	600.34	0.49	213.62	0.00
Média		**600.34**	**0.51**	**77.20**	**0.00**
20	2	600.52	0.73	13.27	0.00
	3	600.62	0.72	470.69	0.00
	4	600.67	0.70	547.75	0.01
	5	600.62	0.67	535.85	0.01
	6	600.81	0.65	576.04	0.02
Média		**600.65**	**0.69**	**428.72**	**0.01**
25	2	600.46	0.81	327.17	0.01
	3	600.70	0.78	479.39	0.01
	4	600.70	0.75	568.16	0.02
	5	601.03	0.73	600.29	0.03
	6	601.35	0.70	600.17	0.04
Média		**600.85**	**0.75**	**515.03**	**0.02**
30	2	600.53	0.84	478.39	0.01
	3	600.93	0.81	564.94	0.01
	4	601.17	0.79	600.31	0.02
	5	601.95	0.76	600.12	0.03
	6	602.52	0.75	600.08	0.05
Média		**601.42**	**0.79**	**568.77**	**0.03**

Conforme os dados da Tabela 4, no modelo 1 o tempo médio de CPU para todos os problemas esteve em torno de 600 segundos, ou seja, todos os problemas alcançaram o limite de tempo estipulado. Já para o modelo 2, a maioria dos problemas-testes foram resolvidos na otimalidade. O tempo médio de CPU para 15, 20, 25 e 30 tarefas respectivamente foi de 77,2 segundos; 428,72 segundos (7,13 minutos); 515,03 segundos (8,58 minutos); e 568,77 (9,48 minutos). No modelo 2, apenas os problemas maiores, com 25 e 30 tarefas, atingiram o limite de tempo de CPU.

O Gráfico 2 ilustra a evolução do tempo de CPU, em segundos, com o aumento do tamanho do problema em cada um dos modelos.

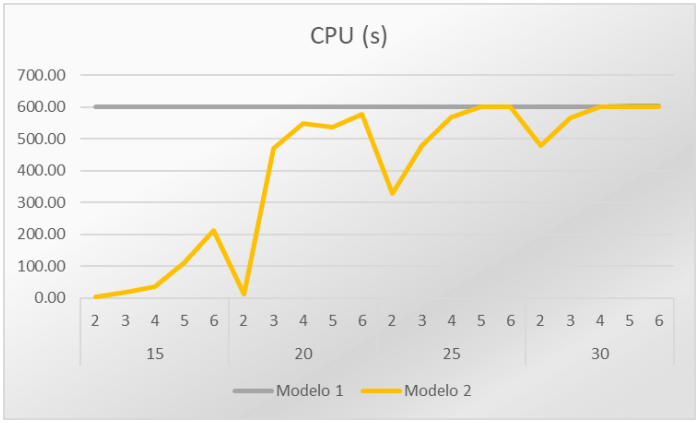


Grafico 1: Tempo médio computacional

Por meio do Gráfico 2, nota-se que o tempo médio de CPU para o modelo 1 é constante em 600 segundos e a curva ilustrada para o modelo 2 apresenta um comportamento bem instável, com tendência de crescimento na medida que o número de tarefas e máquinas também aumentam, conforme esperado. Analisando isoladamente o comportamento da curva dentro de cada intervalo de tarefas, através de curvas de tendência, conclui-se que todas elas são representadas satisfatoriamente por curvas quadráticas com R^2 a partir de 0,91, no entanto as curvas chegam a um R^2 aproximadamente, com 4 algarismos significativos, igual a 1 para uma curva polinomial de ordem maior ou igual a 4.

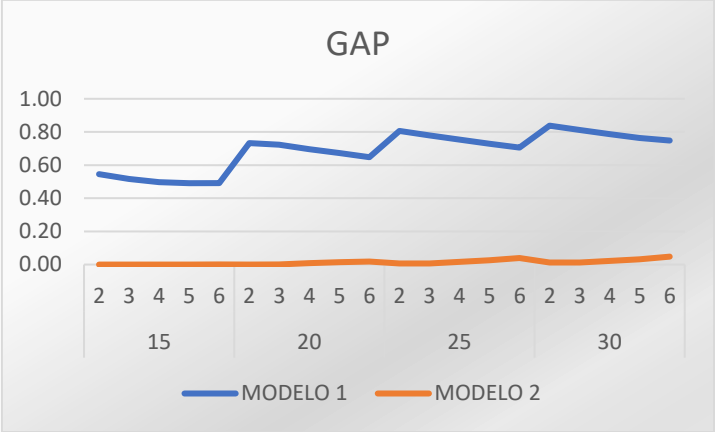


Grafico 2: gap médio de otimalidade

Por meio do Gráfico 3, nota-se que o *gap* médio de otimalidade para o modelo 2 é constante e aproximadamente igual a zero, já que a maior parte dos problemas alcançou a solução ótima. Já a curva ilustrada para o modelo 1, apresenta uma pequena tendência de crescimento na medida que o número de tarefas também aumenta. Explorando de forma isolada o comportamento da curva dentro de cada intervalo de tarefas, através de curvas de tendência, conclui-se que todas elas são representadas satisfatoriamente por funções lineares negativas com R^2 para 15 tarefas igual a 0,83 e para as demais, a partir de 0,98, no entanto as curvas chegam a um R^2 aproximadamente, com 4 algarismos significativos, igual a 1 para uma curva polinomial de ordem maior ou igual a 4.

5. Conclusões

A proposta deste estudo foi examinar a derivação de um problema clássico de programação de tarefas em um ambiente *flowshop*, visando minimizar o tempo médio de fluxo, e consequentemente o estoque em processamento, contribuindo com a redução de custos de armazenamento. O problema considerou como restrição e limitação o *setup* independente da sequência, para tornar o problema mais realista, já que desta forma o *setup* pode ser realizado antes da conclusão da operação de trabalho na máquina anterior, caso haja algum tempo ocioso na máquina seguinte.

Desta forma, para atingir o objetivo proposto, foram elaborados dois modelos de programação linear inteira mista, baseados em diferentes estratégias de formulação. Para estes métodos de solução, foram feitas comparações teóricas por meio da análise do tamanho do problema em relação a cada formulação, e também a comparação prática, por meio da implementação da experimentação computacional.

Foram resolvidos na experimentação computacional 1000 problemas-testes, avaliando-se o tempo médio de CPU e o *gap* de otimalidade. Os resultados mostraram que em geral, o modelo baseado em variáveis de posição apresenta menor tempo médio de CPU e também menores *gaps* quando comparado ao modelo baseado em variáveis de precedência. Como proposta para trabalhos futuros, sugere-se a criação de novos modelos baseados em diferentes paradigmas, como por exemplo o

modelo indexado no tempo, e aplicação de possíveis técnicas computacionais que melhorem a convergência do tempo de CPU e dos *gaps*.

Referências

Allahverdi, A., Gupta, J. N., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*(2), 219-239.

Allahverdi, A., Ng C. T., Cheng, T. C. E. e Kovalyov, M. Y. (2008), A survey of scheduling problems with setup times or costs, European Journal of Operational Research, 187, 985-1032.

Boiko, T. J. P. (2013). Métodos heurísticos para a programação em flow shop permutacional com tempos de setup separados dos tempos de processamento e independentes da seqüência de tarefas (Doctoral dissertation, Universidade de São Paulo).

Cheng, T. E., Gupta, J. N., & Wang, G. (2000). A review of flowshop scheduling research with setup times. Production and operations management, 9(3), 262-282.

Davis, M. M., Chase, R. B., & Aquilano, N. J. (2001). Fundamentos da administração da produção. Bookman.

Fuchigami, H.Y. (2015). Sequenciamento da produção em sistemas flow shop. Goiânia: Gráfica UFG.

GAITHER, N.; FRAZIER, G. (2002). Administração da produção e operações. 8. Ed. São Paulo: Pioneira.

Johnson, S.M. (1954), Optimal two and three-stage production schedules with set-uptimes included. Naval Research Logistics Quarterly,1,–68.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., & Werner, F. (2009). A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Computers & Operations Research, 36(2), 358-378.

Pinedo, M. L. (2016). Scheduling: theory, algorithms, and systems. Springer.

Ruiz, R., & Allahverdi, A. (2007). Some effective heuristics for no-wait flowshops with setup times to minimize total completion time. Annals of Operations Research, 156(1), 143-171.

Silva, A. A. D. Heurística evolutiva para problemas de programação em no-wait flowshop com tempos de setup(Doctoral dissertation, Universidade de São Paulo).,2012

Zandieh, M., Ghomi, S. F., & Husseini, S. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation, 180(1), 111-127.

Agradecimentos: Esta pesquisa teve o apoio do CNPq