UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

FERNANDO SIQUEIRA DE ALMEDA

Heurísticas para o problema de programação *no-wait flowshop*

São Carlos
2018

FERNANDO SIQUEIRA DE ALMEIDA

Heurísticas para o problema de programação *no-wait flowshop*

Monografia apresentada ao Curso de Engenharia de Produção, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro de Produção.

Orientador: Prof. Dr. Marcelo Seido Nagano

Área de concentração: Pesquisa Operacional

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Siqueira de Almeida, Fernando
S363c Heurísticas para o problema de programação *no-wait flowshop* / Fernando Siqueira de Almeida; orientador Marcelo Seido Nagano. São Carlos, 2018.

Monografia (Graduação em Engenharia de Produção) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. no-wait flowshop. 2. makespan. 3. tempo médio de fluxo. 4. tempo total de fluxo. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Candidato: Fernando Siqueira de Almeida
Título do TCC: Heurísticas para o problema de programação no-wait flowshop com avaliação multicritério
Data de defesa: 13/11/2018

Comissão Julgadora	**Resultado**
Professor Doutor Marcelo Seido Nagano (orientador)	Aprovado
Instituição: EESC - SEP	
Professor Doutor Humberto Filipe de Andrade Januário Bettini	Aprovado
Instituição: EESC - SEP	
Pesquisador HUGO HISSASHI MIYATA	Aprovado
Instituição: EESC - SEP	

Presidente da Banca: **Professor Doutor Marcelo Seido Nagano**

A minha família e amigos pelo apoio e carinho.

AGRADECIMENTOS

Ao Prof. Dr. Marcelo Seido Nagano pelo incentivo, pelas orientações e pela grande oportunidade de aprendizado que me proporcionou durante o trabalho.

Ao meu amigo e colega de laboratório Hugo Hissashi Miyata, pelo apoio técnico, sugestões e esclarecimento de dúvidas.

Ao meu amigo e colega de quarto Vinicius Ribeiro da Silva por todo o auxílio e paciência.

Aos docentes e aos funcionários do Departamento de Engenharia de Produção da EESC – USP.

Muito obrigado a todos!

“Intelligence is not a privilege, it's a gift, to be used for the good of mankind”.

(SPIDER-MAN 2, 2004)

RESUMO

ALMEIDA, F. S. **Heurísticas para o problema de programação *no-wait flowshop***. 2018. 22 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Este trabalho aborda o problema de programação *no-wait flowshop*. Dois objetivos são considerados: (1) minimizar o *makespan* sujeito à restrição de que o tempo médio de fluxo é menor ou igual a um dado valor; e (2) minimizar o tempo total de fluxo sujeito à restrição de que o *makespan* é menor ou igual a um dado valor. Dado que esses problemas são considerados intratáveis (*NP-Hard*), diversos métodos heurísticos têm sido propostos. Para cada um dos dois objetivos, é proposta uma adaptação da meta-heurística de Ruiz e Stützle (2007), *Iterated Greedy with Local Search* (GL), com cinco versões L (1, 5, 10, 15 e 20). As cinco versões de GL adaptadas para o objetivo 1 são comparadas com a heurística HH1, proposta por Aydilek e Allahverdi (2012). E as cinco versões de GL adaptadas para o objetivo 2 são comparadas com a heurística PA20, proposta por Allanhverdi e Aydilek (2013). As heurísticas são avaliadas em problemas gerados aleatoriamente, com diferentes números de tarefas e máquinas, e nas mesmas condições iniciais. Todos os resultados são verificados estatisticamente. Os experimentos computacionais relativos ao objetivo 1 mostram que o erro relativo médio geral de G20 é menor do que o de HH1, enquanto o tempo de CPU de G20 é significativamente menor que o de HH1. Portanto, o algoritmo G20 é superior a heurística HH1. Da mesma forma, os experimentos computacionais relacionados ao objetivo 2 mostram que os erros relativos médios gerais de G10, G15 e G20 são menores do que o de PA20. Portanto, os algoritmos G10, G15 e G20 superam a performance da heurística PA20.

Palavras-chave: *No-wait flowshop*. *Makespan*. Tempo médio de fluxo. Tempo total de fluxo.

ABSTRACT

ALMEIDA, F. S. **Heuristics for the no-wait flowshop scheduling problem**. 2018. 22 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

This work addresses the *no-wait flowshop* scheduling problem. Two objectives are considered: (1) minimizing makespan subject to the constraint that mean completion time is less than or equal to a given value; and (2) minimizing the total completion time subject to the constraint that the makespan is less than or equal to a given value. Since these problems are considered intractable (NP-Hard), several heuristic methods have been proposed. For each of the two objectives, it is proposed an adaptation of Ruiz and Stützle' metaheuristic (2007), Iterated Greedy with Local Search (GL), with five versions L (1, 5, 10, 15 and 20). The five versions of GL adapted for objective 1 are compared with the HH1 heuristic proposed by Aydilek and Allahverdi (2012). And the five versions of GL adapted for objective 2 are compared with the PA20 heuristic proposed by Allanhverdi and Aydilek (2013). The heuristics are evaluated on randomly generated problems, with different numbers of jobs and machines, and under the same initial conditions. All results are statistically verified. Computational experiments related to objective 1 show that the overall average relative error of G20 is smaller than that of HH1, while the CPU time of G20 is significantly smaller than that of HH1. Therefore, the G20 algorithm is superior to the HH1 heuristic. In the same way, computational experiments related to objective 2 show that the overall average relative errors of G10, G15 and G20 are smaller than that of PA20. Therefore, the G10, G15 and G20 heuristics outperform the PA20 heuristic.

Keywords: No-wait flowshop. Makespan. Mean completion time. Total completion time.

SUMÁRIO

1 INTRODUÇÃO

O problema de programação *flowshop* tem sido objeto de estudado desde a década de 50 (GUPTA; STAFFORD, 2005; JOHNSON, 1954), sendo conhecido por sua complexidade (GAREY; JOHNSON; SETHI, 1976) e extensa área de aplicação (HALL; SRISKANDARAJAH, 1996). O problema *flowshop* regular consiste em *n* tarefas a serem processadas em *m* máquinas. Cada tarefa requer *m* operações e cada operação requer uma diferente máquina. Todos as tarefas são processadas na mesma ordem de máquinas (AYDILEK; ALLAHVERDI, 2012). Em um problema *flowshop* regular, um *buffer* infinito é assumido e as tarefas podem aguardar entre as máquinas (NAGANO; MIYATA, 2016a).

No entanto, existem outras situações em que filas e o processamento descontínuo não são permitidos, como o problema de programação *no-wait flowshop*. De acordo com Allahverdi e Aydilek (2013), "Um problema *no-wait flowshop* ocorre quando as operações de um tarefa devem ser processadas continuamente do início ao fim, sem interrupções, tanto no início como entre as máquinas". Minimizar o tempo total de fluxo e o *makespan* são dois objetivos comuns nesse problema de programação. Minimizar o *makespan* é importante em situações nas quais é necessário a conclusão de um lote completo de produtos o mais rápido possível. Minimizar o tempo total de fluxo é importante em situações nas quais é necessário a entrega de cada produto assim que for concluído (ALLAHVERDI; AYDILEK, 2013; AYDILEK; ALLAHVERDI, 2012). Como o problema *no-wait flowshop* é considerado intratável (*NP-hard*), os métodos propostas não garantem solução ótima (NAGANO; MIYATA, 2016a). Portanto, o desenvolvimento de heurísticas para obtenção de boas soluções a um custo computacional razoável representa um importante tema de estudo.

Neste trabalho, é abordado o problema de programação *no-wait flowshop* considerando duas situações: (1) minimizar o *makespan* sujeito à restrição de que o tempo médio de fluxo é menor ou igual a um dado valor; e (2) minimizar o tempo total de fluxo sujeito à restrição de que o *makespan* é menor ou igual a um dado valor. O objetivo é propor heurísticas para os dois casos e avaliar seu desempenho frente a outras heurísticas equivalentes na literatura. O restante deste trabalho é organizado da como se segue. O Capítulo 2 é dedicado a revisão da literatura. No Capítulo 3, é apresentada a definição do problema nos dois casos de otimização. No Capítulo 4, é apresentado dois algoritmos para obtenção das restrições e soluções iniciais. No Capítulo 5, é feita a descrição das heurísticas avaliadas. O Capítulo 6 é dedicado aos resultados do experimento computacional. Por fim, as conclusões são apresentadas no Capítulo 7.

2 REVISÃO BIBLIOGRÁFICA

Diferente do problema de programação *flowshop* regular, no qual é assumido um *buffer* infinito, o problema *no-wait flowshop* caracteriza-se por ter suas tarefas processadas sem interrupção entre máquinas consecutivas. Para esse tipo de problema, são encontradas diversas formas de aplicações práticas. As indústrias de metal, plástico, produtos químicos e de alimentos são alguns exemplos. Nessas indústrias, existem alguns parâmetros do material em processo, como por exemplo temperatura e viscosidade, que exigem que cada operação siga a anterior imediatamente (HALL; SRISKANDARAJAH, 1996). O *no-wait flowshop* também é amplamente aplicado na manufatura de semicondutores (CHIEN et al., 2008; RITZO et al., 2011) e na manufatura de placas de circuito impresso (CHE; CHU, 2007). Além disso, ambientes modernos como sistemas flexíveis de manufatura, *just-in-time* e manufatura ágil também podem ser modelados como um problema de programação *no-wait* (BERTOLISSI, 2000).

Minimizar o tempo total de fluxo (ou o equivalente tempo médio de fluxo) e o *makespan* são dois objetivos amplamente estudados pelos pesquisadores. Minimizar o tempo total de fluxo de todas as tarefas é importante em situações nas quais cada produto concluído é necessário assim que for processado. Esse objetivo também traz benefícios quando se busca reduzir inventário ou conter custos. Reduzir o *makespan* é um objetivo importante em situações nas quais o recebimento de um lote completo de produtos é requerido assim que possível. Além disso, minimizar o *makespan* tende a aumentar a utilização de máquinas e recursos (ALLAHVERDI; AYDILEK, 2013). Ambos os objetivos têm sido amplamente abordados para diferentes ambientes de programação.

O primeiro registro de estudo sobre o problema *no-wait flowshop* com o objetivo de minimizar o *makespan* vem da década de 70, por Van Deman e Baker (1974). Desde então, muitas heurísticas já foram propostas para o problema, como por exemplo as de Bonney e Gundry (1976), King e Spachis (1980), Gangadharan e Rajendran (1993), Rajendran (1994) e Aldowaisan e Allahverdi (2003). Grabowski e Pempera (2005) comparou várias de suas heurísticas com duas heurísticas de busca local propostas por Schuster e Framinan (2003), nas quais as heurísticas de busca local foram comparadas com a heurística proposto por Rajendran (1994). Framinan e Nagano (2008) propuseram uma heurística baseada no problema *no-wait flowshop* e o problema *Traveling Salesman*. Tseng e Lin (2010) apresentaram um algoritmo genético híbrido e Zhu et al. (2009) apresentaram um algoritmo de busca local. Qian et al. (2009) propuseram para o problema um algoritmo do tipo evolução diferencial híbrido. Já

Nagano e Miyata (2016b) propuseram uma heurística construtiva a partir de sequências parciais das tarefas.

Na literatura, também são encontrados diversos trabalhos sobre o problema *no-wait flowshop* voltados ao objetivo de minimizar o tempo total de fluxo. Rajendran e Chaudihuri (1990), por exemplo, propuseram duas heurísticas que se provaram superiores às suas predecessoras. Chen et al. (1996) posteriormente desenvolveu um algoritmo genético (GA) e o comparou com os algoritmos de Rajendran e Chaudihuri (1990). Fink e VoB (2003) examinou a aplicação de diferentes métodos heurísticos. Aldowaisan e Allahverdi (2004) propuseram várias outras heurísticas que se provaram superiores as de Rajendran e Chaudihuri (1990) e as de Chen et al. (1996). Shyu et al. (2004) propuseram uma heurística de otimização de colônia de formigas e a compararam com as heurísticas anteriores. Pan et al. (2008) apresentaram um algoritmo de otimização *particle swarm* para o problema. Framinan et al. (2010) propuseram uma nova heurística para o problema e mostraram que ela possuía melhor performance do que as heurísticas pré-existentes. Nagano et al. (2012) abordou o mesmo problema, mas considerou os tempos de *setup* de trabalho separado dos tempos de processamento. Nagano et al. (2012) propuseram uma combinação de GA e pesquisa de *cluster* que se mostrou superior às heurísticas anteriores.

Além dos trabalhos mencionados até então, que consideram a minimização de somente uma medida de performance, existem outros estudos que abordaram o problema considerando mais de um parâmetro. Allahverdi e Aldowaisan (2002) consideraram o problema de otimização com *makespan* e o tempo total de fluxo, reduzindo o problema por meio da conversão dos dois parâmetros para uma medida, em uma soma ponderada das duas. No caso do estudo de Framinan e Leisten (2006), foi considerado um *flowshop* regular (sem *no-wait*) como o objetivo de minimizar o *makespan* sujeito a restrição de que o máximo atraso devesse ser menor ou igual a um dado valor. Aydilek e Allahverdi (2012) abordaram o problema no qual o objetivo era minimizar o *makespan* sujeito à restrição de que o tempo médio de fluxo não fosse maior ou igual a um dado valor. Logo em seguida, Allahverdi e Aydilek (2013) abordaram o problema com o objetivo de minimizar o tempo total de fluxo sujeito à restrição de que o *makespan* não fosse maior ou igual a um dado valor. Mais recentemente, Allahverdi et al. (2018) propuseram um algoritmo para minimizar o atraso total sujeito à restrição de que o *makespan* não fosse maior do que um dado valor.

Revisões abrangentes sobre o estado da arte nesta área de programação são apresentadas por Hall e Sriskandarajah (1996), Nagano e Miyata (2016a) e Allahverdi (2016).

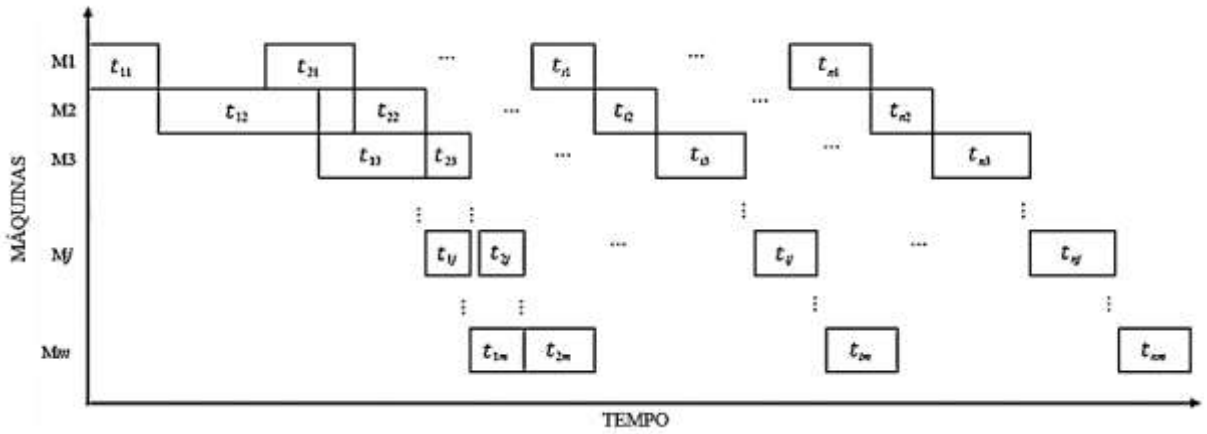
3 DEFINIÇÃO DO PROBLEMA

Neste Capítulo, é apresentada a descrição do problema de programação *no-wait flowshop* e os dois objetivos de otimização. Deve ser notado que o tempo médio de fluxo e o tempo total de fluxo são medidas de performance equivalentes. Os problemas só foram abordados dessa forma para facilitar a comparação com as heurísticas da literatura selecionadas para avaliação.

3.1 O PROBLEMA DE PROGRAMAÇÃO *NO-WAIT FLOWSHOP*

Seja $\{J_1, J_2, ..., J_n\}$ um conjunto de n tarefas e $\{M_1, M_2, ..., M_m\}$ um conjunto de m máquinas. Sejam $O_{i,j}$ e $t_{i,j}$ respectivamente a operação e o tempo de processamento da tarefa J_j na máquina M_i. O problema de programação *flowshop* regular ocorre quando cada uma das n tarefas são processadas em todas as m máquinas com a mesma sequência de processamento ou ordem de máquinas. Quando o *flowshop* apresenta o fenômeno *no-wait*, as tarefas não podem esperar entre duas máquinas sucessivas. Isso implica que, se necessário, o início de uma tarefa deve ser atrasado na primeira máquina, para garantir que o fim de uma operação coincida com o início da operação seguinte na máquina subsequente (NAGANO; MIYATA, 2016a). Um gráfico de *Gantt* ilustrando uma programação genérica do *no-wait flowshop* é apresentado na Figura 1.

Figura 1 – Gráfico de *Gantt* do problema de programação *no-wait flowshop*



Fonte: Adaptado de (NAGANO; MIYATA, 2016a).

Para a abordagem do *no-wait flowshop*, são assumidas as seguintes condições:

(1) As máquinas estão à disposição em 100% do tempo (nunca se quebram);
(2) Cada máquina pode processar no máximo um trabalho por vez;
(3) Qualquer tarefa pode ser processada em no máximo uma máquina por vez;
(4) Todos as tarefas estão disponíveis desde o início do horizonte de planejamento;
(5) Uma vez iniciada a operação, ela é processada até ser concluída;
(6) Os tempos de *setup* são ignorados ou assumidos inclusos nos tempos de processamento;
(7) Os tempos de processamento são determinados e conhecidos antecipadamente.

Seja $d_{i\text{-}1,i}$ o atraso mínimo na primeira máquina entre o final da tarefa J_i e o início da tarefa J_{i+1}. E seja $\{a_k\}_{k=x}^{y}$ uma sequência cujo domínio é dado por $\{k \in \mathbb{N} \mid x \leq k \leq y\}$. Nessas condições, $d_{i\text{-}1,i}$ pode ser definido como:

$$\forall\, i \in \mathbb{N} \mid 2 \leq i \leq n,$$
$$d_{i-1,i} = max\left(\left\{\sum_{j=2}^{k} t_{i-1,j} - \sum_{p=1}^{k-1} t_{i,p}\right\}_{k=2}^{m},\ 0\right) \quad (1)$$

Seja C_i o tempo de conclusão de todas as operações da tarefa J_i. Assim temos:

$$C_i = \sum_{k=2}^{i} d_{k-1,k} + \sum_{p=1}^{i} t_{p,1} + \sum_{j=2}^{m} t_{i,j} \quad (2)$$

O *makespan* (C_{max}) é definido como o tempo necessário para o processamento de todas as tarefas do sistema, ou seja, o intervalo de tempo entre o momento de início do processamento da operação $O_{1,1}$ e o momento de conclusão do processamento da operação $O_{n,m}$. Desta forma, C_{max} (equivalente a C_n) pode ser calculado como:

$$C_{max} = \sum_{k=2}^{n} d_{k-1,k} + \sum_{p=1}^{n} t_{p,1} + \sum_{j=2}^{m} t_{n,j} \quad (3)$$

3.2 MINIMIZAÇÃO DE MAKESPAN SUJEITO AO TEMPO MÉDIO DE FLUXO

O tempo médio de fluxo (*TMF*) é definido como a média aritmética das somas dos tempos de conclusão de todas as tarefas do sistema, ou seja

$$TMF = \frac{1}{n} \sum_{i=1}^{n} C_i. \quad (4)$$

Seja *M* o limite superior para o *MTC*. Além disso, deixe que $C_{max}(\pi)$ e $TMF(\pi)$ representem o *makespan* e o tempo médio de fluxo de uma determinada sequência π. Nessas condições, o problema pode ser definido como:

$$\begin{aligned} &Minimizar\ Cmax(\pi) \\ &Subjeito\ ao\ TMF(\pi)\ \leq M \end{aligned} \tag{5}$$

Em outras palavras, o problema consiste em encontrar uma sequência de processamento π que minimize o C_{max} tal que o *TMF* seja menor ou igual ao valor *M*. Em um problema real, esse valor *M* dever ser dado pelo programador. No caso em que o valor *M* não é dado, ele pode ser obtido por um algoritmo, como o apresentado no Capítulo 4.

3.3 MINIMIZAÇÃO DO TEMPO TOTAL DE FLUXO SUJEITO AO MAKESPAN

O tempo total de fluxo (*TTF*) é definido como a soma dos tempos de conclusão de todas as tarefas do sistema, ou seja

$$TTF = \sum_{i=1}^{n} C_i. \tag{6}$$

Seja *K* o limite superior para o *TTF*. Além disso, deixe que $C_{max}(\pi)$ e $TTF(\pi)$ representem o *makespan* e o tempo total de fluxo de uma determinada sequência π. Nessas condições, o problema pode ser definido como:

$$\begin{aligned} &Minimizar\ TTF(\pi) \\ &Subjeito\ ao\ Cmax(\pi) \leq K \end{aligned} \tag{7}$$

Em outras palavras, o problema consiste em encontrar uma sequência de processamento π que minimize o *TTF* tal que o C_{max} seja menor ou igual ao valor *K*. Em um problema real, esse valor *K* dever ser dado pelo programador. No caso em que o valor *K* não é dado, ele pode ser obtido por um algoritmo, como o apresentado no Capítulo 4.

4 ALGORITMOS PARA OBTENÇÃO DAS SOLUÇÕES INICIAIS E RESTRIÇÕES

Neste capítulo, é apresentado o Algoritmo-M, proposto por Aydilek e Allahverdi (2012), para a obtenção de um limite superior para *TMF* (valor *M*) e o Algoritmo-K, proposto por Allahverdi e Aydilek (2013), para obtenção de um limite superior para o C_{max} (valor *K*), usados na obtenção das soluções iniciais e restrições das heurísticas para o experimento computacional.

4.1 ALGORITMO-M

Passo 1: Defina $p = 1$, $h = 1$

Passo 2: Selecione uma sequência aleatoriamente, chamada de sequência π_p. Assuma $M_p = MTC(\pi_P)$;

Passo 3: Permute as duas tarefas nas posições *h* e *h+1* da sequência π_P, e se o *MTC* da sequência depois da troca for menor do que M_p, então atualize a sequência π_P depois da troca e defina $M_p = TMF(\pi_P)$;

Passo 4: Defina $h = h+1$, se $h = n$, siga para o passo 5, caso contrário, volte ao passo 3;

Passo 5: Defina $p = p+1$, se $p = n$, siga para o passo 6, caso contrário, volte ao passo 2;

Passo 6: Set $M = \text{mín}(M_1, ..., M_n)$;

Passo 7: Assuma π como sendo a sequência onde *M* é obtida.

Segundo Aydilek e Allahverdi (2012), o algoritmo acima pode ser resumido como:

> *n* sequências são aleatoriamente selecionadas. Para cada sequência aleatória selecionada, defina um valor M_p para o C_{max}, então busque (pelo método da troca de pares) para encontrar um melhor valor *M*, onde M_p é atualizado cada vez que um melhor valor *M* é obtido. Ao final, *n* de *M* valores $(M_1, ..., M_n)$. O menor de $M_1, ..., M_n$ é definido como o valor *M*.

4.2 ALGORITMO-K

Passo 1: Defina $p = 1$, $h = 1$

Passo 2: Selecione aleatoriamente uma sequência, chamada de sequência πp. Defina $K_p = C_{max}(\pi_p)$;

Passo 3: Permute as duas tarefas nas posições *h* e $h + 1$ da sequência πp, e se o C_{max} da sequência depois da troca for menor que K_p, então atualize a sequência π_p e defina $K_p = C_{max}(\pi_p)$;

Passo 4: Defina $h = h + 1$. Se $h = n$, siga para o Passo 5; caso contrário, volte para o Passo 3;

Passo 5: Defina $p = p + 1$. Se $p = n$, siga para o Passo 6; caso contrário, volte para o Passo 2;

Passo 6: Defina $K = \text{mín}(K_1, ..., K_n)$;

Passo 7: Defina π como a sequência onde *K* é obtida.

Segundo Allahverdi e Aydilek (2013), o algoritmo acima pode ser resumido como:

> *n* sequências são selecionadas aleatoriamente. Para cada sequência selecionada, defina o K_p para o C_{max}, então procure (pelo método da troca de pares) o melhor valor *K*, onde K_p é atualizado cada vez que um melhor valor K é encontrado. Ao final, haverá n valores *K* (K_1, ..., K_n). O menor de K_1, ..., K_n é escolhido como o valor *K*.

5 HURÍSTICAS

Neste capítulo, são apresentadas as heurísticas avaliadas na experimentação computacional.

5.1 HURÍSTICA HH1

A heurística HH1, proposta por Aydilek e Allahverdi (2012) para minimizar o C_{max} sujeito ao *TMF*, é composta pela combinação de duas outras heurísticas: *modified Simulated Annealing* (mSA) e HA. Mais especificamente, a partir de uma sequência inicial, a heurística mSA gera uma nova solução que é usada como sequência inicial na heurística HA.

5.2.1 modified Simulated Annealing (mSA)

Existem duas grandes mudanças em relação ao *Simulated Annealing* tradicional. A primeira é o acréscimo de um teste para verificar se uma nova sequência s_t é factível, ou seja, se obedece a condição $TMF(s_t) \leq M$. A segunda é que, ao invés de permutar duas posições selecionadas aleatoriamente no Passo 4, um trabalho escolhido aleatoriamente é inserido em uma posição também aleatória. Os passos de mSA são descritos a seguir:

Passo 1: Defina a temperatura inicial T_i, a temperatura final T_f, fator de resfriamento *cf*, o número de repetições R_n, e a sequência inicial s_i (Algoritmo-M);
Passo 2: Defina a temperatura $T = T_i$, e a sequência $s = s_i$;
Passo 3: Defina $j = 1$;
Passo 4: Escolha dois números aleatórios *k* e *l* entre 1 e *n*. Insira a tarefa da sequência *s* da posição *k* para a posição *l*, e chame esta nova sequência de s_t ;
Passo 5: Calcule $L = F(s)$ e $Lt = F(s_t)$ no qual *F* é a função objetivo a ser minimizada;
Passo 6: Se $TMF(s_t) \leq M$, siga para o Passo 7. Caso contrário, vá para o passo 8;
Passo 7: Se $L_t < L$, atualize *s* com s_t, ou seja, $s = s_t$. Caso contrário, atualize *s* com s_t com probabilidade $exp(-d/T)$, em que $d = (Lt = L)/L$;
Passo 8: Defina $j = j + 1$. Se $j = R_n + 1$, siga para o Passo 9, caso contrário volte ao Passo 4;
Passo 9: Defina $T = T * cf$.
Passo 10: Se $T < T_f$, siga para o Passo 11, caso contrário volte ao passo 3.
Passo 11: *s* é a sequência solução adotada.

Os parâmetros da heurística escolhidos para a análise computacional foram $T_i = 0.10$, $T_f = 0.0001$, $cf = 0.98$ e $R_n = 50$, definidos como os de máxima performance por Aydilek e Allahverdi (2012).

5.2.2 Heurística HA

Na heurística HA, os 11 primeiros passos são repedidos *L* vezes, de forma que cada iteração se inicia com a sequência obtida no Passo 11. Desta forma, *L* é um parâmetro de entrada para a heurística HA. Todos os passos são descritos a seguir:

Passo 1: Usando uma sequência inicial π, defina um valor inteiro positivo para *L*, e defina d=1, $\theta_1=\pi$;
Passo 2: Defina $h = 1$;
Passo 3: Selecione a h-ésima tarefa da sequência θ_d e a insira em todas as n posições da sequência θ_d para obter n sequências. Chame essas sequências de $\pi_1, \pi_2, ..., \pi_n$;
Passo 4: Calcule $C_{max}(\pi_r)$ para $r = 1, 2, ..., n$;
Passo 5: Defina $r = 1$, $u = 1$;
Passo 6: Se $C_{max}(\pi_r) < C_{max}(\theta_d)$, e se $TMF(\pi_r) \leq M$, então assuma $\beta_u = \pi_r$, e defina $u = u + 1$;
Passo 7: Defina $r = r + 1$. Se $r = n + 1$, siga para o Passo 8, caso contrário volte ao Passo 6;
Passo 8: Encontre uma sequência entre β_u's (calculado no passo 6) com o mínimo C_{max}, e chame esta sequência de σ_h (se $u = 1$, então defina $\sigma_h = \theta_d$);
Passo 9: Defina $h = h + 1$. Se $h = n + 1$, siga para o Passo 10, caso contrário volte ao Passo 3;
Passo 10: Defina $d = d + 1$;
Passo 11: Encontre a sequência entre σ_h's com o mínimo C_{max}, e chame esta sequência θ_d;
Passo 12: Se $d < L$, siga para o passo 13, caso contrário volte ao Passo 2;
Passo 13: Encontre uma sequência entre os σ_h's com o mínimo C_{max}, e chame esta sequência θ;
Passo 14: Defina $f = 1$;
Passo 15: Permute as duas tarefas nas posições f e $f + 1$ da sequência θ, e chame a sequência depois da troca de φ. Se $C_{max}(\varphi) < C_{max}(\theta)$ e se $TMF(\varphi) \leq M$, então defina $\theta = \varphi$;
Passo 16: Defina $f = f + 1$. Se $f = n$, siga para o Passo 17, caso contrário volte ao passo 15;
Passo 17: A solução é a sequência θ.

O parâmetro de iteração da heurística escolhido foi L = 20, definido como o de máxima performance por Aydilek e Allahverdi (2012).

5.2 HEURÍSTICA PAL

O heurística PAL, proposto por Allahverdi e Aydilek (2013), busca minimizar o *TTF* sujeito ao C_{max} a partir de uma sequência inicial. Nessa heurística, os Passos de 3 a 11 são repetidos L vezes, de forma que o procedimento sempre se reinicia com a solução obtida no Passo 11. Assim, L é um parâmetro de entrada para o algoritmo PAL. Os passos de PAL são descritos a seguir:

Passo 1: Usando uma sequência inicial π, defina um valor para *L*, e defina $d = 1$ e $\theta_1 = \pi$
Passo 2: Defina $h = 1$;
Passo 3: Selecione a h-ésima tarefa da sequência θ_d e a insira em todas as n posições da sequência θ_d, obtendo n sequências, chamadas de sequências $\pi_1, \pi_2, ..., \pi_n$;
Passo 4: Calcule $TTF(\pi_r)$, para $r = 1, ..., n$;
Passo 5: Defina $r = 1$, $u = 1$;

Passo 6: Se $TTF(\pi_r) < TTF(\theta_d)$, e se $C_{max}(\pi_r) \leq K$, então deixe $\beta_u = \pi_r$, e defina $u = u + 1$;
Passo 7: Defina $r = r + 1$. Se $r = n + 1$, siga para o Passo 8; caso contrário, volte ao passo 6;
Passo 8: Encontre a sequência entre os β_u (calculada no Passo 6) com o menor *TTF*, e chame esta sequência de σ_h (se $u = 1$, então defina $\sigma_h = \theta_d$);
Passo 9: Defina $h = h + 1$. Se $h = n + 1$, siga para o Passo 10; caso contrário, volte ao Passo 3;
Passo 10: Defina $d = d + 1$;
Passo 11: Encontre a sequência entre os σ_n com o menor *TTF*, e chame está sequência de θ;
Passo 12: Se $d > L$, siga para o Passo 13; caso contrário, volte para o Passo 2;
Passo 13: Encontre a sequência entre os θ_d com o menor *TTF*, e chame esta sequência de θ;
Passo 14: Defina $f = 1$;
Passo 15: Permute as duas tarefas das posições f e $f + 1$ da sequência θ, e chame a sequência resultante de φ. Se $TTF(\varphi) < TTF(\theta)$, e se $C_{max}(\varphi) \leq K$, então defina $\theta = \varphi$;
Passo 16: Defina $f = f + 1$. Se $f = n$, siga para o Passo 17; caso contrário, volte ao Passo 15;
Passo 17: A solução é a sequência θ;

O parâmetro de iteração da heurística escolhido foi L = 20 (portanto PA20), definido como o de máxima performance por Allahverdi e Aydilek (2013).

5.3 HEURÍSICA PROPOSTA - *ITERATED GREEDY WITH LOCAL SEARCH* (GL)

Em resumo, a heurística *Iterated Greedy with Local Search*, proposta por Ruiz e Stützle (2007), executa a partir de uma solução inicial uma sequência de iterações na busca de um candidato a solução. O processo de obtenção da solução ocorre em duas fases: destruição e construção. Durante a destruição, um número determinado de elementos aleatórios é removido da solução anterior. Em seguida, na fase de construção, os elementos removidos são reinseridos para a construção de uma nova sequência. Também é adicionado um processo de pesquisa local para melhorar a solução encontrada. Por fim, um critério de aceitação é aplicado para verificar se o candidato a solução deve substituir a solução anterior. Todo o processo é repetido até que algum critério de parada seja satisfeito, como o tempo computacional ou um determinado número de iterações (NAGANO et al., 2015).

Nesta adaptação proposta, só são aceitas as sequências geradas que respeitam a restrição pela qual estão sujeitas (*M* ou *K*). A fase de destruição é aplicada removendo-se *d* tarefas da solução inicial. A fase de construção é implementada por meio da heurística construtiva NEH de Nawaz et al. (1983). O procedimento de busca local é composta pela combinação de dois processos (inserção e intercâmbio), conforme proposto por Nagano et al. (2015). O critério de aceitação é aplicando usando um parâmetro de temperatura *T*, semelhante ao critério do *Simulated Annealing* (YANG, 2010, p. 182). E o critério de parada é dado pelo número de iterações *L*.

O algoritmo da adaptação completa da heurística é ilustrado na Figura 2. A execução de cada etapa é descrita a seguir:

1. *Solução inicial*: Para minimizar o C_{max} sujeito ao *TMF*, é usado o Algoritmo-M para obter a sequência inicial π_0 e o valor *M*. Para minimizar o *TTF* sujeito ao C_{max}, é usado o Algoritmo-K para obter a sequência inicial π_0 e o valor *K*.
2. *Destruição*: Seja π a sequência em vigor no início da iteração. Das *n* tarefas de π, *d* tarefas são removidas aleatoriamente na ordem em que forem escolhidas. Desse procedimento, resulta a sequência π_R de tamanho *d*, contendo as tarefas removidas, e a sequência π_D de tamanho *n - d*, contendo as tarefas não removidas.
3. *Construção*: A primeira tarefa de π_R é inserida em todas as *n-d+1* posições de π_D, gerando *n-d+1* sequências. Feito isso, seleciona-se a melhor das sequências geradas e repete-se o procedimento de inserção até que π_D obtenha o tamanho n.
4. *Inserção*: Cada tarefa da sequência π_D é reinserida em todas as suas possíveis posições gerando $(n - 1)^2$ sequências. A solução é obtida selecionando-se a melhor das sequências geradas, chamada π', tal que $F(\pi') \leq F(\pi_D)$, em que $F(\pi')$ e $F(\pi_D)$ representam as funções objetivo (C_{max} ou *TTF*) de suas respectivas sequencias.
5. *Intercâmbio*: Realiza uma permutação entre pares de tarefas da sequência π', não necessariamente adjacentes, em todas as combinações possíveis gerando $n(n - 1)/2$ sequências. Da mesma forma, a solução é obtida selecionando a melhor das sequências geradas, chamada π'', tal que $F(\pi'') \leq F(\pi')$.
6. *Teste de aceitação*: A sequência candidata a solução π'' é aceita com uma probabilidade $exp(-\Delta/T_{emp})$, ou seja, π'' é aceito se $exp(-\Delta/T_{emp}) \geq r$, no qual *r* é um número aleatório tal que $0 \leq r \leq 1$. Os valores de Δ e T_{emp} são dados por:

$$\Delta = -\frac{F(\pi'') - F(\pi)}{F(\pi)}, \tag{8}$$

$$T_{emp} = T\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij}}{m \times n \times 10}. \tag{9}$$

Conforme apresentado na descrição, a heurística GL proposta possui três parâmetros de entrada: *d*, *T* e *L*. Para os testes computacionais, os dois primeiros parâmetros foram definidos como $d = 4$ e $T = 0{,}5$. Esses valores são os melhores encontrados para o problema de programação *flowshop* regular nos experimentos de Ruiz e Stützle (2007). Já para o

parâmetro *L*, foram definidos cinco valores (1, 5, 10, 15 e 20) para gerar as cinco versões de GL (G1, G5, G10, G15 e G20).

Figura 2 – Algoritmo para a heurística GL

```
procedimento GL
    π := π0 do Algoritmo-M ou Algoritmo-K;
    πb := π;
    for i := 1 to L do
        π' := π;                                  # Fase de destruição
        for i :=  1 to d do
            remova uma tarefa aleatória de π' e a insira em π'R;
        endfor
        for i :=  1 to d do                       # Fase de construção
            π' := melhor sequência obtida inserindo a tarefa π'R(i) em todas as possíveis posições de π'R;
        endfor
        π'' := inserção(π');                      # Busca local
        π'' := intercâmbio(π'');                  # Busca local
        if F(π'') < F(π) then                     # Critério de aceitação
            π := π'';
            if F(π'') < F(π) then                 # Teste se é nova melhor sequência
                πb := π;
            endif
        elseif (exp(– Δ/Temp) ≥ r) then
            π := π'';
        endif
    endfor
    return πb
end
```

Fonte: Adaptado de (RUIZ; STÜTZLE, 2007).

6 EXPERIMENTO COMPUTACIONAL

Testes computacionais foram realizados para todas as heurísticas apresentas. A heurística GL, com suas cinco versões L (1, 5, 10, 15, 20), foram adaptadas para os dois objetivos de otimização abordados.

A implementação foi efetuada em Python em um PC com CPU Intel Core i5-4200U 1.60 GHz com Impulso Turbo para 2.30 GHz, 6 GB de RAM e operando sob o sistema operacional Windows 10.

Foi usado um banco de dados de tempos de processamento com variação no número de tarefas *n* e no número de máquinas *m*. Os valores para *n* foram 15, 20, 25 e 30, enquanto para *m* foram 2, 3, 4, 5 e 6. Para cada combinação de *m* e *n*, foram gerados 25 problemas, totalizando 500 problemas. Os tempos de processamento foram aleatoriamente gerados com uma distribuição discreta uniforme U(1, 100), conforme a recomendação de Hall e Posner (2001) de usar uma distribuição de dados ampla para implementação. A partir desse material, foi criada para cada um dos dois objetivos outro banco de dados, desta vez com as soluções iniciais e restrições de todos os problemas, por meio da implementação do Algoritmo-M e do Algoritmo-K. Por fim, usando esses três bancos, foi implementada todas as heurísticas.

As performances foram avaliadas pelas porcentagens de erro relativo (*ER*). Sejam $\bar{F}(H')$ e $\bar{F}(H)$ as médias da função objetivo da heurística avaliada e da melhor heurística, respectivamente. A porcentagem *ER* é definida como:

$$ER = 100\left(\frac{\bar{F}(H') - \bar{F}(H)}{\bar{F}(H)}\right). \quad (10)$$

Os resultados foram comparados usando o teste HSD de *Tukey*. A avaliação é apresentada nos tópicos a seguir.

6.1 ANÁLISE DAS HEURÍTICAS PARA MINIMIZAÇÃO DE C_{MAX} SUJEITO AO *TMF*

A performance das heurísticas para a minimização de C_{max} sujeito ao *TMF* (G1, G5, G10, G15, G20 e HH1) são avaliadas nesta seção. Os resultados para o erro relativo são apresentados na Tabela 1, na qual cada valor representa a média de 25 problemas. Considerando a Média Geral, pode-se verificar que G20 obteve desempenho superior às demais heurísticas, seguido de HH1 e G15 que apresentaram valores muito próximos.

Tabela 1 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1

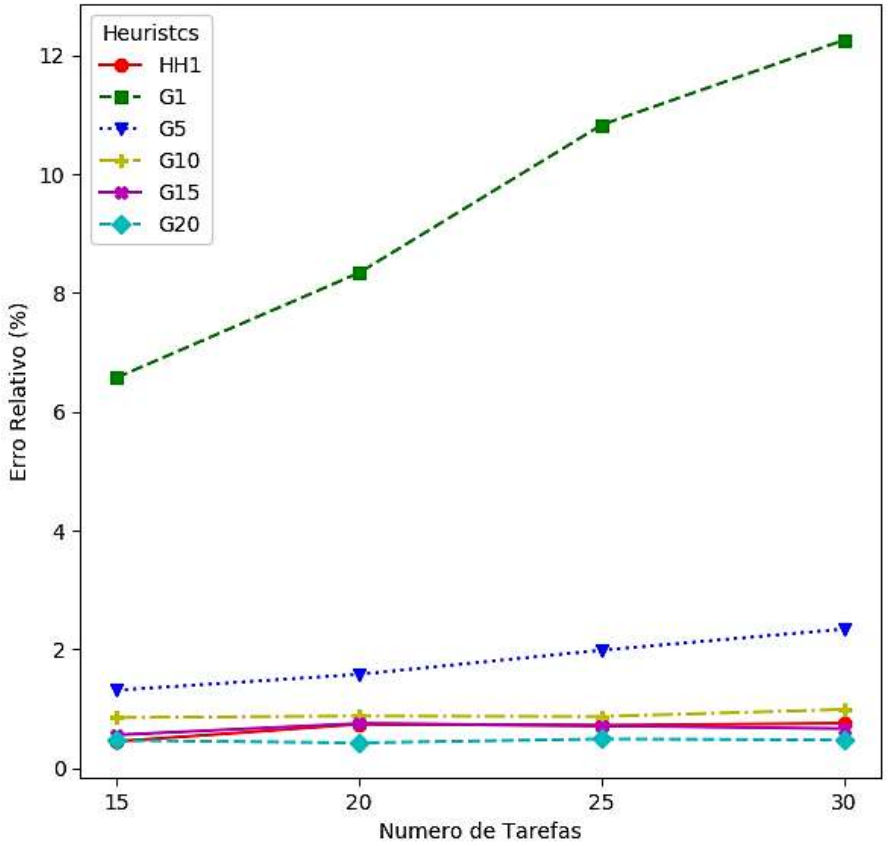
Tarefas	Máquinas	HH1	G1	G5	G10	G15	G20
15	2	0,41	1,97	0,39	0,12	0,26	0,11
	3	0,45	5,86	1,12	0,80	0,54	0,36
	4	0,43	7,64	1,53	1,00	0,44	0,64
	5	0,64	8,03	1,73	0,75	0,67	0,45
	6	0,34	9,37	1,80	1,60	0,89	0,79
20	2	0,29	4,01	0,47	0,28	0,22	0,21
	3	0,84	7,97	1,36	0,65	0,60	0,29
	4	0,90	8,45	1,53	1,05	0,96	0,38
	5	0,95	10,84	1,96	1,10	0,91	0,49
	6	0,74	10,42	2,59	1,32	1,08	0,79
25	2	0,49	4,81	0,37	0,21	0,14	0,13
	3	0,98	10,00	1,65	0,82	0,42	0,34
	4	0,99	11,94	2,71	1,10	0,89	0,48
	5	0,61	12,84	2,63	1,04	0,99	0,90
	6	0,56	14,54	2,58	1,19	1,14	0,62
30	2	0,44	5,49	0,66	0,26	0,15	0,14
	3	0,95	11,45	1,94	1,07	0,59	0,32
	4	0,82	13,97	3,06	0,98	0,61	0,52
	5	0,87	15,67	3,23	1,42	0,78	0,46
	6	0,71	14,70	2,82	1,23	1,21	0,97
Média Geral		**0,67**	**9,50**	**1,80**	**0,90**	**0,68**	**0,47**

Fonte: O Autor (2018).

Uma comparação mais precisa pode ser feita analisando as Figuras 3 e 4, que apresentam os valores do erro relativo projetados contra o número de tarefas e o número de máquinas, respectivamente. Na Figura 3, cada ponto representa a média de 125 pontos (25 problemas para 5 diferentes quantidades de máquinas). Na Figura 4, cada ponto representa a média de 100 pontos (25 problemas para 4 diferentes quantidades de máquinas).

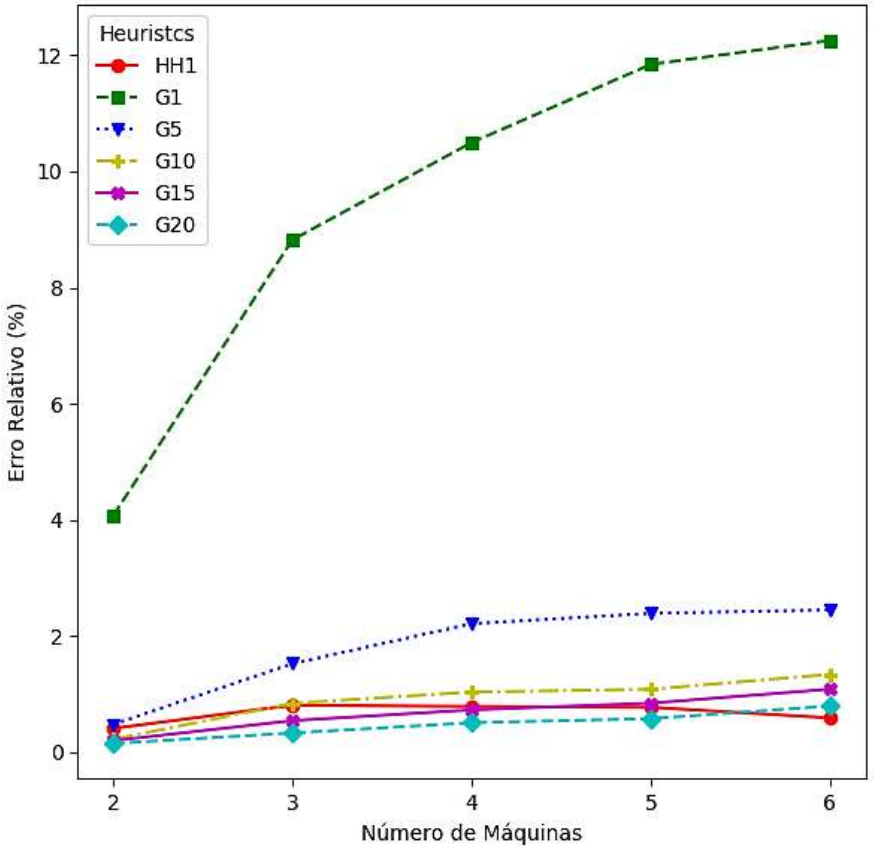
A partir das Figuras 3 e 4, pode-se notar que as heurísticas propostas G10, G15 e G20 são equiparáveis a heurística HH1. No entanto, HH1 apresenta um tempo de computacional muito maior, conforme ilustrado nas Figuras 5 e 6. De modo geral, à medida que o número de máquinas aumenta, o erro médio de todas as heurísticas também aumenta. Esse fenômeno é característico da otimização de C_{max}, que tende a maximizar a eficiência dos recursos. Quanto menor o número de máquinas, melhor é a distribuição dos recursos. O acréscimo de máquinas conturba o sistema até um ponto de estabilidade, o que explica o perfil das curvas. Já em relação ao número de tarefas, pode-se verificar uma estabilidade em torno da média do desvio relativo, principalmente entre as versões a partir de G10. G15, G20 e HH1.

Figura 3 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1 em relação ao número de tarefas



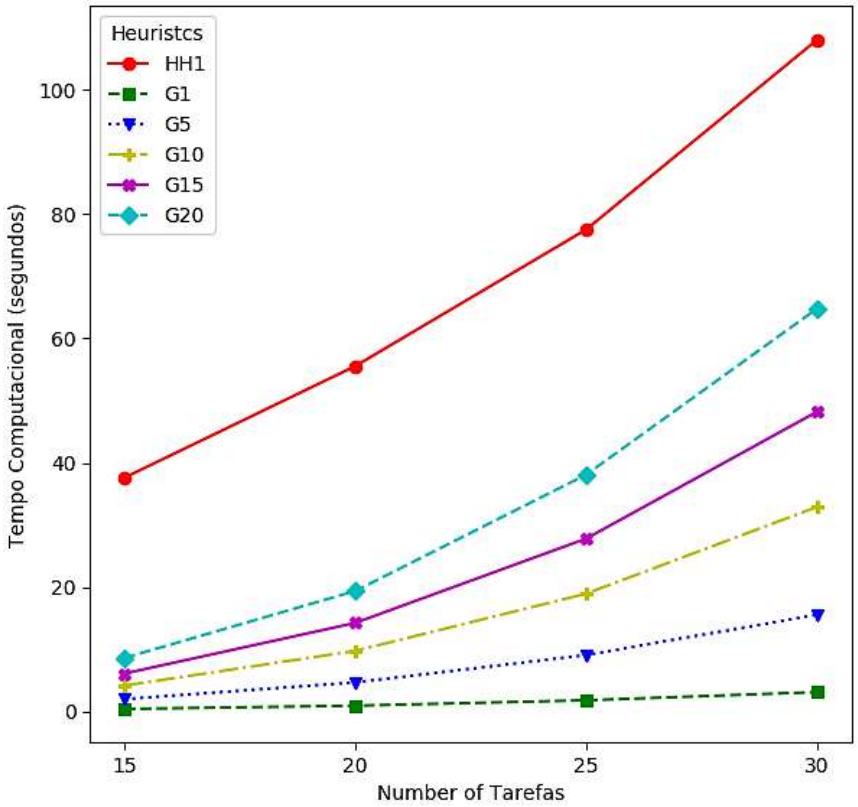
Fonte: O Autor (2018).

Figura 4 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1 em relação ao número de máquinas



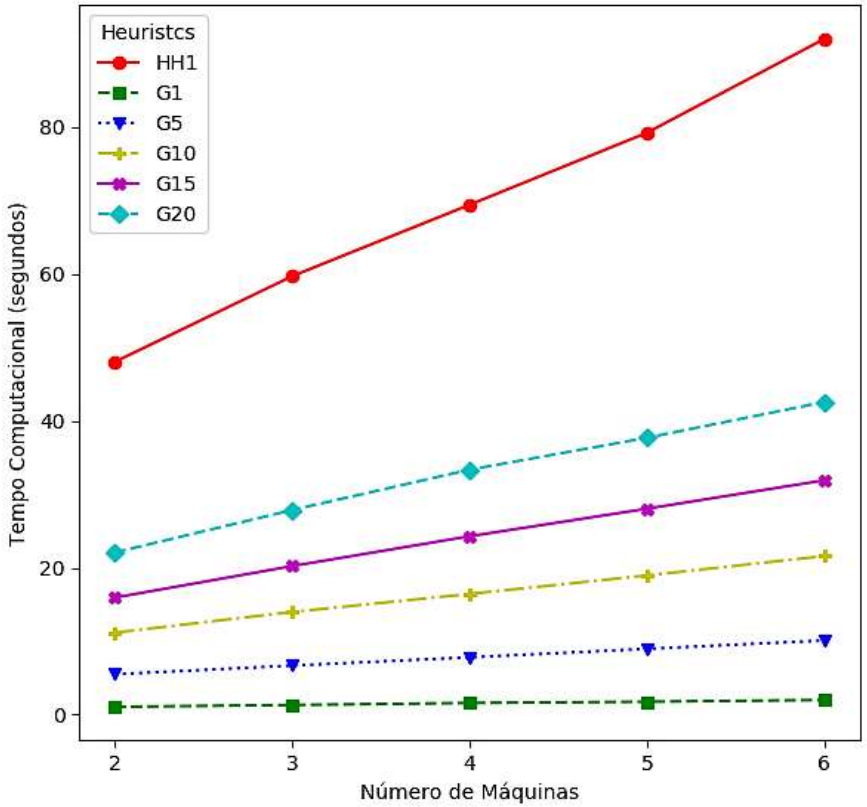
Fonte: O Autor (2018).

Figura 5 – Tempo computacional das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1 em relação ao número de tarefas



Fonte: O Autor (2018).

Figura 6 – Tempo computacional das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1 em relação ao número de máquinas



Fonte: O Autor (2018).

A Tabela 2 apresenta os resultados do teste de hipótese de *Tukey*, comparando os desvios relativos médios gerais das heurísticas com um nível de significância de 5% (0,05).

Tabela 2 – Resultados do teste *Tukey* das heurísticas GL (L = 1, 5, 10, 15 e 20) e HH1

(I) Heurística	(J) Heurística	Diferença média (I-J)	Erro Padrão	Significância	Intervalo de Confiança 95%	
					Limite inferior	Limite superior
HH1	G1	-8,82837*	0,13890	0,000	-9,2245	-8,4323
	G10	-0,23077	0,13890	0,558	-0,6269	0,1653
	G15	-0,00637	0,13890	10,000	-0,4024	0,3897
	G20	0,20110	0,13890	0,698	-0,1950	0,5972
	G5	-1,13781*	0,13890	0,000	-1,5339	-0,7417
G1	HH1	8,82837*	0,13890	0,000	8,4323	9,2245
	G10	8,59760*	0,13890	0,000	8,2015	8,9937
	G15	8,82201*	0,13890	0,000	8,4259	9,2181
	G20	9,02947*	0,13890	0,000	8,6334	9,4256
	G5	7,69057*	0,13890	0,000	7,2945	8,0866
G5	HH1	1,13781*	0,13890	0,000	0,7417	1,5339
	G1	-7,69057*	0,13890	0,000	-8,0866	-7,2945
	G10	0,90704*	0,13890	0,000	0,5110	1,3031
	G15	1,13144*	0,13890	0,000	0,7354	1,5275
	G20	1,33891*	0,13890	0,000	0,9428	1,7350
G10	HH1	0,23077	0,13890	0,558	-0,1653	0,6269
	G1	-8,59760*	0,13890	0,000	-8,9937	-8,2015
	G15	0,22441	0,13890	0,588	-0,1717	0,6205
	G20	0,43187*	0,13890	0,023	0,0358	0,8280
	G5	-,90704*	0,13890	0,000	-1,3031	-,5110
G15	HH1	0,00637	0,13890	10,000	-0,3897	0,4024
	G1	-8,82201*	0,13890	0,000	-9,2181	-8,4259
	G10	-0,22441	0,13890	0,588	-0,6205	0,1717
	G20	0,20746	0,13890	0,668	-0,1886	0,6035
	G5	-1,13144*	0,13890	0,000	-1,5275	-0,7354
G20	HH1	-0,20110	0,13890	0,698	-0,5972	0,1950
	G1	-9,02947*	0,13890	0,000	-9,4256	-8,6334
	G10	-0,43187*	0,13890	0,023	-0,8280	-0,0358
	G15	-0,20746	0,13890	0,668	-0,6035	0,1886
	G5	-1,33891*	0,13890	0,000	-1,7350	-0,9428

* A diferença média é significativa no nível 0,05.

Fonte: O Autor (2018).

A primeira e a segunda colunas indicam as heurísticas comparadas. A terceira coluna mostra a diferença média, ou seja, a diferença entre o erro relativo da heurística na primeira coluna menos o da heurística na segunda coluna. A quarta coluna mostra o erro padrão e a coluna seguinte, a significância. O símbolo * nos valores da quinta coluna indica se a diferença foi significativa.

O teste de *Tukey*, ao nível de 95% de confiança, apresentou 4 subconjuntos de médias, conforme pode-se verificar na Tabela 3. Esses grupos reúnem as heurísticas com médias sem diferença estatística significativa. Nota-se que a heurística HH1, se equipara com as heurísticas G15 e G20 no subconjunto 1, e com as heurísticas G15 e G10 no subconjunto 2.

Tabela 3 – Identificação de subconjuntos homogêneos ao nível de 95% de confiança, de acordo com teste de *Tukey*

Heurística	**N**	**Subconjunto para alfa = 0,05**			
		1	**2**	**3**	**4**
G20	500	0,4686			
HH1	500	0,6697	0,6697		
G15	500	0,6761	0,6761		
G10	500		0,9005		
G5	500			1,8075	
G1	500				9,4981
Significância		0,668	0,558	1,000	1,000

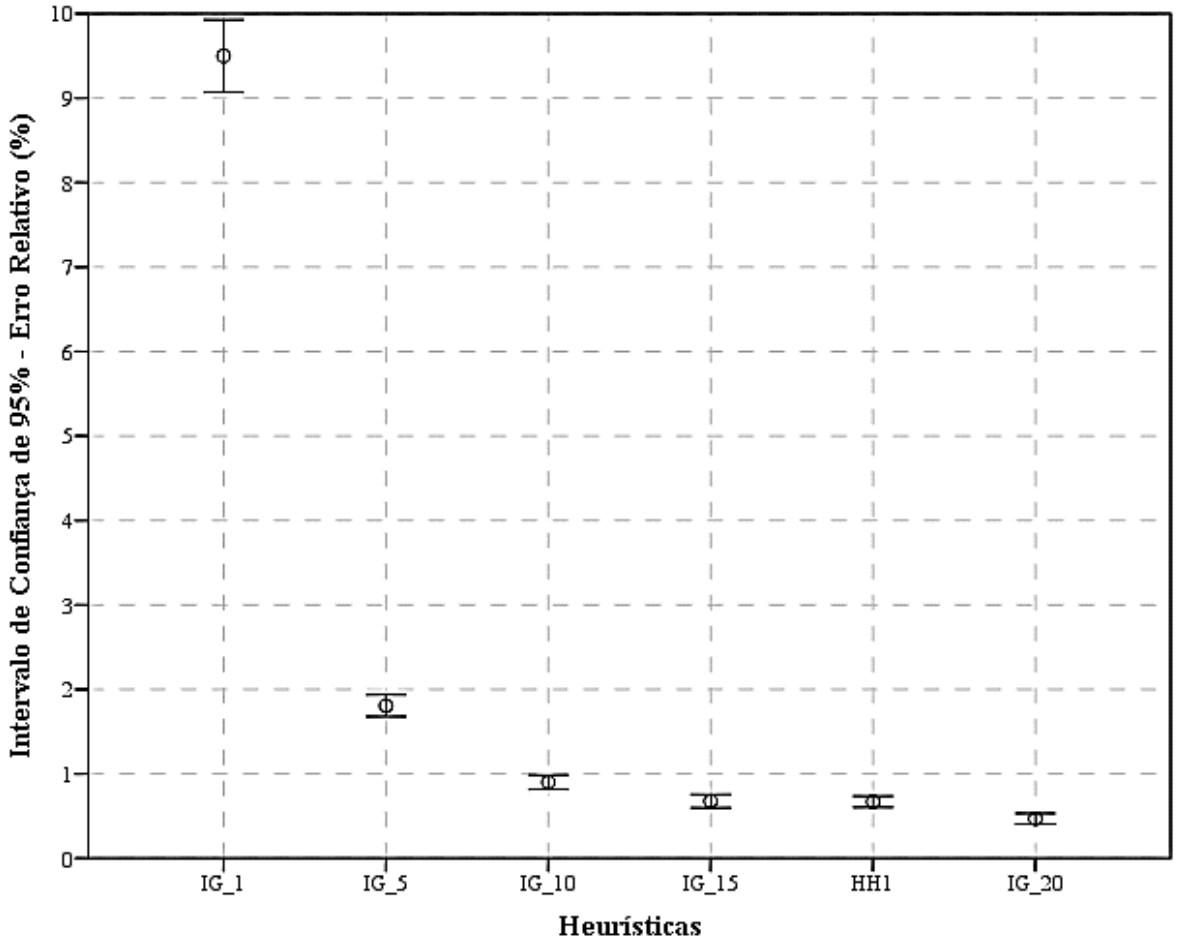
São exibidas as médias para os grupos em subconjuntos homogêneos.

Fonte: O Autor (2018).

A Figura 7 ilustra o desvio relativo médio geral e a variabilidade das heurísticas avaliadas. Além de apresentar a menor média, G20 possui uma variância menor dos desvios em comparação às outras heurísticas. A Figura também ilustra se as médias apresentam ou não diferença estatística significativa, como o caso no qual se sobreponham as médias de G15 e HH1.

Como era esperado, fica evidente o ganho de performance da heurística GL em função do aumento do número de iterações L. No entanto, o ganho diminui a cada acréscimo. Isso significa que a melhora da heurística custará cada vez mais tempo computacional e haverá um momento no qual não será mais possível obter ganho somente aumentando o valor desse parâmetro.

Figura 7 – Gráfico de médias, ao nível de 95% de confiança, do desvio relativo médio geral



Fonte: O Autor (2018)

6.2 ANÁLISE DAS HEURÍTICAS PARA MINIMIZAÇÃO DE *TTF* SUJEITO AO C_{MAX}

A performance das heurísticas para a minimização de *TTF* sujeito ao C_{max} (G1, G5, G10, G15, G20 e PA20) são avaliadas nesta seção. Os resultados para o erro relativo são apresentados na Tabela 4, na qual cada valor também representa a média de 25 problemas. Pode-se verificar que a heurística GL proposta supera a heurística PA20 a partir da versão G10 em todas as combinações de *m* e *n* testadas.

As Figura 8 e 9 apresentam os valores do erro relativo projetados contra o número de tarefas e o número de máquinas, respectivamente. Assim como na seção anterior, cada ponto na Figura 8 representa a média de 125 pontos (25 problemas para 5 diferentes quantidades de máquinas). E na Figura 9, cada ponto representa a média de 100 pontos (25 problemas para 4 diferentes quantidades de máquinas). Pode se verificar que, em relação a qualidade da solução, a heurística G5 alcança resultados equiparáveis a PA20.

Tabela 4 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20

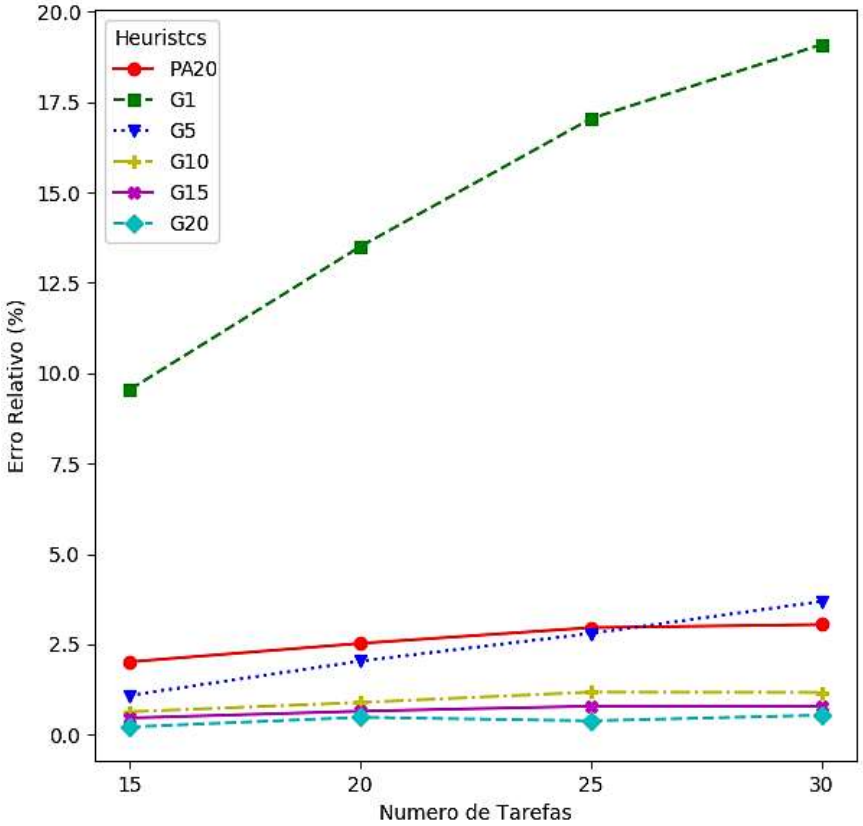
Tarefas	Máquinas	PA20	G1	G5	G10	G15	G20
15	2	2,30	10,52	0,99	0,57	0,37	0,06
	3	1,94	10,56	1,05	0,82	0,74	0,13
	4	2,64	10,05	1,45	0,61	0,44	0,33
	5	1,38	8,62	0,82	0,49	0,47	0,35
	6	1,83	7,97	1,15	0,71	0,31	0,22
20	2	2,43	16,25	1,59	0,37	0,63	0,20
	3	2,87	13,51	2,43	0,84	0,35	0,87
	4	3,00	11,81	1,68	1,28	0,76	0,74
	5	2,22	13,09	2,05	1,07	0,92	0,17
	6	2,12	12,87	2,45	0,91	0,62	0,45
25	2	3,13	19,04	2,73	0,67	0,28	0,22
	3	3,92	19,44	2,67	1,24	0,59	0,30
	4	2,95	15,33	3,12	1,28	1,15	0,28
	5	2,68	15,20	2,65	1,28	1,10	0,77
	6	2,15	16,21	2,85	1,43	0,85	0,37
30	2	3,66	24,01	3,51	0,84	0,63	0,24
	3	3,51	18,39	3,88	1,08	0,96	0,43
	4	2,75	16,93	3,24	0,71	0,69	0,62
	5	2,62	18,56	3,91	1,69	0,81	0,69
	6	2,71	17,58	3,91	1,55	0,88	0,74
Média Geral		**2,64**	**14,80**	**2,41**	**0,97**	**0,68**	**0,41**

Fonte: O Autor (2018).

De modo geral, o erro médio das heurísticas diminui à medida que o número de máquinas aumenta. Esse fenômeno é característico da minimização do *TTF*, que tende a otimizar a resposta do sistema para entrega e reduzir o inventário de tarefas. No entanto, quanto menor o número de máquinas, mais instável é o sistema. O acréscimo de máquinas ajuda a estabilizar o fluxo de tarefas, conforme o perfil das curvas. Por outro lado, assim como na otimização de C_{max}, mantem-se a tendência de aumento do erro relativo médio à medida que o número de tarefas aumenta, principalmente em relação às heurísticas propostas.

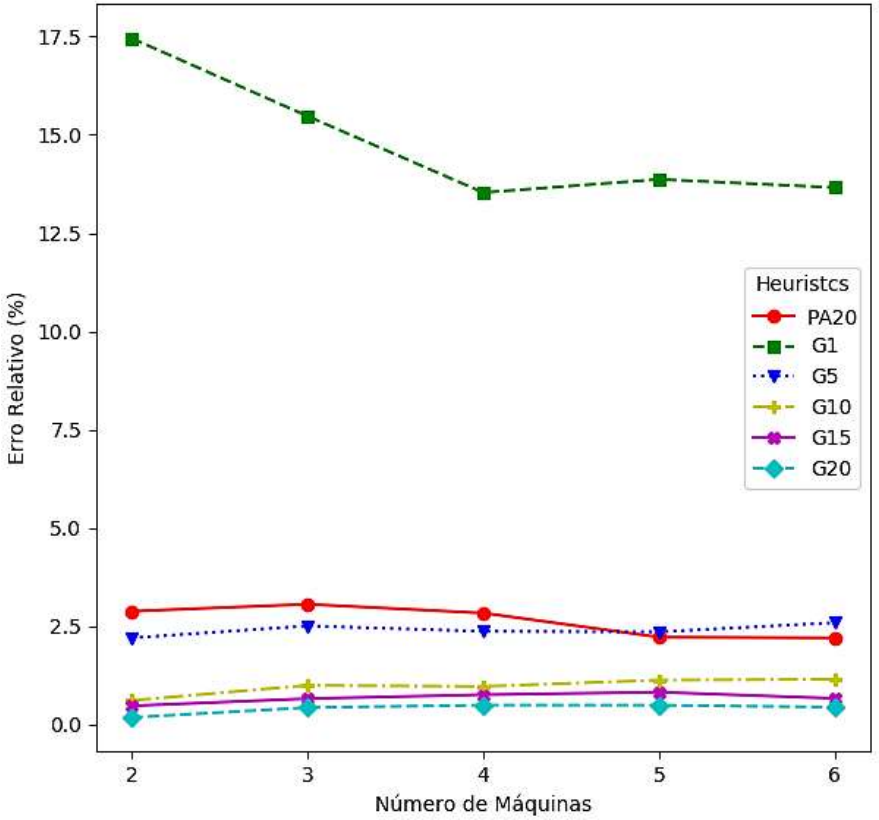
A Tabela 5 apresenta os resultados do teste de hipótese de *Tukey*, comparando as médias entre as heurísticas com um nível de significância de 5% (0,05). A partir dos dados, pode-se identificar que há diferença significativa das heurísticas G10, G15 e G20 em comparação com PA20. Além disso, embora PA20 apresente um tempo de computacional inferior a G20, seu tempo é equiparável a G15 e superior a G10, conforme ilustrado nas Figuras 10 e 11. Essas análises, somadas a avaliação do gráfico de médias da Figura 12, indicam que as 3 versões GL propostas superam a heurística PA20.

Figura 8 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20 em relação ao número de tarefas



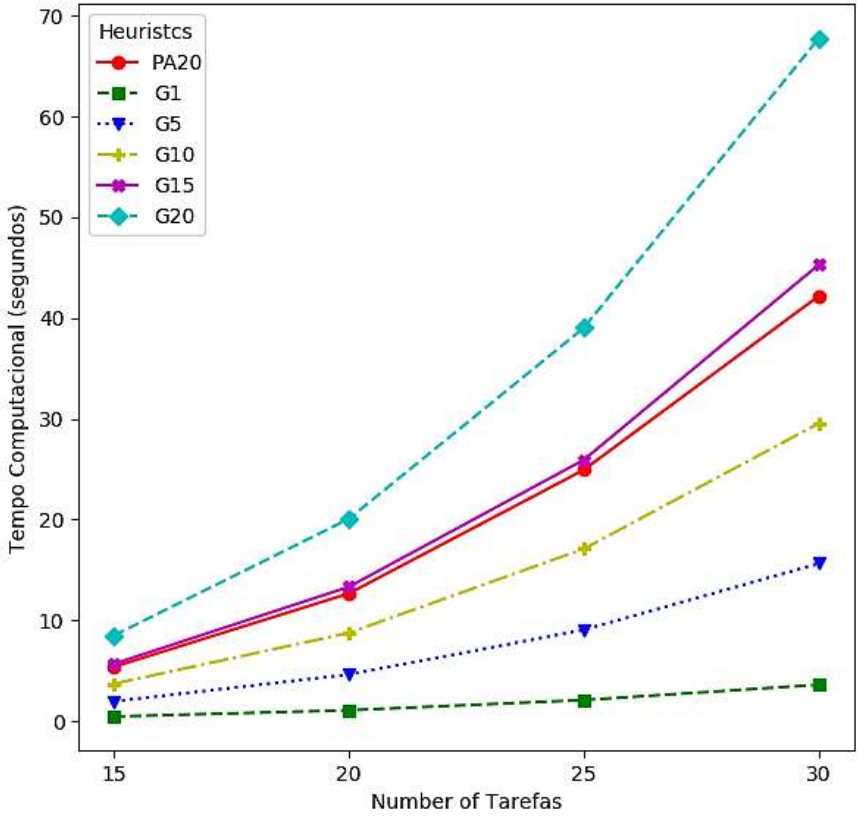
Fonte: O Autor (2018).

Figura 9 – Erro relativo médio das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20 em relação ao número de máquinas



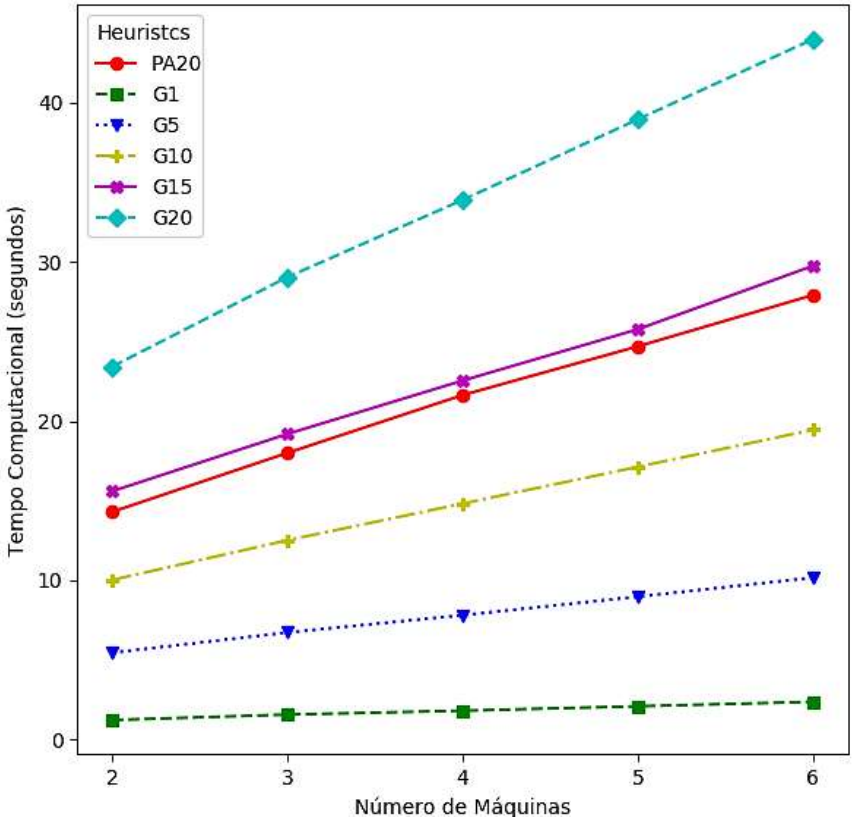
Fonte: O Autor (2018).

Figura 10 – Tempo computacional das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20 em relação ao número de tarefas



Fonte: O Autor (2018).

Figura 11 – Tempo computacional das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20 em relação ao número de máquinas



Fonte: O Autor (2018).

Tabela 5 – Resultados do teste *Tukey* das heurísticas GL (L = 1, 5, 10, 15 e 20) e PA20

					Intervalo de Confiança 95%	
(I) Heurísticas	(J) Heurísticas	Diferença média (I-J)	Erro Padrão	Significância	Limite inferior	Limite superior
G1	G10	13,82415*	0,18792	0,000	13,2883	14,3600
	G15	14,11958*	0,18792	0,000	13,5837	14,6554
	G20	14,38768*	0,18792	0,000	13,8518	14,9235
	G5	12,38966*	0,18792	0,000	11,8538	12,9255
	PA20	12,15543*	0,18792	0,000	11,6196	12,6913
G5	G1	-12,38966*	0,18792	0,000	-12,9255	-11,8538
	G10	1,43449*	0,18792	0,000	0,8986	1,9704
	G15	1,72992*	0,18792	0,000	1,1941	2,2658
	G20	1,99802*	0,18792	0,000	1,4622	2,5339
	PA20	-0,23423	0,18792	0,814	-0,7701	0,3016
G10	G1	-13,82415*	0,18792	0,000	-14,3600	-13,2883
	G15	0,29543	0,18792	0,617	-0,2404	0,8313
	G20	0,56353*	0,18792	0,033	0,0277	1,0994
	G5	-1,43449*	0,18792	0,000	-1,9704	-0,8986
	PA20	-1,66872*	0,18792	0,000	-2,2046	-1,1329
G15	G1	-14,11958*	0,18792	0,000	-14,6554	-13,5837
	G10	-0,29543	0,18792	0,617	-0,8313	0,2404
	G20	0,26810	0,18792	0,711	-0,2678	0,8040
	G5	-1,72992*	0,18792	0,000	-2,2658	-1,1941
	PA20	-1,96415*	0,18792	0,000	-2,5000	-1,4283
G20	G1	-14,38768*	0,18792	0,000	-14,9235	-13,8518
	G10	-0,56353*	0,18792	0,033	-1,0994	-0,0277
	G15	-0,26810	0,18792	0,711	-0,8040	0,2678
	G5	-1,99802*	0,18792	0,000	-2,5339	-1,4622
	PA20	-2,23225*	0,18792	0,000	-2,7681	-1,6964
PA20	G1	-12,15543*	0,18792	0,000	-12,6913	-11,6196
	G10	1,66872*	0,18792	0,000	1,1329	2,2046
	G15	1,96415*	0,18792	0,000	1,4283	2,5000
	G20	2,23225*	0,18792	0,000	1,6964	2,7681
	G5	0,23423	0,18792	0,814	-0,3016	0,7701

* A diferença média é significativa no nível 0,05.

Fonte: O Autor (2018).

Na Tabela 6, podemos identificar que o teste de hipótese de *Tukey*, ao nível de 95% de confiança, encontrou 4 subconjuntos de médias. A heurística PA20, de acordo com os dados, se equipara com a versão G5. Já G15 encontra-se entre os subconjuntos 1 e 2, onde estão contidos G20 e G10, respectivamente.

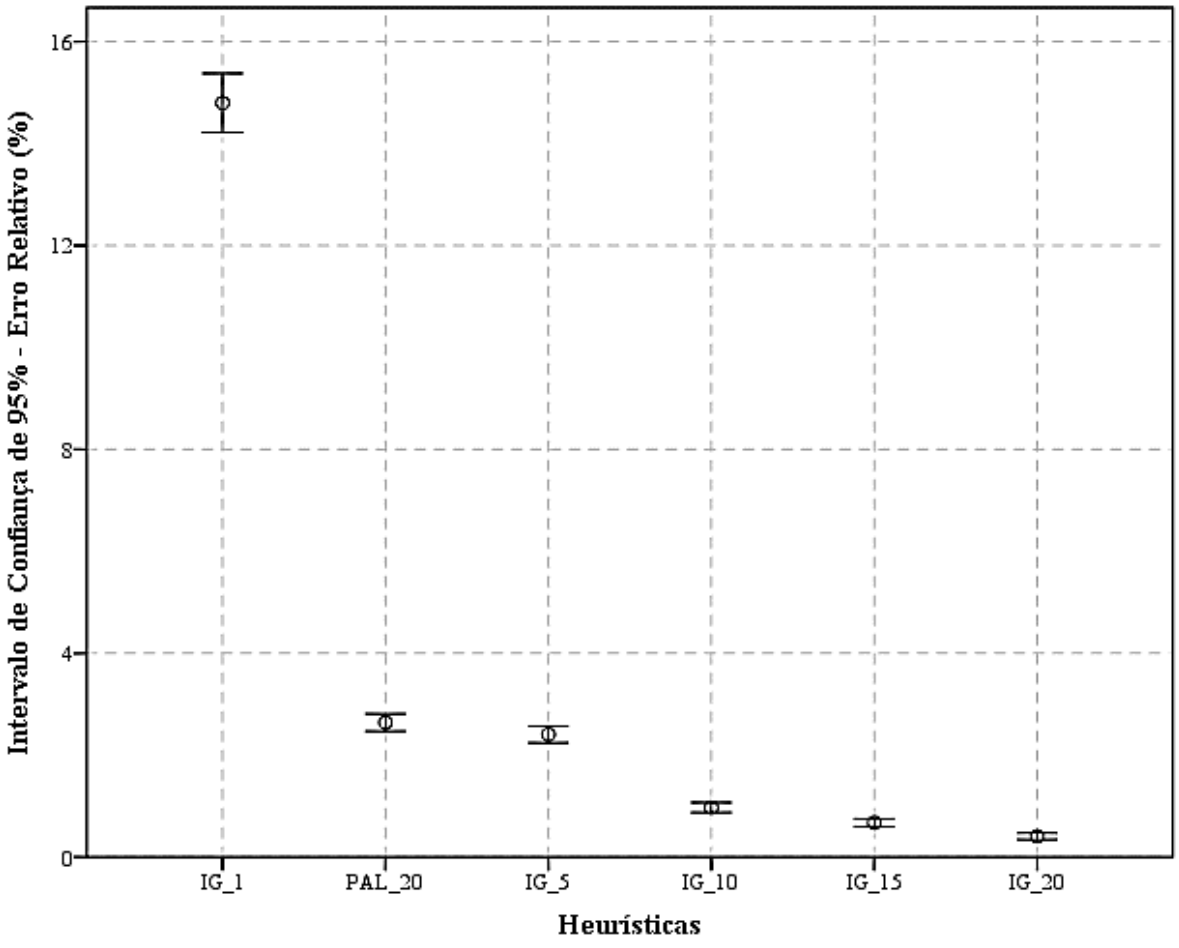
Tabela 6 – Identificação de subconjuntos homogêneos ao nível de 95% de confiança, de acordo com teste de *Tukey*.

Heurísticas	N	Subconjunto para alfa = 0,05			
		1	2	3	4
G20	500	0,4087			
G15	500	0,6768	0,6768		
G10	500		0,9722		
G5	500			2,4067	
PA20	500			2,6409	
G1	500				14,7963
Significância		0,711	0,617	0,814	1,000

São exibidas as médias para os grupos em subconjuntos homogêneos.

Fonte: O Autor (2018).]

Figura 12 – Gráfico de médias, ao nível de 95% de confiança, do desvio relativo médio geral das heurísticas avaliadas



Fonte: O Autor (2018).

A Figura 12 ilustra o desvio relativo médio geral e a variabilidade das heurísticas. Nota-se que G20 possui a menor média e menor variância dos desvios. Além disso, a diferença das heurísticas G20, G15 e G10 em relação a PA20 fica evidente. Verifica-se também a proximidade de PA20 e G5. O comportamento da performance de GL em relação ao número de iterações é semelhante ao experimento anterior.

4 CONCLUSÃO

Neste trabalho, foi abordado o problema de programação *no-wait flowshop* com dois objetivos: (1) minimizar o *makespan* sujeito à restrição de que o tempo médio de fluxo é menor ou igual a um dado valor; e (2) minimizar o tempo total de fluxo sujeito à restrição de que o *makespan* é menor ou igual a um dado valor. Foi apresentado o Algoritmo-M e o Algoritmo-K, usados na obtenção das soluções iniciais e restrições das heurísticas para os experimentos computacionais.

Para o objetivo (1), foi avaliada a heurística HH1 e cinco versões da heurística proposta GL (G1, G5, G10, G15 e G20). Os erros relativos médios gerais, sobre o número de tarefas e máquinas, de HH1, G5, G10, G15 e G20 foram 0,67; 9,50; 1,80; 0,90; 0;68; e 0,47; respectivamente. Embora os erros de HH1, G10, G15 e G20 sejam equiparáveis, o tempo computacional de HH1 foi significativamente maior. Isso evidencia a superioridade da heurística GL. Para o objetivo (2), foi avaliada a heurística PA20 (PAL com parâmetro L = 20) e novamente cinco versões da heurística proposta GL (G1, G5, G10, G15 e G20). Os erros relativos médios gerais de PA20, G5, G10, G15 e G20 foram 2,64; 14,80; 2,41; 0,97; 0,68; e 0,41; respectivamente. Esses dados evidenciam a superioridade de G10, G15 e G20. E embora o erro de PA20 e G5 sejam equiparáveis, o tempo computacional de PA20 foi significativamente maior. Isso evidencia a superioridade de GL também para o objetivo (2).

É importante destacar que dos três parâmetros de entrada da heurística GL (d, T e L), somente variações de L foram testadas. Mesmo tendo definido para os outros dois parâmetros os melhores valores para um problema de programação *flowshop* regular (RUIZ; STÜTZLE, 2007), é possível que esses valores não sejam os melhores para o caso *no-wait*. Portanto, em uma abordagem futura, poder-se-ia explorar variações desses parâmetros com o objetivo de verificar se ainda existe a possibilidade de ganho de performance.

Implementar as heurísticas com diferentes variações no número de tarefas e máquinas também é outra possibilidade a ser explorada. Nos experimentos computacionais apresentados aqui, foram testados valores baixos para o número de tarefas e máquinas. Mesmo que aumentar esses valores signifique lidar com um custo computacional elevado, isso seria validado para verificar se as heurísticas apresentariam mudança de comportamento.

Outra extensão sugerida é considerar os tempos de *setup* e manutenção. Neste trabalho, esses tempos foram ignorados ou assumidos como inclusos nos tempos de processamento. No entanto, essa abordagem pode não ser adequada para alguns ambientes de processamento, sendo necessário tratar esses dados de forma separada.

REFERÊNCIAS

ALDOWAISAN, T.; ALLAHVERDI, A. New heuristics for no-wait flowshops to minimize makespan. **Computers and Operations Research**, v. 30, n. 8, p. 1219–1231, 2003.

ALDOWAISAN, T.; ALLAHVERDI, A. New heuristics for m-machine no-wait flowshop to minimize total completion time. **Omega**, v. 32, n. 5, p. 345–352, 2004.

ALLAHVERDI, A. **A survey of scheduling problems with no-wait in processEuropean Journal of Operational Research**, 2016.

ALLAHVERDI, A.; ALDOWAISAN, T. No-wait flowshops with bicriteria of makespan and total completion time. **Journal of the Operational Research Society**, v. 53, n. 9, p. 1004–1015, 21 set. 2002.

ALLAHVERDI, A.; AYDILEK, H. Algorithms for no-wait flowshops with total completion time subject to makespan. **The International Journal of Advanced Manufacturing Technology**, v. 68, n. 9–12, p. 2237–2251, 24 out. 2013.

ALLAHVERDI, A.; AYDILEK, H.; AYDILEK, A. No-wait flowshop scheduling problem with two criteria; total tardiness and makespan. **European Journal of Operational Research**, v. 269, n. 2, p. 590–601, set. 2018.

AYDILEK, H.; ALLAHVERDI, A. Heuristics for no-wait flowshops with makespan subject to mean completion time. **Applied Mathematics and Computation**, v. 219, n. 1, p. 351–359, 2012.

BERTOLISSI, E. Heuristic algorithm for scheduling in the no-wait flow-shop. **Journal of Materials Processing Technology**, v. 107, n. 1–3, p. 459–465, nov. 2000.

BONNEY, M. C.; GUNDRY, S. W. Solutions to the Constrained Flowshop Sequencing Problem. **Operational Research Quarterly (1970-1977)**, v. 27, n. 4, p. 869, 1976.

CHE, A.; CHU, C. Cyclic hoist scheduling in large real-life electroplating lines. **OR Spectrum**, v. 29, n. 3, p. 445–470, 2007.

CHEN, C. L.; NEPPALLI, R. V.; ALJABER, N. Genetic algorithms applied to the continuous flow shop problem. **Computers and Industrial Engineering**, v. 30, n. 4, p. 919–929, 1996.

CHIEN, C. F. et al. **Modeling and analysis of semiconductor manufacturing in a shrinking world: Challenges and successes**. Proceedings - Winter Simulation Conference. **Anais**...2008

FINK, A.; VOSS, S. Solving the continuous flow-shop scheduling problem by metaheuristics. **European Journal of Operational Research**, v. 151, n. 2, p. 400–414, 2003.

FRAMINAN, J. M.; LEISTEN, R. A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. **Production**, v. 99, p. 28–40, 2006.

FRAMINAN, J. M.; NAGANO, M. S. Evaluating the performance for makespan minimisation in no-wait flowshop sequencing. **Journal of Materials Processing Technology**, v. 197, n. 1–3, p. 1–9, 2008.

FRAMINAN, J. M.; NAGANO, M. S.; MOCCELLIN, J. V. An efficient heuristic for total flowtime minimisation in no-wait flowshops. **International Journal of Advanced Manufacturing Technology**, v. 46, n. 9–12, p. 1049–1057, 2010.

GANGADHARAN, R.; RAJENDRAN, C. Heuristic algorithms for scheduling in the no-wait flowshop. **International Journal of Production Economics**, v. 32, n. 3, p. 285–290, 1993.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The Complexity of Flowshop and Jobshop Scheduling. **Mathematics of Operations Research**, 1976.

GRABOWSKI, J.; PEMPERA, J. Some local search algorithms for no-wait flow-shop problem with makespan criterion. **Computers and Operations Research**, v. 32, n. 8, p. 2197–2212, 2005.

GUPTA, J. N. D.; STAFFORD, E. F. Flowshop scheduling research after five decades. 2005.

HALL, N. G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. **Operations Research**, v. 44(3), p. 510(16), 1996.

JOHNSON, S. M. Optimal two- and three-stage production schedule with setup times included. **Nav Res Logist**, v. 1, p. 61–68, 1954.

KING, J. R.; SPACHIS, A. S. Heuristics for flow-shop scheduling. **International Journal of Production Research**, v. 18, n. 3, p. 345–357, 25 maio 1980.

NAGANO, M. S.; LORENA, L. A. N.; SILVA, A. A. A new evolutionary clustering search for a no-wait flow shop problem with set-up times. **Engineering Applications of Artificial Intelligence**, v. 25, n. 6, p. 1114–1120, 2012.

NAGANO, M. S.; MIYATA, H. H. Review and classification of constructive heuristics mechanisms for no-wait flow shop problem. **International Journal of Advanced Manufacturing Technology**, v. 86, n. 5–8, p. 2161–2174, 2016a.

NAGANO, M. S.; MIYATA, H. H. A High Quality Solution Constructive Heuristic for No-Wait Flow Shop Scheduling Problem. **Industrial Engineering & Management Systems**, v. 15, n. 3, p. 206–214, 2016b.

NAGANO, M. S.; MIYATA, H. H.; ARAÚJO, D. C. A constructive heuristic for total flowtime minimization in a no-wait flowshop with sequence-dependent setup times. **Journal of Manufacturing Systems**, v. 36, p. 224–230, 1 jul. 2015.

NAWAZ, M.; ENSCORE, E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, v. 11, n. 1, p. 91–95, 1 jan. 1983.

POSNER, N. G. H. ; M. E. **Generating experimental data for.pdfOperations Research**, 2001.

QIAN, B. et al. A DE-based approach to no-wait flow-shop scheduling. **Computers & Industrial Engineering**, v. 57, n. 3, p. 787–805, 2009.

RAJENDRAN, C. A no-wait flowshop scheduling heuristic to minimize makespan. **Journal of the Operational Research Society**, v. 45, n. 4, p. 472–478, 20 abr. 1994.

RAJENDRAN, C.; CHAUDHURI, D. Heuristic algorithms for continuous flow-shop problem. **Naval Research Logistics (NRL)**, v. 37, n. 5, p. 695–705, out. 1990.

RITZO, C. H. C. et al. Experiences in implementing simulation-based support for operational decision making in semiconductor manufacturing. **European J. of Industrial Engineering**, v. 5, n. 3, p. 272, 2011.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, v. 177, n. 3, p. 2033–2049, 16 mar. 2007.

SCHUSTER, C. J.; FRAMINAN, J. M. Approximative procedures for no-wait job shop scheduling. **Operations Research Letters**, v. 31, n. 4, p. 308–318, 2003.

SHYU, S. J.; LIN, B. M. T.; YIN, P. Y. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. **Computers and Industrial Engineering**, v. 47, n. 2–3, p. 181–193, 2004.

TSENG, L. Y.; LIN, Y. T. A Hybrid Genetic Algorithm for Flowshop Scheduling. **International Journal of Production Economics**, v. 128, p. 144–152, 2010.

VAN DEMAN, J. M.; BAKER, K. R. Minimizing Mean Flowtime in the Flow Shop with No Intermediate Queues. **A I I E Transactions**, v. 6, n. 1, p. 28–34, mar. 1974.

YANG, X.-S. **Engineering optimization : An introduction with metaheuristic applications**. [s.l.] John Wiley & Sons, 2010.

ZHU, J.; LI, X.; WANG, Q. Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. **European Journal of Operational Research**, v. 198, n. 2, p. 378–386, 2009.