UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

DANIELLA CASTRO ARAÚJO

Novos métodos heurísticos para a programação em *no-wait flowshop* com tempos de *setup* dependentes da sequência

São Carlos

2010

Daniella Castro Araújo

Novos métodos heurísticos para a programação em *no-wait flowshop* com tempos de *setup* dependentes da sequência

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia de Produção.

Área de concentração: Processos e Gestão de Operações.
Orientador: Prof. Dr. Marcelo Seido Nagano

São Carlos
2010

"Noites de insônia
Cãs prematuras
Prantos convulsos
Meu Deus, salvai-o!
Filhos são o demo
Melhor não tê-los...
Mas se não os temos
Como sabê-los?"

Vinícius de Moraes

Aos meus pais,
José Carlos e Arlete.

AGRADECIMENTOS

Agradeço primeiramente a quem também dedico este trabalho: àqueles que me permitiram realizar este e todos os outros projetos de minha vida – meus pais.

Ao Prof. Dr. Marcelo Seido Nagano, toda a minha gratidão pela paciência, bom humor e compreensão.

A todos os meus amigos, de Goiânia, de São Carlos e do Rio De Janeiro. Foram, cada um de sua maneira, muito importantes para o desempenho deste trabalho. Em especial, agradeço ao Francisco e ao Sérgio, por todo o apoio.

Agradeço também a todos os professores e funcionários da Escola de Engenharia de São Carlos, principalmente aos do Departamento de Engenharia de Produção, e à FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo), pela bolsa concedida, sem os quais não teria sido possível a conclusão desta pesquisa.

RESUMO

ARAÚJO, D. C. Novos métodos heurísticos para a programação em *no-wait flowshop* com tempos de *setup* dependentes da sequência. 207f. 2010. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010.

Este trabalho aborda o problema de programação em um ambiente *no-wait flowshop* com tempos de *setup* dependentes da sequência. Aplicações para esse problema são comuns na indústria farmacêutica e de química fina. São considerados dois critérios de minimização: o tempo total de fluxo (*total flowtime*) e a duração total da programação (*makespan*). Dois métodos heurísticos são propostos e comparados aos métodos de Bianco, Dell´Olmo e Giordani (1999) e Brown, McGarvey e Ventura (2004), através do banco de dados de Ruiz, Maroto e Alcaraz (2005). Os métodos são avaliados em relação à porcentagem de sucesso, ao desvio relativo médio e ao tempo médio de computação. Os resultados da experimentação computacional mostram a superioridade dos métodos propostos.

Palavras-chave: Programação da Produção, *No-wait Flowshop*, Tempos de *Setup* Dependentes da Sequência, Heurísticas, *Makespan*, *Flowtime*.

ABSTRACT

ARAÚJO, D. C. New heuristics for the no-wait flowshop with sequence-dependent setup times. 185f. 2010. Dissertation (M.Sc.) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010.

This work addresses the problem of scheduling jobs in a no-wait flowshop problem with sequence-dependent setup times. The motivation for this study comes from the production of several components in fine chemical and pharmaceutical manufacturing plants. The performance measures of the production schedules are both makespan and total flowtime. Two heuristics are proposed and compared to Bianco, Dell´Olmo e Giordani (1999) and Brown, McGarvey e Ventura (2004) heuristics. The heuristics are evaluated considering the success rate, the average relative deviation and the computional effort. The results show the superiority of the proposed methods.

Key-words: Scheduling, No-wait Flowshop, Sequence-Dependent Setup Times, Heuristics, Makespan, Flowtime.

Lista de Símbolos

C_{max}	Duração total da programação (*makespan*)
$\sum C_j$	Soma das datas de término (*total completion time*)
DR_h	Desvio relativo
$F2$	*Flowshop* com duas máquinas
FF	*Flowshop* flexível
F_m	*Flowshop* com m-máquinas
$\overline{F}$	Tempo médio de fluxo (*mean flowtime*)
$\sum F_j$	Tempo de fluxo total (*total flowtime*)
h	Posição relativa de inserção de uma tarefa na sequência
$j_{[i]}$	Tarefa que ocupa a i-ésima posição na sequência
J	Conjunto das tarefas a serem programadas, $J = \{j_1, .., j_n\}$
L_{max}	Atraso ou adiantamento máximo (*maximum lateness*)
m	número de máquinas do problema
M	Conjunto das máquinas a serem programadas, $M = \{m_1, .., m_m\}$
n	número de tarefas do problema
op_{ki}	Operação da tarefa i na máquina k
p_{ki}	Tempo de processamento da operação op_{ki}
$prec$	Restrições de precedência
r_j	Datas de liberação
R_{si}	Tempos de remoção
ST_{sd}	Tempos de *setup* dependentes da sequência
ST_{si}	Tempos de *setup* independentes da sequência

s_{ij}^{k}	Tempo de *setup* (dependente da sequência) da tarefa *i* para a tarefa *j* na máquina *k*
T_{max}	Atraso máximo (*maximum tardiness*)
$\sum T_j$	Atraso total
U	Conjunto das tarefas não-programadas
$\sum w_j C_j$	Soma ponderada das datas de término
$\sum w_j F_j$	Soma ponderada dos tempos de fluxo
$\sum w_j T_j$	Soma ponderada dos atrasos
σ	Conjunto das tarefas programadas

Lista de Siglas

ACO	*Ant Colony Optimization*
ATSP	*Assymetric Traveling Salesman Problem*
ATSP-RT	*Asymmetric Travelling Salesman Problem with Ready Times*
DPSO	*Discrete Particle Swarm Otimization*
FITSP	*Farthest Insertion Traveling Salesman Problem*
FMS	*Flexible Manufacturing Systems*
GRASP	*Greedy Randomized Search Procedure*
ILS	*Iterated Local Search*
JIT	*Just-In-Time*
LPT	*Longest Processing Lime*
MILP	*Mixed-Integer Linear Programming*
MIP	*Mixed Integer Programming*
NWFS	*No-wait Flowshop*
OPT	O*ptimized Production Technology*
RIPS	*Radom Insertion Pertubation Scheme*
SDST	*Sequence Dependent Setup Times*
SMED	*Single Minute Exchange of Die*
SPT	*Shortest Processing Time*
TSP	*Traveling Salesman Problem*
WIP	*Work In Process*
WWW	*World Wide Web*

Lista de Figuras

Lista de Gráficos

Lista de Tabelas

Sumário

CAPÍTULO 1

INTRODUÇÃO

Um sistema de produção pode ser definido como um processo, ou um conjunto de processos, pelo qual os produtos e serviços são gerados. Em outras palavras, sistema de produção é a transformação de insumos em produtos ou serviços, através de processos e operações (MONKS, 1987). De acordo com Slack et al. (1999), a produção deve ser o foco central para uma organização, porque é responsável por produzir bens e serviços que são a razão de sua existência.

Segundo Johnson e Montgomery (1974), os sistemas de produção podem ser classificados em:

a) Sistema Contínuo, onde poucos tipos de produtos similares são fabricados em grande volume;

b) Sistema Intermitente, onde ocorrem freqüentes mudanças de um produto para outro nos estágios produtivos, como conseqüência de uma grande variedade de produtos fabricados. Nesta classe distinguem-se dois tipos de sistemas:

- *Flowshop*, onde os itens fabricados em uma linha ou célula de manufatura têm a mesma seqüência de operações nas diversas máquinas;
- *Jobshop*, onde os itens fabricados em um setor produtivo não têm o mesmo roteiro de fabricação;

c) Sistema Grande Projeto, onde são produzidos itens complexos e especiais, muitas vezes únicos.

Dentro do Sistema Intermitente, um importante nível de decisão é a programação de operações, que oferece oportunidade a grandes ganhos em tempo e aproveitamento da capacidade instalada. De acordo com MacCarthy e Liu (1993) e Yang e Liao (1999), a programação de operações pode ser definida como a alocação de recursos através do tempo para a realização de tarefas, para melhor satisfazer um conjunto de critérios pré-definidos.

1.1 Programação de Operações

De acordo com Pinedo (2008), a programação da produção ou *scheduling* refere-se à ordenação de tarefas a serem executadas, em uma ou diversas máquinas, considerando-se uma base de tempo, ou seja, determinando-se, principalmente, as datas de início e fim de cada tarefa.

Os problemas de programação de operações podem ser descritos com a notação de três parâmetros: α/β/γ. O parâmetro α descreve o ambiente das máquinas, o parâmetro β contém as características do processo e o parâmetro γ possui a função objetivo, ou seja, o critério de avaliação da programação (PINEDO, 1995).

A maioria dos problemas reais de programação da produção é muito complexa e de difícil solução. Além disso, cada ambiente de fabricação possui suas particularidades, apresentando diferentes tipos de problemas.

Para Maccarthy e Liu (1993), um problema de programação é

especificado em termos das restrições tecnológicas do ambiente de produção onde as tarefas devem ser realizadas e dos objetivos de programação. Essas restrições são determinadas principalmente pelo padrão de fluxo das tarefas nas máquinas. Dessa maneira, os problemas de programação podem ser classificados conforme segue:

a) *Jobshop*: cada tarefa tem sua própria ordem de processamento nas máquinas;

b) *Flowshop*: todas as tarefas têm o mesmo fluxo de processamento nas máquinas;

c) *Openshop*: não há fluxo padrão especificado para nenhuma tarefa e cada estágio de produção possui apenas uma máquina;

d) *Flowshop* Permutacional: um *flowshop* em que a ordem de processamento das tarefas em todas as máquinas é a mesma;

e) Máquina Única: há apenas um estágio de produção com uma única máquina disponível;

f) Máquinas Paralelas: há mais de uma máquina disponível em um único estágio de produção, onde cada tarefa necessita de apenas uma dessas máquinas;

g) *Jobshop* com Múltiplas Máquinas: *jobshop* no qual existe um conjunto de máquinas paralelas em cada estágio de produção;

h) *Flowshop* com Múltiplas Máquinas: *flowshop* no qual existe um conjunto de máquinas paralelas em cada estágio de produção.

A Figura 1 a seguir ilustra a relação entre os diferentes problemas de programação.

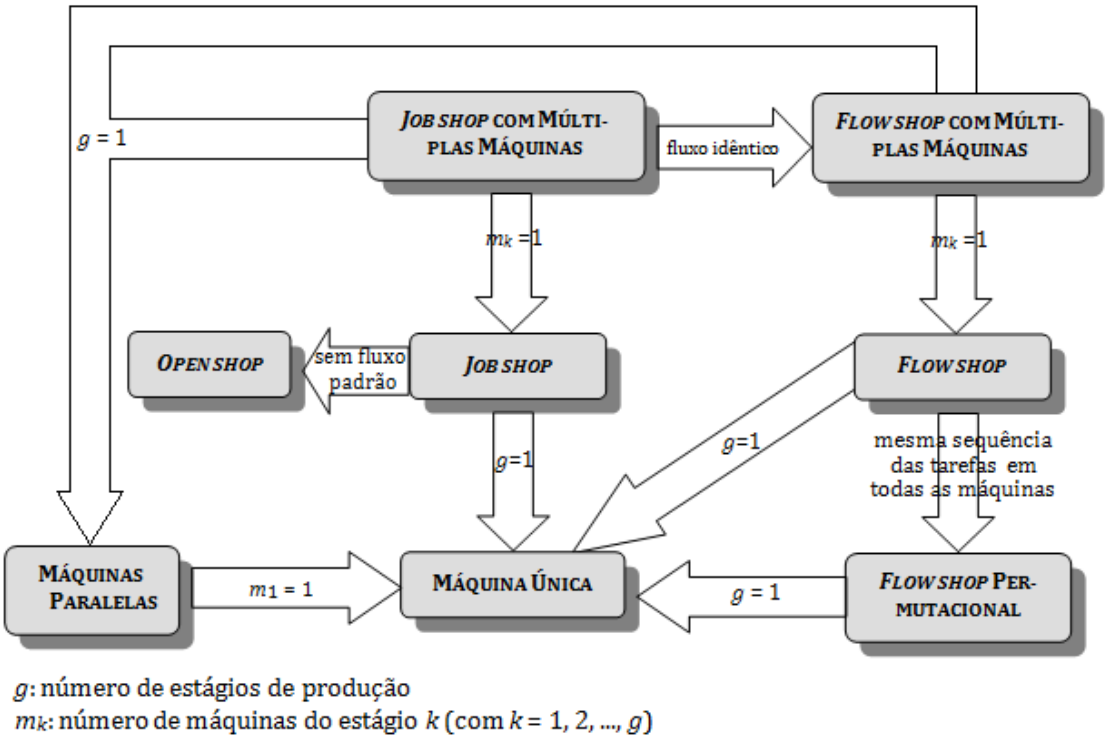

Figura 1 – Relação entre as classes de problemas (MACCARTHY; LIU, 1993)

Neste trabalho, será abordado o problema em ambiente *flowshop* permutacional com duas condições adicionais: não há estocagem entre as máquinas (*no-wait flowshop*) e há tempos de *setup* dependentes da sequência. O objetivo é minimizar o tempo total de fluxo (*total flowtime*) e a duração total da programação (*makespan*). Um problema geral de programação em ambiente *flowshop* pode ser entendido, conforme Taillard (1993), como um conjunto J= $\{j_1, j_2, j_3, \dots, j_n\}$ de n tarefas que devem ser processadas em um conjunto $M = \{m_1, m_2, m_3, \dots, m_m\}$ de m máquinas disponíveis. O processamento de uma tarefa j_i em uma máquina m_k é denominado uma operação, designado por op_{ki}. Para cada operação op_{ki} existe um tempo de processamento p_{ki} associado. Existem problemas (geralmente de pequeno porte) em que se consegue obter a solução

ótima através de programação linear inteira. Entretanto, a grande maioria dos problemas de programação da produção é muito difícil de ser resolvida. Segundo Yang e Liao (1999), tais problemas podem ser classificados em duas classes distintas: $\mathcal{P}$ e $\mathcal{NP}$. $\mathcal{P}$ denota a classe de problemas que são eficientemente resolvidos em tempo polinomial, e $\mathcal{NP}$ a classe cujos problemas não possuem métodos de obtenção da solução ótima em tempo polinomial.

1.2 Métodos de Solução

Muitos métodos têm sido desenvolvidos na tentativa de solucionar os problemas de programação. Esses métodos (também chamados de algoritmos) podem basicamente ser de dois tipos:

- Métodos de solução exata; e
- Métodos heurísticos.

Apesar dos problemas de programação *flowshop* com mais de três máquinas pertecerem à classe $\mathcal{NP}$ (GAREY; JOHNSON, 1979 e RICKEL, 1988), vários autores desenvolvem métodos de solução exata para eles. Existe uma razão principal para a formulação de um modelo matemático: ela permite um melhor entendimento do problema, o que pode ser útil no desenvolvimento de métodos heurísticos, limitantes inferiores e superiores e relações de dominância (SETHANAN, 2001). Vários autores, tais como Bianco, Dell´Olmo e Giordani (1999) e Brown, McGarvey e Ventura (2004) desenvolveram em um mesmo trabalho modelos matemáticos e heurísticas. Como exemplos de métodos de solução exata,

têm-se a programação matemática de Srikar e Ghosh (1986) e o método de enumeração *branch-and-bound* de Ríos-Mercado e Bard (1998a).

De acordo com MacCarthy e Liu (1993), o tempo despendido para encontrar a solução ótima é uma função polinomial das variáveis do problema (tarefas e máquinas). Isso faz com que estes métodos sejam apenas eficientes computacionalmente em problemas de pequeno porte. Por este motivo, muitos métodos heurísticos têm sido propostos para a solução de problemas de programação da produção.

Um método heurístico é um processo de solução de problema apoiado em critérios racionais ou computacionais para escolher um caminho entre vários possíveis, sem a preocupação de percorrer todas as possibilidades ou atingir a melhor solução. As heurísticas são aplicadas em problemas onde uma solução aproximada traz mais benefícios do que uma solução exata, em relação à redução do tempo necessário para obtê-la.

Os métodos heurísticos classificam-se de modo geral em métodos construtivos e meta-heurísticas. Os métodos construtivos se caracterizam pela geração de apenas uma solução. Como exemplos de métodos construtivos, têm-se o NEH de Nawaz, Enscore e Ham (1983), o N&M de Nagano e Moccellin (2002) e o método proposto por Framinan, Nagano e Moccellin (2010).

As meta-heurísticas, de forma geral, utilizam combinação de escolhas aleatórias e conhecimento histórico (dos resultados anteriores adquiridos pelo método) para se guiarem e realizarem suas buscas pelo espaço de pesquisa em vizinhanças dentro do espaço de pesquisa, o que evita paradas prematuras em ótimos locais. Alguns exemplos de meta-heurísticas são: algoritmo genético, (RUIZ;

MAROTO; ALCARAZ, 2005), busca tabu (FINK e VOβ, 2003), *simulated annealing* (ALDOWAISAN e ALLAHVERDI, 2003), *Greedy Randomized Search Procedure* (GRASP) (RÍOS-MERCADO e BARD, 1998b), dentre outros.

1.3 Objetivos e Estrutura do Trabalho

O objetivo da pesquisa proposta é o desenvolvimento de dois novos métodos heurísticos construtivos para o problema *no-wait flowshop* com tempos de *setup* dependentes da sequência, a fim de minimizar o *makespan* e o *total flowtime* da programação. Adjacente a este tema central proposto, são objetivos também do trabalho: revisão bibliográfica acerca dos temas *flowshop* com tempos de *setup* dependentes da sequência, *no-wait flowshop* básico (ou seja, sem tempos de *setup*) e *no-wait flowshop* com tempos de *setup* independentes e dependentes da sequência e a implementação dos principais métodos encontrados na literatura para o caso, para fins de comparação.

Com base nesses objetivos descritos, a dissertação está organizada em 10 capítulos:

- O Capítulo 2 é dedicado ao problema de *flowshop* com tempos de *setup*, apresentando uma revisão bibliográfica de *flowshop* com tempos de *setup* dependentes da sequência;
- O Capítulo 3 apresenta uma revisão bibliográfica sobre o problema *no-wait flowshop* básico;
- O Capítulo 4 apresenta uma revisão bibliográfica acerca do problema

no-wait flowshop com tempos de *setup* independentes e dependentes da sequência;

- O Capítulo 5 apresenta a notação e a definição do problema tratado – *no-wait flowshop* com tempos de *setup* dependentes da sequência;
- O Capítulo 6 detalha os métodos heurísticos existentes para o problema;
- O Capítulo 7 propõe os novos métodos heurísticos, detalhando os passos de cada método com exemplos numéricos;
- O Capítulo 8 detalha a experimentação computacional, realizada em duas partes, sendo uma para o critério de minimização do *makespan* e outra do *total flowtime*.
- O Capítulo 9 apresenta a análise dos resultados, também dividida em duas partes, com apresentação de gráficos e tabelas e conclusões acerca da porcentagem de sucesso, do desvio relativo médio e do tempo médio de computação dos métodos avaliados.
- O Capítulo 10 apresenta as conclusões do trabalho.

1.4 Metodologia de Pesquisa

Como mencionado no tópico acima, um dos objetivos deste trabalho é a revisão bibliográfica. Para Silva e Menezes (2001), uma das etapas mais

importantes de um projeto de pesquisa é a revisão de literatura, já que ela dá sustentação ao desenvolvimento da pesquisa.

Essa revisão buscará a determinação do "estado da arte" do *no-wait flowshop* com tempos de *setup* dependentes da sequência, procurando mostrar através da literatura existente o que já se sabe sobre o tema e as lacunas existentes. Para tanto, serão buscados artigos científicos, livros, dissertações e teses acerca dos temas "*flowshop* com tempos de *setup* dependentes da sequência", "*no-wait flowshop* básico" *e "no-wait flowshop* com tempos de *setup* independentes e dependentes da sequência". Todos os trabalhos encontrados serão incluídos na revisão, para que se tenha uma real dimensão do interesse científico sobre o problema, dos métodos de solução e dos critérios mais utilizados e da evolução dentro do quadro teórico de refências. Ademais, uma extensa revisão bibliográfica auxilia na aprendizagem sobre a programação de operações e mesmo na criatividade para o desenvolvimento de métodos heurísticos, podendo também servir de base para pesquisas futuras.

A revisão bibliográfica, dentro de cada tema citado, será divida em três seções: métodos de solução exata, métodos heurísticos construtivos e meta-heurísticas. Cada uma dessas seções será resumida em uma tabela, e no final de cada capítulo haverá um pequeno fichamento, contendo considerações acerca dos trabalhos encontrados.

CAPÍTULO 2

FLOWSHOP COM TEMPOS DE *SETUP*

O problema de seqüenciamento em um sistema de produção *flowshop* tem interessado pesquisadores por quase meio século. Desde Johnson (1954), milhares de artigos com diferentes problemas de programação surgiram na literatura. A maioria desses artigos assume que o tempo/custo de *setup* é insignificante ou parte do tempo de processamento. Enquanto esse pressuposto simplifica a análise e reflete certas aplicações (quando os *setups* são pequenos, independentes da seqüência e inseparáveis dos tempos de processamento), ele adversamente afeta a qualidade da solução de várias outras (ALLAHVERDI et al., 2008, ALDOWAISAN, 2001, RAJENDRAN; ZIEGLER, 1997 e MOCCELLIN; NAGANO, 2007).

Por definição, tempo de *setup* é o tempo necessário na preparação de determinado recurso (máquinas, pessoas) para executar uma atividade (tarefa, operação). Custo de *setup* é o custo de um recurso utilizado para iniciar execução de uma atividade. As atividades de *setup* em um sistema de manufatura incluem a obtenção de ferramentas, posicionamento do material a ser trabalhado, devolução de ferramentas, limpeza, ajuste de peças e acessórios, ajuste de ferramentas e inspeção de material. Em uma organização de serviços, incluem a adequação do ambiente à execução de atividades, entre outros (ALLAHVERDI; SOROUSH, 2008).

De acordo com Allahverdi e Soroush (2008), a decisão de fabricar múltiplos produtos/serviços com recursos comuns resulta na necessidade de

trocas de ferramentas e de atividades de *setup*, que geram onerosas rupturas na produção. Assim, a redução do *setup* é uma importante característica para o programa de melhoria contínua de qualquer empresa, e é ainda mais importante para empresas que objetivam reduzir o *lead time* e o tamanho dos lotes e aumentar os padrões de qualidade.

Houve um grande aumento nas pesquisas envolvendo tempos/custos de *setup* separáveis nos últimos dez anos (ALLAHVERDI et al., 2008). Segundo Allahverdi e Soroush (2008), esse aumento vem do fato de que há imensas economias quando as atividades de *setup* são explicitamente incorporadas nas decisões de programação da produção. Tratar os tempos/custos de *setup* separados dos tempos/custos de processamento permite que algumas operações sejam executadas simultaneamente, aumentando assim a utilização de recursos, o que é muito importante para os sistemas modernos de gestão da produção, como o *just-in-time* (JIT), a tecnologia de grupo, a tecnologia de produção otimizada (O*ptimized Production Technology* - OPT) e a manufatura celular.

A importância e as aplicações de modelos de sequenciamento com considerações explícitas de tempos de *setup* foram amplamente discutidas em vários estudos a partir de meados de 1960 (ALLAHVERDI et al., 2008). A seguir estão algumas aplicações recentes:

- Laguna (1999) considerou uma instalação que produz cartuchos para fotocopiadoras e impressoras a *laser*. Ele destacou o fato de que para mudar a produção de um *toner* para outro havia grandes tempos de *setup*;

- Schaller, Gupta e Vakharia (2000) abordaram o problema de fabricação de placas de circuito em uma máquina automática. Eles mostraram que o problema possui tempos de *setup*;

- Na indústria têxtil, tempos de *setup* são significativos e devem ser considerados separadamente dos tempos de processamento, como mencionado em Gendreau, Laporte e Guimarães (2001). Quando o tipo de tecido é mudado em uma máquina, o rolo de tecido deve ser trocado e o tempo para isso depende tanto do atual como do próximo tipo de tecido;

- Várias aplicações *World Wide Web* (WWW) necessitam de acesso, transferência e sincronização de dados de multimídia, como áudio, vídeo e imagens, através de uma rede de comunicação. O processamento e transferência de grandes dados através da internet afetam o tempo de resposta dos utilizadores finais. Assim, a programação dos dados de multimídia é um aspecto crítico para sua distribuição. Allahverdi e Al-Anzi (2002) mostraram que o problema de dados de multimídia para a aplicação WWW pode ser modelada como um *flowshop* com duas máquinas e tempos separáveis de *setup*;

- Kim et al. (2002) consideraram a produção de componentes semicondutores. Eles mostraram que as máquinas utilizadas na sua produção devem ser ajustadas a cada novo corte de pastilhas. Assim, diferentes tempos de *setup* são necessários, dependendo da sequência das pastilhas;

- Chang, Hsieh e Wang (2003) descreveram a rotina de uma indústria de filmes de polipropileno, que produzia fitas adesivas, álbuns de foto, capas de livro, etc. Eles mostraram que o tempo, matéria-prima e equipamentos necessários para preparar o próximo produto dependiam do produto anterior;

- Lin e Liao (2003) estudaram uma indústria de etiquetas, composta por um *flowshop* híbrido de dois estágios onde o primeiro estágio possuía uma máquina de alta velocidade que era usada para passar cola no material da superfície e forrá-lo. Eles notaram que, quando a máquina do primeiro estágio tinha a classe de material mudada, havia um *setup* dependente da sequência.

Existem dois tipos de *setup*: dependente e independente da sequência. Se o tempo/custo de *setup* depende somente da tarefa a ser processada, independentemente da tarefa anterior, é chamado de independente da sequência. Já o *setup* dependente da sequência depende tanto da tarefa a ser processada quanto da tarefa anterior (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999).

Neste trabalho, será estudado o *setup* dependente da sequência. A seguir, no Tópico 2.1, detalha-se este problema, e nos Tópicos 2.1.1, 2.1.2 e 2.1.3 são realizadas revisões bibliográficas com métodos de solução exata, métodos heurísticos e métodos meta-heurísticos, respectivamente.

2.1 *Flowshop* com Tempos de *Setup* Dependentes da Sequência

Uma variação interessante e realista do problema de *flowshop* permutacional é a que ocorre quando as tarefas têm tempos de *setup* dependentes da sequência nas máquinas (*Sequence Dependent Setup Times* - SDST), ou seja, em cada máquina há um tempo de *setup* significativo para a próxima tarefa, que depende da tarefa que está sendo processada. Um exemplo claro desse tipo de

problema vem da indústria de corte de papel, onde as máquinas de corte precisam ser ajustadas a cada novo lote de papel. Outro exemplo pode ser obtido do setor cerâmico: azulejos de cerâmica são produzidos em linhas de processamento idênticas, compostas por vários processos, como moldagem, secagem, pintura, forno, controle de qualidade e embalagem. Se determinada linha estiver preparada para determinado produto, por exemplo, azulejo preto de 33x33cm, mudar a linha para a produção de azulejo branco de 45x45cm requererá tempo de *setup* considerável para trocar os moldes na moldagem e para limpar e preparar a pintura para a cor branca. No entanto, mudar a linha para a produção de azulejo amarelo de 33x33cm requererá apenas a limpeza e preparação da pintura, já que a moldagem estará pronta (RUIZ; MAROTO; ALCARAZ, 2005).

A importância dos tempos de *setup* dependentes da sequência foi investigada em vários estudos. O trabalho de Wilbrecht e Prescott (1969) indicou que os tempos de *setup* dependentes da sequência são significativos quando o chão de fábrica está operando próximo da sua capacidade máxima. Uma pesquisa em indústrias realizada por Panwalkar, Dudek e Smith (1973) mostrou que aproximadamente 75% das indústrias têm pelo menos uma tarefa que possui tempo de *setup* dependente da sequência e que em 15% das indústrias todas as tarefas possuem tempos de *setup* dependentes da sequência. Flynn (1987) mostrou que o uso conjunto de procedimentos para minimização dos tempos de *setup* dependentes da sequência e de princípios de tecnologia de grupo aumenta a capacidade de produção em arranjos celulares. Wortman (1992) apresentou a importância de se considerar tempos de *setup* dependentes da sequência para uma gestão efetiva de controle da capacidade. Krajewski et al. (1987) examinaram os fatores que têm maior influência no desempenho de uma fábrica e concluíram que,

independente do sistema de produção utilizado, a redução simultânea dos tempos de *setup* e dos tamanhos de lote é a principal maneira de se reduzir o estoque e melhorar o serviço ao consumidor.

Tendo em vista a importância que os tempos/custos de *setup* exercem na eficiência da fábrica, há atualmente vários tipos de tecnologias que buscam reduzi-los, tais como os sistemas de manufatura flexíveis *(Flexible Manufacturing Systems* - FMS) e a troca rápida de ferramentas (*Single Minute Exchange of Die* - SMED). Apesar delas, os procedimentos de *setup* continuam sendo significativos no tempo e custo de produção da maioria das fábricas.

Gupta (1986) demonstrou que o problema de *flowshop* com tempos de *setup* dependentes da sequência é $\mathcal{NP}$-completo. Até 1990, muito pouco foi publicado sobre esse problema complexo. A partir de 1995, a literatura envolvendo o *flowshop* SDST cresceu significativamente (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999; CHENG; GUPTA; WANG, 2000; TSENG; STAFFORD, 2001; ALLAHVERDI et al., 2008).

2.1.1 Pesquisas com Métodos de Solução Exata

Corwin e Esogbue (1974) consideraram o problema de programação da produção com duas máquinas com o objetivo de minimizar o *makespan,* em que apenas uma das máquinas (primeira ou segunda) possuía tempo de *setup* dependente da sequência, compondo assim dois diferentes problemas. Eles

resolveram os problemas utilizando uma formulação de programação dinâmica que, do ponto de vista computacional, é comparável ao problema do caixeiro viajante (*Traveling Salesman Problem* - TSP).

Uskup e Smith (1975) desenvolveram um algoritmo *branch-and-bound* para o problema com duas máquinas no qual as tarefas possuíam datas de entrega e tempos de *setup* antecipatórios e dependentes da sequência. A solução ótima encontrada pelo algoritmo minimiza o custo total do *setup* (supondo-se que o custo de *setup* seja proporcional ao tempo de *setup*), além de respeitar as datas de entrega das tarefas.

Gupta (1975) apresentou um algoritmo de busca lexicográfica para solucionar o problema F_m/ST_{sd} com o objetivo de minimizar o custo total de oportunidade, custo esse que envolve o *setup* dependente da sequência. O algoritmo proposto especifica uma relação linear lexicográfica e gera sistematicamente sequenciamentos até que todas as possíveis soluções sejam examinadas e a solução seja encontrada. Os resultados computacionais mostraram que o algoritmo é mais eficiente do que um procedimento de completa enumeração, já que a sequência ótima pode ser reconhecida em uma das fases iniciais da busca.

Srikar e Ghosh (1986) desenvolveram uma formulação de programação linear inteira-mista (*Mixed-Integer Linear Programming* – MILP) para o problema com *m*-máquinas para minimizar o *makespan* ou o *mean flowtime*. O modelo utiliza menor número de variáveis do que formulações baseadas no TSP, sendo, portanto, mais rápido, mas consegue resolver otimamente apenas problemas com até seis tarefas e seis máquinas.

Szwarc e Gupta (1987) mostraram matematicamente que o modelo de tempos de *setup* aditivos é um caso especial de tempo de *setup* dependente da sequência. Esse modelo divide o tempo de *setup* em dois componentes: um depende da tarefa em curso, e o outro da tarefa seguinte. Eles desenvolveram um algoritmo delimitado polinomialmente para minimizar o *makespan*, que encontra soluções ótimas para o caso de duas máquinas.

Gupta, Das e Ghosh (1995) desenvolveram um algoritmo *branch-and-bound* para a obtenção da solução ótima do problema $F_m/ST_{sd}/C_{max}$ envolvendo até 20 tarefas.

Ríos-Mercado e Bard (1996) basearam-se na formulação MILP do problema abordado por Srikar e Ghosh (1986) e desenvolveram várias inequações válidas para duas formulações: uma de programação linear inteira-mista (MILP) e uma de programação inteira-mista (*Mixed Integer Programming* – MIP) para o problema com *m*-máquinas, com o objetivo de minimizar o *makespan*.

Hwang e Sun (1997) estudaram uma linha de produção de uma indústria de caminhões com duas máquinas, onde todas as tarefas necessitavam de processamento na primeira máquina mais de uma vez e o tempo de *setup* de uma tarefa na primeira máquina dependia das duas tarefas anteriores. Devido à complexidade do problema, eles redefiniram os elementos das tarefas, e então o formularam como um problema geral de duas máquinas com restrições de precedência. Em seguida, apresentaram uma abordagem de programação dinâmica com o objetivo de minimização do *makespan*.

Sonmez e Baykasoglu (1998) desenvolveram uma formulação de programação dinâmica para o problema $F_m/ST_{sd}/\sum w_j T_j$, aplicada a uma fábrica de

tubos de plástico. Os testes computacionais mostraram que um aumento no número de tarefas aumenta muito o tempo computacional do programa, enquanto que o aumento no número de máquinas praticamente não afeta seu tempo.

Ríos-Mercado e Bard (1998a) abordaram o problema $F_m/ST_{sd}/C_{max}$ utilizando a programação MIP em duas formulações diferentes. A primeira formulação é baseada no problema do caixeiro viajante, enquanto que a segunda utiliza várias variáveis binárias e restrições, sendo menos estruturada do que a primeira. Eles então avaliaram empiricamente as formulações MIP dentro da estrutura *branch-and-cut* e mostraram que essa abordagem é superior ao algoritmo *branch-and-bound.* Os resultados computacionais mostraram ainda que a segunda formulação matemática é mais rápida do que a primeira.

Continuando a pesquisa, Ríos-Mercado e Bard (1999a) apresentaram um algoritmo *branch-and-bound* com limitantes inferiores e superiores. Os resultados computacionais mostraram que o algoritmo proposto é mais eficiente do que os algoritmos baseados em programação linear, já que ele busca sequências parciais, obtendo assim economias de tempo.

Sun e Hwang (2001) abordaram o problema $F2/ST_{sd}/C_{max}$, com tempos de *setup* apenas na segunda máquina. O tempo de *setup* dependia das k tarefas imediatamente precedentes, sendo $k>1$. Eles desenvolveram uma formulação de programação dinâmica e encontraram a solução ótima utilizando condições de dominância na sequência.

Sethanan (2001) estudou o problema em ambiente *flowshop* com m estágios com uma ou mais máquinas em cada estágio, conhecido como *flowshop* flexível. Em cada máquina, havia um *setup* dependente da sequência e o objetivo

era minimizar o *makespan*. Ele desenvolveu um modelo matemático para problemas pequenos, com até 5 tarefas, 5 estágios e 8 máquinas.

Tseng e Stafford (2001) apresentaram dois modelos MILP para o *flowshop* com *m*-máquinas para quatro funções objetivo: *makespan*, tempo médio de fluxo, tempo total ocioso das tarefas e tempo ocioso de máquina. O primeiro modelo, TS1, foi derivado diretamente do modelo de MILP de Stafford (1988) para *flowshops* clássicos e o segundo, TS2, combinou as propriedades do modelo TS1 com o modelo MILP SG*, de Srikar e Ghosh (1986), corrigido por Stafford e Tseng (1990), que resolve problemas de até seis tarefas e seis máquinas. A experimentação computacional foi realizada comparando os modelos criados com o modelo SG*. Ambos os modelos se mostraram superiores ao modelo SG*, já que produzem as mesmas soluções (ótimas) do SG*, com tempo computacional significativamente inferior. O modelo TS1 mostrou-se ainda superior ao TS2, sendo classificado como o melhor modelo para o problema em ambiente *flowshop* com *setups* dependentes da sequência.

Posteriormente, Stafford e Tseng (2002) propuseram dois novos modelos MILP com o critério de minimização do *makespan*, que servem para o *flowshop* convencional, para o *flowshop* com *setups* dependentes, para o *no-wait flowshop* e para o *no-wait flowshop* com *setups* dependentes, todos com *m*-máquinas. Ambos os modelos se mostraram viáveis para o problema com $n \leq 7$, e um dos modelos se mostrou melhor do que os modelos existentes em relação à eficiência computacional para o *flowshop* com *setups* dependentes com $n \geq 8$.

Ríos-Mercado e Bard (2003) estudaram dois diferentes modelos de programação MIP para o problema $F_m/ST_{sd}/C_{max}$. Um modelo é relacionado ao

problema assimétrico do caixeiro viajante (*Assymetric Traveling Salesman Problem* - ATSP) e o outro é derivado do modelo SG* de Srikar e Ghosh (1986). Eles também desenvolveram inequações inteiras-mistas para os dois modelos e as avaliaram através de uma estrutura *branch-and-cut*.

Crowder (2006) abordou o problema de *flowshop* flexível com *m*-máquinas, *setups* dependentes da sequência, máquinas uniformes e limite de estocagem, com o objetivo de minimizar o *makespan*, e desenvolveu um modelo matemático para resolver otimamente problemas de pequeno porte. Ele reportou que, mesmo para resolver problemas muito pequenos, o modelo requer alto tempo computacional.

Naini (2007) abordou o problema em ambiente *flowshop* em manufatura celular com o objetivo de minimizar *total flowtime*. Ele desenvolveu um algoritmo *branch-and-bound* juntamente com um eficiente limitante inferior. Para a otimização bi-critério, Naini propôs um algoritmo genético multi-objetivo, que teve seu resultado comparado com limitantes inferiores, obtendo um desvio relativo de apenas 1%.

Recentemente, Eren (2010) propôs um modelo de programação inteira para o *flowshop* com *m*-máquinas com um objetivo de minimização bi-critério: *makespan* e soma ponderada das datas de término. O modelo consegue resolver problemas com até 18 tarefas e 6 máquinas.

A Tabela 1 a seguir resume as pesquisas com métodos de solução exata realizadas no ambiente *flowshop* com tempos de *setup* dependentes da sequência.

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Corwin e Esogbue (1974)	2	C_{max}	*Setup* dependente em uma máquina	Programação Dinâmica
Uskup e Smith (1975)	2	Custo total de *setup*	Com datas de entrega	*Branch-and-bound*
Gupta (1975)	*m*	Custo Total de Oportunidade	-	Algoritmo de Busca Lexicográfica
Srikar e Ghosh (1986)	*m*	$C_{max}, F/$	Problemas com até 6 tarefas e 6 máquinas	Formulação MILP
Szwarc e Gupta (1987)	2	C_{max}	*Setup* dividido em dois componentes: um depende da tarefa em curso e o outro da tarefa seguinte	Algoritmo delimitado polinomialmente
Gupta, Das e Ghosh (1995)	*m*	C_{max}	Problemas com até 20 tarefas	*Branch-and-Bound*
Ríos-Mercado e Bard (1996)	*m*	C_{max}	-	Inequações válidas para uma formulação MIP e uma MILP
Hwang e Sun (1997)	2	C_{max}	*Work-Flow* Reentrante/ Tempo de *setup* na primeira máquina depende das duas tarefas anteriores	Programação Dinâmica
Sonmez e Baykasoglu (1998)	*m*	$\sum w_j T_j$	-	Programação Dinâmica
Ríos-Mercado e	*m*	C_{max}	-	2 Formulações MIP

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Bard (1998a)				Comparação da estrutura *branch-and-cut* com a *branch-and-bound*
Ríos-Mercado e Bard (1999a)	*m*	C_{max}	-	*Branch-and-Bound* Limitantes Inferiores e Superiores
Sun e Hwang (2001)	2	C_{max}	Tempos de setup dependem de k tarefas precedentes (presentes apenas na segunda máquina)	Programação Dinâmica
Sethanan (2001)	*m*	C_{max}	*Flowshop* flexível / Problemas com até 5 tarefas, 5 estágios e 8 máquinas	Modelo Matemático
Tseng e Stafford (2001)	*m*	C_{max}, $\sum F_j$, tempo total ocioso das tarefas e tempo ocioso de máquina	-	2 Formulações MILP
Stafford e Tseng (2002)	*m*	C_{max}	Modelos para *flowshop*: convencional, com *setups* dependentes, *no-wait* e *no-wait* com *setups* dependentes	2 Formulações MILP
Ríos-Mercado e Bard (2003)	*m*	C_{max}	-	2 Formulações MIP *Branch-and-Cut* Inequações Inteiras-Mistas

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Crowder (2006)	*m*	C_{max}	*Flowshop* flexível/ Limites de estocagem / Problemas pequenos	Modelo Matemático
Naini (2007)	*m*	$\sum F_j$	Manufatura Celular	*Branch-and-Bound* / Limitantes Inferiores
Eren (2010)	*m*	$\sum w_j C_j + C_{max}$	Problemas com até 18 tarefas e 6 máquinas	Modelo Matemático

Tabela 1 – Resumo das pesquisas com métodos de solução exata para *flowshop* com *setups* dependentes da sequência

Pode-se observar na Tabela 1 que a maioria dos trabalhos com métodos de solução exata para o problema de *flowshop* com tempos de *setup* dependentes da sequência possui como critério de minimização o *makespan.* Ademais, observa-se que a maioria desses métodos propostos possui restrições para funcionar, como limitações no tamanho do porte do problema, o que pode ser justificado devido à complexidade do problema. A seguir, no Tópico 2.1.2, serão detalhadas as pesquisas com métodos heurísticos construtivos para esse mesmo problema.

2.1.2 Pesquisas com Métodos Heurísticos Construtivos

Gupta e Darrow (1985) generalizaram o problema em ambiente *flowshop* com duas máquinas abordado por Corwin e Esogbue (1974), considerando tempos de *setup* dependentes da sequência em ambas as máquinas e estabeleceram duas heurísticas para o problema. Em seguida, Gupta e Darrow

(1986) consideraram o mesmo problema e mostraram que ele é fortemente *NP-hard* até no caso em que os tempos de *setup* são dependentes da sequência em apenas uma máquina. Eles também apresentaram quatro heurísticas para problemas de pequeno e grande porte com o objetivo de minimizar o *makespan* e as testaram empiricamente. Os resultados mostraram que, quando os tempos de *setup* correspondem a até 10% dos tempos de processamento, as heurísticas obtêm bons resultados. Porém, quando a proporção dos tempos de *setup* com relação aos tempos de processamento aumenta, o desempenho das heurísticas diminui drasticamente.

Rajagopalan e Karimi (1987) apresentaram uma heurística construtiva para minimizar o *makespan* em um *flowshop* com *m*-máquinas com *setups* dependentes da sequência e tempos de transferência. A heurística, chamada de *Idle Matrix Search*, realiza a programação baseada na matriz de tempos ociosos das máquinas. Na experimentação computacional, adaptaram para o problema abordado a heurística RA de Dannenbring (1977), que quebra o problema em *m/2* problemas de duas máquinas e resolve com a regra de Johnson aquele com a maior soma dos tempos de processamento. Os resultados mostraram que, em comparação com a RA, a heurística proposta obtém resultados 8% melhores com esforço computacional 50% menor.

Simons (1992) apresentou quatro heurísticas para o problema em ambiente *flowshop* com tempos de *setup* com o objetivo de minimizar o *makespan*, sendo duas delas específicas para tempos de *setup* dependentes da sequência: a heurística TOTAL, que utiliza a soma dos tempos de processamento e dos tempos de *setup*, e a heurística SETUP, que utiliza apenas a soma dos tempos de *setup*. As heurísticas foram testadas com três métodos de *benchmark*: um de programação

aleatória, um que não considerava os tempos de *setup* e um que assumia que um tempo de *setup* médio era incluído em cada tempo de processamento. Os resultados computacionais mostraram que ambas heurísticas produzem soluções melhores do que os *benchmarks*, e que a heurística TOTAL obtém melhores resultados do que a SETUP.

Das, Gupta e Khumawala (1995) desenvolveram um método heurístico com um índice para economia de tempo com o objetivo de encontrar a solução que minimizasse o *makespan*. O método proposto calcula a economia em tempo de cada sequência e então indica aquela de maior economia como a melhor solução. Os resultados computacionais mostraram que ele obtém bons resultados em situações cujos tempos de *setup* são maiores do que os tempos de processamento.

Rajendran e Ziegler (1997) abordaram o problema $F_m/ST_{sd}/C_{max}$. Eles propuseram três heurísticas e incorporaram a elas um método de melhoria a fim de aumentar a qualidade das soluções. A experimentação computacional foi realizada com a heurística de duas fases de Proust, Gupta e Deschamps (1991), adaptada ao caso. Uma das heurísticas propostas se mostrou superior às demais, tanto na qualidade da solução quanto no tempo computacional.

Ríos-Mercado e Bard (1998b) também abordaram o problema $F_m/ST_{sd}/C_{max}$ e apresentaram dois métodos heurísticos. Um deles era uma extensão do método NEH, de Nawaz, Enscore e Ham (1983), que insere uma tarefa na melhor posição disponível de uma série de sequências incompletas, e o outro método será abordado no próximo tópico da dissertação, por se tratar de uma meta-heurística. A heurística apresentada se mostrou superior ao método SETUP de Simons (1992), para o caso em que os tempos de *setup* são menores em

comparação aos tempos de processamento. Entretanto, o método SETUP de Simons (1992) se mostrou relativamente superior nos casos onde os tempos de *setup* e de processamento são distribuídos identicamente.

Em seguida, Ríos-Mercado e Bard (1999b) apresentaram uma heurística que transforma o problema $F_m/ST_{sd}/C_{max}$ no TSP através de uma função de custo que penaliza tanto tempos de *setup* altos como programações ruins. Na experimentação computacional, mostraram que a heurística desenvolvida obtém melhores resultados do que a meta-heurística proposta por Ríos-Mercado e Bard (1998b), para casos em que o número de máquinas é pequeno ou quando as variações do tempo de *setup* são grandes.

Maddux III e Gupta (2003) abordaram o problema $F2/ST_{sd}/C_{max}$ sem estocagem entre as máquinas, em um ambiente onde algumas tarefas são finalizadas já na primeira máquina, e outras devem passar pelas duas máquinas. Eles desenvolveram um limitante inferior e apresentaram uma heurística para resolver o problema.

Rajendran e Ziegler (2003) estudaram o problema de *flowshop* com tempos de *setup* dependentes da sequência com uma combinação de dois dos objetivos considerados por Rajendran e Ziegler (1997) e Sonmez e Baykasoglu (1998), ou seja, $F_m/ST_{sd}/\sum w_j T_j + \sum w_j F_j$ (soma ponderada dos atrasos e soma ponderada dos tempos de fluxo). Eles desenvolveram uma heurística que gera duas sequências: uma parte de uma sequência vazia e identifica a melhor tarefa que pode ser programada em uma sequência parcial, considerando o tempo total da programação com a inserção da tarefa e os custos de atraso da programação, e a outra é obtida a partir das datas de entrega e custos de atraso e de espera das tarefas. A melhor das duas sequências é então escolhida. Eles ainda desenvolveram

um método de melhoria para a heurística. Para a experimentação computacional, adaptaram para o problema a melhor heurística desenvolvida por Gelders e Sambandam (1978), GS*, que sequencia as tarefas uma a uma, seguindo o critério da menor soma do tempo ocioso de todas as máquinas. Então, compararam sua heurística com a GS*, com um procedimento de busca aleatória, e com uma busca local *greedy*. Os resultados mostraram que a heurística desenvolvida obtém melhores resultados do que os métodos testados, em menor tempo computacional.

Como citado no tópico anterior, Crowder (2006) estudou o problema de *flowshop* flexível com *setups* dependentes da sequência, máquinas uniformes e limite de estocagem, com o objetivo de minimizar o *makespan*. Além do modelo matemático proposto, ele desenvolveu duas heurísticas construtivas e dois limitantes inferiores. Os resultados mostraram que as heurísticas obtêm bons resultados para problemas pequenos, mas têm seu desempenho diminuído com o aumento do número de estágios da produção.

Tseng, Gupta e Stafford (2006) desenvolveram um método heurístico baseado em penalidades para a minimização do *makespan*. O método determina a penalidade em relação ao tempo de cada sequência e seleciona a sequência com a menor penalidade. Na experimentação computacional, a heurística se mostrou superior em relação à qualidade de soluções, quando comparada à heurística de Das, Gupta e Khumawala (1995).

Moccellin e Nagano (2007) apresentaram uma propriedade estrutural para o problema de programação da produção com tempos de preparação das máquinas separados dos tempos de processamento das tarefas, podendo os tempos de preparação ser tanto dependentes quanto independentes da sequência.

Essa propriedade fornece um limitante superior do tempo de máquina parada entre a sua preparação e o início de execução das tarefas. Utilizando a propriedade, o problema abordado com o objetivo de minimizar o *makespan* pode ser resolvido de maneira heurística através de uma analogia com o ATSP. Na experimentação computacional, Moccellin e Nagano (2007) utilizaram seis relações para os tempos de processamento das tarefas e os de preparação das máquinas, com base nos trabalhos de Simons (1992), Das, Gupta e Khumawala (1995), Rajendran e Ziegler (1997), Rios-Mercado & Bard (1998b, 1999b) e Weng, Lu e Ren (2001). Os resultados mostraram que a analogia proposta é adequada para problema com $n \geq 10$.

Eren (2010), além de propor um modelo de programação inteira, também apresentou três heurísticas construtivas para o *flowshop* com *m*-máquinas com o objetivo de minimização do *makespan* e da soma ponderada das datas de término. As heurísticas são baseadas na heurística NEH, de Nawaz, Enscore e Ham (1983). Na experimentação computacional, o autor comparou as heurísticas com as soluções ótimas obtidas pelo modelo de programação inteira, para problemas de pequeno porte, e para problemas de grande porte, mediu o desvio médio encontrado entre elas.

Fuchigami (2010) abordou o problema de *flowshop* flexível com critério de minimização do *makespan* e definiu treze regras de sequenciamento para as tarefas nos diversos estágios de produção, sendo seis delas baseadas na regra do menor tempo de processamento (*Shortest Processing Time* – SPT), seis baseadas na regra do maior tempo de processamento (*Longest Processing Time* – LPT) e uma na regra de ordenação aleatória no primeiro estágio, utilizada como parâmetro de comparação. O autor ainda desenvolveu sete heurísticas

construtivas, sendo que duas delas, que tratavam com prioridade os estágios com máquina única (gargalos da produção), obtiveram os melhores desempenhos.

A Tabela 2 a seguir resume as pesquisas com métodos heurísticos construtivos realizadas no ambiente *flowshop* com tempos de *setup* dependentes da sequência.

Referências	**# de Estágios**	**Critérios**	**Condições Adicionais**	**Abordagem**
Gupta e Darrow (1985)	2	C_{max}	-	2 Heurísticas
Gupta e Darrow (1986)	2	C_{max}	-	Provaram que o problema é $\mathcal{NP}$-*hard*
				4 Heurísticas
Rajagopalan e Karimi (1987)	m	C_{max}	Tempos de transferência	Heurística
Simons (1992)	m	C_{max}	-	2 Heurísticas
Das, Gupta e Khumawala (1995)	m	C_{max}	Heurística possui um índice para economia de tempo	Heurística
Rajendran e Ziegler (1997)	m	$\sum w_j F_j$	-	Método de Melhoria
				3 Heurísticas
Ríos-Mercado e Bard (1998b)	m	C_{max}	-	Heurística
Ríos-Mercado e Bard (1999b)	m	C_{max}	Heurística transforma o problema no TSP	Heurística
Maddux III e Gupta (2003)	2	C_{max}	Algumas operações não necessitam de processamento na segunda máquina	Limitante Inferior
				Heurística
Rajendran e	m	$\sum w_j F_j$ +	Heurística gera	Heurística

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Ziegler (2003)		$\sum w_j T_j$	duas sequências e escolhe a melhor	Método de Melhoria
Crowder (2006)	m	C_{max}	*Flowshop* flexível/ Limites de estocagem	2 Limitantes Inferiores
				2 Heurísticas
Tseng, Gupta e Stafford (2006)	m	C_{max}	Heurística baseada em penalidades	Heurística
Moccellin e Nagano (2007)	m	C_{max}	Heurística baseada no ATSP	Propriedade Estrutural
				Heurística
Eren (2010)	m	$\sum w_j C_j + C_{max}$	-	3 Heurísticas
Fuchigami (2010)	m	C_{max}	*Flowshop* flexível	13 Regras de Sequenciamento
				7 Heurísticas

Tabela 2 - Resumo das pesquisas com métodos heurísticos construtivos para *flowshop* com *setups* dependentes da sequência

Observando-se a Tabela 2 acima, nota-se um predomínio do critério de minimização do *makespan* nas pesquisas com métodos heurísticos construtivos. Nota-se também que as pesquisas com soluções aproximativas construtivas são mais recentes do que as pesquisas com soluções exatas, tendo sido inciadas em meados dos anos 1980. A seguir, no Tópico 2.1.3, serão detalhadas as pesquisas com métodos meta-heurísticos para esse mesmo problema.

2.1.3 Pesquisas com Métodos Meta-Heurísticos

Parthasarathy e Rajendran (1997a) consideraram o problema $F_m/ST_{sd}/\sum w_j T_j$ e realizaram um estudo de caso em uma indústria de brocas. Eles

desenvolveram um método baseado no *simulated annealing* e apresentaram um novo método de geração de vizinhança, chamado de *Random Insertion Perturbation Scheme* (RIPS). Os resultados computacionais mostraram que o *simulated annealing* supera a busca tabu em cerca de 70% das vezes. Em seguida, Parthasarathy e Rajendran (1997b) ampliaram sua heurística para minimizar o atraso máximo.

Como citado no tópico anterior, Ríos-Mercado e Bard (1998b), além da heurística construtiva, também desenvolveram uma meta-heurística para o problema $F_m/ST_{sd}/C_{max}$, chamada de *Greedy Randomized Adaptive Search Procedure* (GRASP). Eles ainda apresentaram um limitante inferior de duas fases e desenvolveram procedimentos de busca local.

Hwang e Sun (1998) deram continuidade ao trabalho de Hwang e Sun (1997) em uma linha de produção de uma indústria de caminhões com duas máquinas, onde todas as tarefas necessitavam de processamento na primeira máquina mais de uma vez e o tempo de *setup* de uma tarefa na primeira máquina dependia das duas tarefas anteriores. Eles acrescentaram ao trabalho um algoritmo genético, visto que o esforço computacional da programação dinâmica é inviável para problemas de grande porte. Na experimentação computacional, compararam o algoritmo genético desenvolvido com as heurísticas NSH de Karg e Thompson (1964) e LPH de Ahn (1995), adaptadas ao caso. A heurística NSH compõe a sequência através de uma formulação TSP, que busca o menor tempo de *setup.* A heurística LPH é similar à NSH, mas busca o menor custo de oportunidade do *setup* ao invés do menor tempo de *setup*. Os resultados mostraram a superioridade do algoritmo genético.

Norman e Bean (1999) abordaram o problema em ambiente *flowshop* com *m*-máquinas com *setups* dependentes da sequência com o objetivo de minimizar o atraso total, com as seguintes considerações: datas de liberação, inatividade de máquinas e escassez de ferramentas. Eles desenvolveram um algoritmo genético, provaram sua convergência e o testaram para casos reais da indústria automobilística, obtendo bons resultados. Ainda mostraram que o algoritmo pode ser estendido para restrições adicionais, como limite de estocagem entre as máquinas, e utilizado com outras funções-objetivo, como o *makespan*.

Demirkol e Uzsoy (2000) estudaram a minimização do atraso/adiantamento máximo em ambiente *flowshop* reentrante. Esse ambiente consiste em *m*-máquinas organizadas como um *flowshop* clássico. As tarefas passam pela linha, podendo pular algumas máquinas. Assim que uma tarefa completa a passagem pela linha, ela pode retornar a essa linha para realizar mais operações, não necessariamente nas mesmas máquinas em que passou anteriormente. Inicialmente, os autores apresentaram um método genérico de decomposição, aplicável a diferentes ambientes de programação. Porém, ele não obteve bom desempenho no *flowshop* reentrante. Eles então desenvolveram um procedimento de busca tabu e o utilizaram com o método desenvolvido, obtendo melhores resultados, porém em tempo computacional alto.

Como citado anteriormente, Sun e Hwang (2001) desenvolveram uma formulação de programação dinâmica para o problema $F2/ST_{sd}/C_{max}$, com tempos de *setup* presentes apenas na segunda máquina, que dependiam das k tarefas precedentes. Além da programação ótima, os autores desenvolveram um algoritmo genético para problemas de grande porte e mostraram, através do

desenvolvimento de um limitante inferior, que o algoritmo encontra bons resultados em tempo aceitável.

Sethanan (2001), além de desenvolver um modelo matemático para o *flowshop* flexível com m estágios para minimizar o *makespan*, também desenvolveu um método meta-heurístico baseado na busca tabu e um método para gerar sua sequência inicial. A fim de avaliar o desempenho da meta-heurística, ele desenvolveu dois limitantes inferiores, já que não havia trabalhos prévios sobre esse problema, e comparou os resultados da heurística com os valores encontrados. O método que gera a solução inicial se mostrou rápido, mesmo quando utilizado em problemas grandes. A heurística baseada em busca tabu encontra bons resultados, mas seu tempo computacional aumenta muito com o tamanho do problema. Sethanan (2001) ainda mostrou que a taxa de melhoria da solução encontrada pelo método, depois de passar pela heurística de busca tabu, era de 2,95 a 11,85%.

Andrés et al. (2005) abordaram o problema $F_m/ST_{sd}, prec$ com o objetivo de minimizar o *makespan* e o atraso total e propuseram um algoritmo genético multi-objetivos para solucionar o problema.

Ruiz, Maroto e Alcaraz (2005) propuseram dois algoritmos genéticos, GA e HGA, com o objetivo de minimizar o *makespan*, além de adaptarem cinco meta-heurísticas existentes de *flowshop* clássico para *flowshop* com tempos de *setup* dependentes da sequência, como o *simulated annealing* de Osman e Potts (1989) e a busca tabu de Widmer e Hertz (1989), para comparação. Ainda compararam seus algoritmos com as heurísticas TOTAL e SETUP de Simons (1992), a meta-heurística GRASP de Ríos-Mercado e Bard (1998b) e a heurística

baseada no TSP de Ríos-Mercado e Bard (1999b). Os resultados computacionais mostraram que os algoritmos GA e HGA obtêm resultados satisfatórios, superiores à heurística de Ríos-Mercado e Bard (1999b) e, em alguns casos, à meta-heurística GRASP.

Naini (2007), já citado anteriormente, além de considerar o problema em ambiente *flowshop* em manufatura celular com o objetivo de minimizar o *total flowtime*, também abordou esse problema com o objetivo de minimizar o *makespan*. Esses dois critérios foram considerados separadamente e simultaneamente. Ele desenvolveu várias meta-heurísticas baseadas na busca tabu para minimizar o *makespan*. Os resultados computacionais mostraram que as meta-heurísticas obtém bons resultados em tempo razoável, com desvios menores de 1% para problemas pequenos. Para a otimização bi-critério, Naini propôs um algoritmo genético multi-objetivo, que teve seu resultado comparado com limitantes inferiores, obtendo um desvio relativo de apenas 1%.

Ruiz e Stüzle (2008) apresentaram dois algoritmos iterativos *greedy* baseados em busca local simples para os problemas $F_m/ST_{sd}/\sum w_j T_j$ e $F_m/ST_{sd}/C_{max}$. Ambos os algoritmos são baseados na heurística NEH de Nawaz, Enscore e Ham (1983), sendo que um deles possui uma busca local simples descendente. Os experimentos computacionais mostraram que eles obtiveram melhor desempenho do que as heurísticas de Rajendran e Ziegler (2003) e os algoritmos genéticos de Ruiz, Maroto e Alcaraz (2005).

A Tabela 3 a seguir resume as pesquisas com métodos meta-heurísticos realizadas no ambiente *flowshop* com tempos de *setup* dependentes da sequência.

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Parthasarathy e Rajendran (1997a)	*m*	$\sum w_j T_j$	-	*Simulated Annealing* Método de Geração de Vizinhança
Parthasarathy e Rajendran (1997b)	*m*	T_{max}	-	*Simulated Annealing* Método de Geração de Vizinhança
Ríos-Mercado e Bard (1998b)	*m*	C_{max}	-	Meta-Heurística GRASP Limitante Inferior Procedimentos de Busca Local
Hwang e Sun (1998)	2	C_{max}	*Work-Flow* Reentrante/ Tempo de *setup* na primeira máquina depende das duas tarefas anteriores	Algoritmo Genético
Norman e Bean (1999)	*m*	$\sum T_j$	Escassez de ferramentas/ Inatividade de máquina/ Datas de Liberação	Algoritmo Genético
Demirkol and Uzsoy (2000)	*m*	L_{max}	*Flowshop* reentrante	Métodos de decomposição Procedimento de Busca Tabu
Sun e Hwang (2001)	2	C_{max}	Tempos de *setup* dependem de k tarefas precedentes (presentes apenas na segunda máquina)	Limitante Inferior Algoritmo Genético
Sethanan (2001)	*m*	C_{max}	*Flowshop* flexível	Procedimento de inicialização Heurística baseada em Busca-Tabu

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
				2 Limitantes Inferiores
Andrés et al. (2005)	m	C_{max} e $\sum T_j$	Restrições de Precedência	Algoritmo Genético Multi-Objetivos
Ruiz, Maroto e Alcaraz (2005)	m	C_{max}	-	2 Algoritmos Genéticos Adaptações (para *flowshop* com *setup* dependente da sequência) de 5 meta-heurísticas existentes
Naini (2007)	m	$\frac{C_{max}}{C_{max} + \sum F_j}$	Manufatura Celular	Meta-heurísticas baseadas na Busca-tabu Algoritmo Genético Multi-Objetivos
Ruiz e Stützle (2008)	m	C_{max} e $\sum w_j T_j$	-	2 Algoritmos Iterativos *Greedy*

Tabela 3 - Resumo das pesquisas com métodos meta-heurísticos para *flowshop* com *setups* dependentes da sequência

Através da Tabela 3 acima, observa-se que a abordagem com métodos meta-heurísticos para *flowshop* com tempos de *setup* dependentes da sequência é bastante recente, tendo surgido no final dos anos 1990. Nota-se um predomínio do critério de minimização do *makespan* nestas pesquisas, embora não tão acentuado quanto das pesquisas com métodos heurísticos construtivos. A seguir serão realizadas considerações acerca da revisão bibliográfica deste capítulo.

2.1.4 Considerações

A partir da revisão bibliográfica realizada (vide Tabelas 1, 2 e 3), nota-se que a maioria das pesquisas em *flowshop* com tempos de *setup* dependentes da sequência considera o ambiente geral de *m*-máquinas, como pode ser observado na Tabela 4 a seguir. Ademais, como já mencionado anteriormente, há um predomínio das pesquisas com critério de minimização do *makespan*.

	Ambiente de Programação		
Abordagem	***F2***	F_m	**Total**
Exata	5	14	19
Construtiva	3	12	15
Meta-Heurística	2	10	12
Total	10	36	46

Tabela 4 – Quantidade de pesquisas encontradas por ambiente de programação e abordagem

Ao todo foram encontrados 41 trabalhos em *flowshop* com *setups* dependentes da sequência. Na Tabela 4, o total de 46 pesquisas se deve ao fato de um mesmo trabalho abordar diferentes métodos de solução. Destas 46 pesquisas, 36 abordam o *flowshop* com *m*-máquinas e apenas 10 com duas máquinas.

Dentre as três abordagens consideradas na revisão, aquela que possui mais pesquisas é a exata (i.e., abordagem de métodos de solução exata). Isso se deve ao fato de suas pesquisas terem sido iniciadas em meados dos anos 1970, enquanto que as pesquisas com métodos heurísticos (também conhecidos como aproximativos) se iniciaram apenas 10 anos depois.

Através do Gráfico 1 abaixo, pode-se visualizar uma forte tendência atual no emprego dos métodos heurísticos (construtivos e meta-heurísticos), já que os métodos de solução exata não são eficientes computacionalmente em problemas de médio e grande porte.

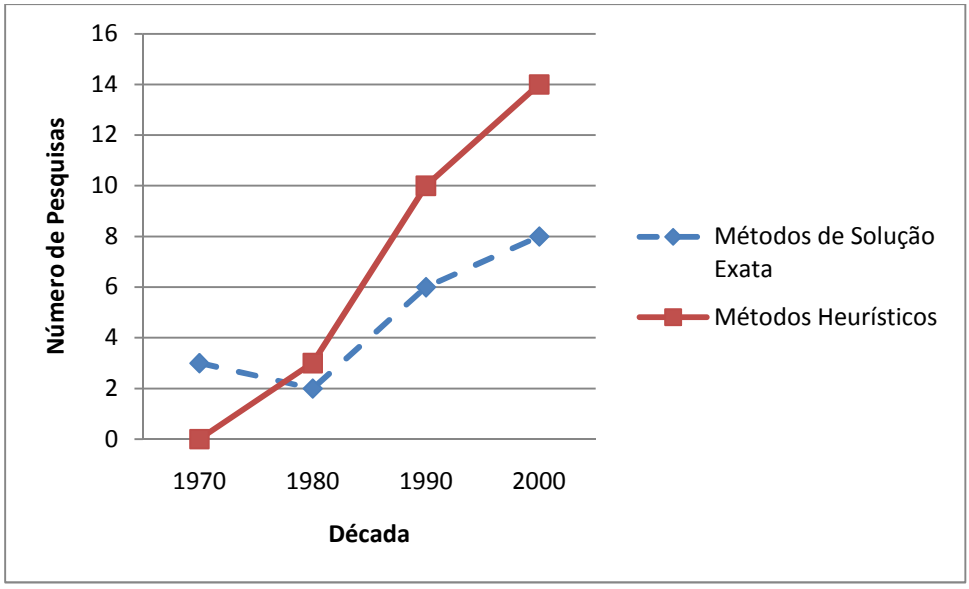

Gráfico 1 - Número de Pesquisas *vs.* Década

A seguir, no Capítulo 3, o problema em ambiente *no-wait flowshop* será abordado e detalhado.

CAPÍTULO 3

NO-WAIT FLOWSHOP

As suposições comumente consideradas em problemas de programação *flowshop* impedem várias aplicações práticas, importantes na realidade (DUDEK; PANWALKAR; SMITH, 1992; TSENG; STAFFORD, 2001). No *flowshop* clássico, considera-se que não há limite de estocagem entre as máquinas. Porém, essa situação pode não ser possível em situações reais. Em um problema de programação da produção no ambiente *no-wait flowshop* (NWFS), as tarefas devem ser processadas do início ao final sem interrupções (HALL; SRISKANDARAJAH, 1996). Consequentemente, o processamento de uma tarefa na primeira máquina pode necessitar de um atraso para garantir que não ocorra interrupção em uma máquina subsequente.

A principal característica do NWFS é que a operação $i+1$ de uma tarefa tem que ser processada logo após o término da operação i, sendo $1 \leq i \leq m - 1$. Assim, não pode haver tempo de espera no processamento de uma tarefa de uma máquina para a próxima. Um exemplo de sequenciamento de tarefas ($1 \leq i \leq n$) para o problema *no-wait flowshop* é mostrado na Figura 2.

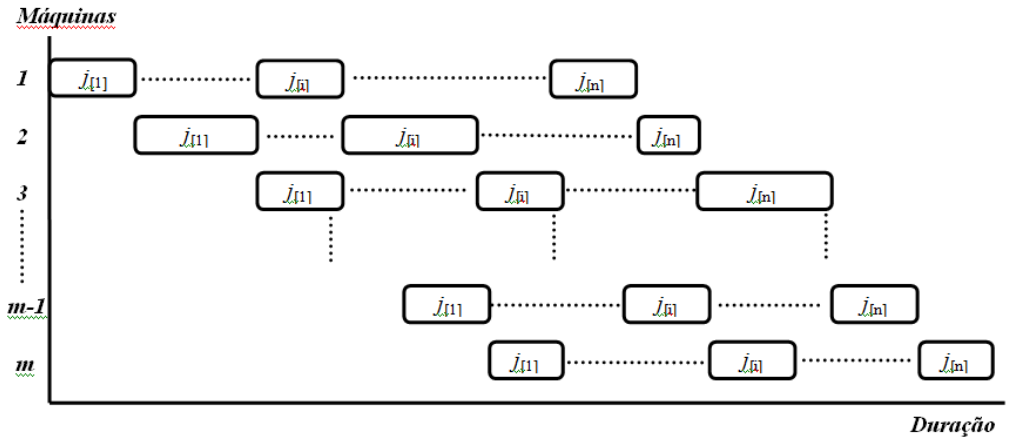

Figura 2 - *No-wait flowshop* com *m* máquinas e *n* tarefas

A Figura 2 apresenta o sequenciamento de um problema de n tarefas em m máquinas e considera que todas as tarefas estão disponíveis ao mesmo tempo no chão de fábrica. Uma sequência (solução) pode ser representada por $\sigma = (j_{[1]}, \dots, j_{[i]}, \dots, j_{[n]})$, onde $[i]$ representa a tarefa que aparece na i-ésima posição da sequência.

O ambiente *no-wait* é comum na produção de aço, plástico, químicas, produtos de alumínio e alimentos (BROWN; MCGARVEY; VENTURA, 2004). Por exemplo, no caso da produção de aço, o metal aquecido deve atravessar a sequência de operações continuamente, antes que esfrie, para prevenir defeitos na composição do material (ALDOWAISAN, 2001). Na indústria alimentícia, ocorre um processo semelhante: para garantir a qualidade dos alimentos, eles devem ser enlatados logo após o cozimento. Outro exemplo pode ser observado na área de serviços, onde os clientes têm alta intolerância à espera.

Este ambiente é também motivado por conceitos modernos de gestão da produção, como o *just-in-time* e a produção enxuta (BIANCO; DELL'OLMO; GIORDANI, 1999).

Para o problema *no-wait flowshop*, vários são os métodos utilizados, como técnicas de eliminação implícita (*branch-and-bound)*, heurísticas, meta-heurísticas, regras de despacho e outros. Os mais utilizados atualmente são os métodos heurísticos e meta-heurísticos, devido às suas características de encontrar soluções próximas do ótimo para problemas de médio e grande porte em tempo computacional aceitável, como apresentado em Fink e Voβ (2003), Aldowaisan e Allahverdi (2004a) e Framinan, Nagano e Moccellin (2010).

A seguir, será realizada uma revisão bibliográfica abordando o problema em ambiente *no-wait flowshop*. Os Tópicos 3.1, 3.2 e 3.3 abordam as pesquisas com métodos de solução exata, métodos heurísticos construtivos e métodos meta-heurísticos, respectivamente.

3.1 Pesquisas com Métodos de Solução Exata

Os primeiros trabalhos para *no-wait flowshop* começaram a surgir na década de 60, com o critério de minimização do *makespan*. Gilmore e Gomory (1964) apresentaram um algoritmo TSP de tempo $O(n\log n)$ para o *flowshop* com duas máquinas. Reddi e Ramamoorthy (1972) e Wismer (1972) mostraram que o problema em ambiente *no-wait flowshop* com n tarefas pode ser modelado como o problema assimétrico do caixeiro viajante (ATSP) com $n+1$ cidades.

Deman e Baker (1974) foram os pioneiros a estudar o problema *no-wait flowshop* com o critério de minimização do *total flowtime*. Em sua pesquisa, eles desenvolveram vários procedimentos para gerar limitantes inferiores, que

foram incorporados em um algoritmo *branch-and-bound*, capaz de resolver otimamente problemas com até 6 máquinas e 12 tarefas. Os autores concluíram que os resultados alcançados foram satisfatórios e que o problema podia ser solucionado tão rapidamente quanto aqueles tradicionais que utilizavam como critério a minimização do *makespan*.

Papadimitriou e Kanellakis (1980) apresentaram um estudo com um caso particular para a solução exata de problemas de pequeno porte com o critério do *makespan*, para 3 e 4 máquinas, em que evidenciaram as dificuldades em estender a resolução exata do problema para portes maiores.

Adiri e Pohoryles (1982) estudaram os ambientes *no-wait* e *no-idle* com o critério de minimização do *makespan.* Eles provaram algumas propriedades para encontrar o valor ótimo, mas não desenvolveram algoritmos para o caso.

Kang e Markland (1988) selecionaram na literatura seis diferentes métodos para o problema de *flowshop* clássico e os adaptaram para o *no-wait flowshop*. Os métodos selecionados foram as heurísticas de Gupta (1968), Campbell, Dudek e Smith (1970), Dannenbring (1977) e Szwarc (1981) e as programações MIP de Wagner (1959) e MILP de Stafford (1983). Os métodos adaptados foram testados para problemas de pequeno porte, de até 10 tarefas e 5 máquinas, considerando-se como critério o *makespan*. Os resultados mostraram que a programação MIP de Wagner (1959) e a programação MILP de Stafford (1983) encontram soluções ótimas, porém com alto tempo computacional, enquanto que as heurísticas de Gupta (1966) e Szwarc (1981) obtêm resultados razoáveis, com boa eficiência computacional.

Pekny e Miller (1991) apresentaram um algoritmo *branch-and-bound* para o TSP, com o objetivo de resolver o problema de programação *no-wait* com critério de minimização do *makespan* em uma indústria química. Eles compararam o algoritmo desenvolvido com uma heurística simples e com um algoritmo de *simulated annealing* adaptado de Press et al. (1988). Os resultados mostraram que a escolha entre o uso de métodos exatos ou de heurísticas aproximadas depende da disponibilidade de tempo e da facilidade de implementação requerida.

Van der Veen e Van Dal (1991) consideraram o problema com *m*-máquinas e critério de minimização do *makespan* com uma restrição: o tempo de processamento era fixo em todas as máquinas, exceto duas. Então, mostraram que se uma das duas máquinas que não possui tempo fixo está na primeira ou na última posição do seqüenciamento, o problema pode ser resolvido em tempo $O(n^2 \log n)$, e se as duas máquinas são consecutivas, o problema pode ser resolvido em tempo $O(n \log n)$. Para o problema com critério de minimização do *total flowtime*, eles mostraram que, quando as matrizes de tempo são ordenadas (i.e., $p_{ki} \leq p_{k+1i}$, para $k \leq m$ máquinas e $i \leq n$ tarefas), o problema pode ser resolvido em tempo polinomial.

Allahverdi e Aldowaisan (2004) abordaram o problema com otimização bi-critério ($C_{\max} + L_{\max}$) e apresentaram uma relação de dominância e um algoritmo *branch-and-bound*. Também desenvolveram um limitante inferior para o caso de $m=2$.

A Tabela 5 a seguir resume as pesquisas com métodos de solução exata realizadas no ambiente *no-wait flowshop* básico.

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Gilmore e Gomory (1964)	2	C_{max}	-	Algoritmo TSP
Reddi e Ramamoorthy (1972)	*m*	C_{max}	-	Modelo Matemático (ATSP)
Wismer (1972)	*m*	C_{max}	-	Modelo Matemático (ATSP)
Deman e Baker (1974)	*m*	$\sum F_j$	Problemas com até 12 tarefas e 6 máquinas	Limitantes Inferiores
				Branch-and-Bound
Papadimitriou e Kanellakis (1980)	*m*	C_{max}	Problemas com até 4 máquinas	Modelo Matemático
Adiri e Pohoryles (1982)	*m*	C_{max}	*no-idle*	Propriedades matemáticas
Kang e Markland (1988)	*m*	C_{max}	Problemas com até 10 tarefas e 5 máquinas	6 adaptações de modelos (de solução exata e aproximativos) existentes
Pekny e Miller (1991)	*m*	C_{max}	Problemas com até 4 máquinas	*Branch-and-Bound*
Van der Veen e Van Dal (1991)	*m*	C_{max}	O tempo de processamento é fixo em todas as máquinas, exceto duas	Modelo Matemático
		$\sum F_j$	As matrizes de tempo são ordenadas	Propriedades matemáticas
Allahverdi e Aldowaisan (2004)	*m*	$C_{max} + L_{max}$	-	*Branch-and-Bound*
				Relação de Dominância

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
	2			Limitante Inferior

Tabela 5 - Resumo das pesquisas com métodos de solução exata para *no-wait flowshop* básico

Como pode ser observado na Tabela 5, a maioria dos trabalhos com métodos de solução exata para o problema *no-wait flowshop* básico possui como critério de minimização o *makespan*, sendo que foram encontrados apenas dois trabalhos para o critério do *total flowtime*. É também interessante notar que na última década foi encontrada apenas uma pesquisa com abordagem exata, devido à dificuldade de se encontrar a solução ótima para esse problema. A seguir, no Tópico 3.2, serão detalhadas as pesquisas com métodos heurísticos construtivos para esse mesmo problema.

3.2 Pesquisas com Métodos Heurísticos Construtivos

Bonney e Gundry (1976) foram os primeiros a estabelecer um método heurístico para o problema com *m*-máquinas a fim de minimizar o *makespan*, utilizando relações geométricas e extensões de métodos criados para o problema de máquina única. King e Spachis (1980) abordaram o mesmo problema e apresentaram heurísticas com formas diferenciadas de ordenação de tarefas.

Rajendran e Chaudhuri (1990) apresentaram dois métodos heurísticos construtivos para minimizar o *total flowtime*, compostos de duas fases: uma primeira fase de ordenação inicial das tarefas e outra de inserção de tarefas,

estabelecendo no final a seqüência das tarefas. Os resultados da experimentação computacional mostraram que, para $12 \leq n \leq 20$, as soluções obtidas são melhores que aquelas obtidas pelos métodos propostos por Bonney e Gundry (1976) e King e Spachis (1980) adaptados para o problema de minimização do *total flowtime*.

Gangadharan e Rajendran (1993) desenvolveram dois métodos heurísticos construtivos para o problema com *m*-máquinas com critério de minimização do *makespan*. Os métodos possuem duas fases: na primeira, obtém-se a sequência inicial das tarefas e na segunda é construída a sequência final, através de um método de melhoria por inserção de tarefas. Na experimentação computacional, eles mostraram que ambos os métodos obtém melhores resultados do que as heurísticas de Bonney e Gundry (1976) e de King e Spachis (1980).

Em seguida, Rajendran (1994) também apresentou uma heurística construtiva de duas fases para o mesmo problema, que se mostrou superior às de Gangadharan e Rajendran (1993). Seu estudo destacou dois aspectos importantes para a minimização do *makespan*: os *delays* (esperas na primeira máquina, devido ao ambiente *no-wait*) devem ser minimizados e a última tarefa a ser programada deve possuir tempos de processamento pequenos. Tendo em vista esses aspectos, Rajendran (1994) apresentou uma heurística simples baseada no processo de inserção de tarefas. Atualmente, sua heurística ainda é considerada um dos melhores métodos construtivos para o ambiente.

Bertolissi (1999) apresentou um método heurístico de uma fase para o critério de minimização do *total flowtime*, no qual é calculado inicialmente o *flowtime* parcial de cada par de tarefas consecutivas, e então sequencia-se os menores *flowtime*s parciais. Na experimentação computacional, o autor comparou

a heurística criada com as heurísticas de Rajendran e Chaudhuri (1990) e de Bonney e Gundry (1976) adaptada para o caso. A experimentação computacional mostrou que o método proposto não obtém soluções superiores à heurística de Rajendran e Chaudhuri (1990).

Em um novo estudo, Bertolissi (2000) apresentou um novo método heurístico composto de duas fases. Na primeira fase é definida uma ordenação inicial, similar à sua heurística proposta anteriormente. Na segunda fase, aplica-se o procedimento de inserção de tarefas de Rajendran e Chaudhuri (1990). Os resultados computacionais mostraram que a heurística é superior à de Rajendran e Chaudhuri (1990).

Fink e Voβ (2003) abordaram o problema com *m*-máquinas com o critério de minimização do *total flowtime* e apresentaram três métodos construtivos, chamados NN, CHINS e PILOT, além de vários procedimentos meta-heurísticos, como busca tabu, *steepest descent* e *simulated annealing*. De acordo com os resultados computacionais, os procedimentos *simulated annealing* e busca tabu obtêm os melhores resultados, quando inicializados com as soluções providas pelas heurísticas CHINS e PILOT.

Aldowaisan e Allahverdi (2004a) abordaram o mesmo problema e apresentaram quatro novos métodos heurísticos compostos de três fases, chamados de PH1, PH2, PH3 e PH4, além de um método de melhoria, que foi implementado nas quatro heurísticas. As heurísticas acrescidas dos métodos de melhoria foram chamadas de PH1(p), PH2(p), PH3(p) e PH4(p). Os resultados dos experimentos mostraram que todas as heurísticas são superiores às duas heurísticas de Rajendran e Chaudhuri (1990) e ao algoritmo genético GA de Chen,

Neppalli e Aljaber (1996) para problemas com grande número de tarefas, sendo a heurística PH1(p) superior às demais.

Scardoelli (2006) também abordou o problema com critério de minimização do *total flowtime* e desenvolveu um método de duas fases, baseado no trabalho de Bertolissi (2000), e dois métodos de fase única, elaborados através da combinação de procedimentos de busca de melhoria por vizinhanças de permutação e de inserção de tarefas. Dentre os três métodos propostos, um dos métodos de fase única se mostrou superior aos demais e aos métodos de Rajendran e Chaudhuri (1990) e Bertolissi (2000).

Branco (2006) abordou o problema com critério de minimização do *makespan* e apresentou um método de duas fases, semelhante ao de Rajendran (1994). A experimentação computacional incluiu o método de Rajendran (1994) e adaptações dos métodos de Nawaz, Enscore e Ham (1983), Nagano e Moccellin (2005) e Framinan e Leisten (2003), mostrando a superiodade do método proposto.

Li, Wang e Wu (2008) apresentaram um método heurístico de inserção, que calcula o *makespan* de forma eficiente. A experimentação computacional mostrou que o método desenvolvido obtém melhores resultados do que a melhor heurística de Gangadharan e Rajendran (1993), a heurística de Rajendran (1994) e meta-heurística SA-2 de Aldowaisan e Allahverdi (2003), além de requerer menor esforço computacional.

Framinan e Nagano (2008) propuseram uma heurística de dois passos, de complexidade $O(n^2m)$, baseada na analogia entre o problema *no-wait* e o TSP. No passo 1 é computado o *makespan* parcial para cada par de tarefas. No passo 2,

um procedimento rápido, chamado de FITSP é aplicado, a fim de solucionar o TSP criado no passo 1. Na experimentação computacional, os autores compararam a heurística proposta com outras estratégias de melhoria reportadas na literatura, mostrando que ela as supera para problemas de 5 a 80 tarefas.

Laha e Chakraborty (2009) também apresentaram uma heurística construtiva baseada em inserção de tarefas para o problema de minimização do *makespan*, e a compararam com as mesmas comparadas por Li, Wang e Wu (2008). Os resultados mostraram que ela obtém melhores resultados, mas requer maior tempo computacional do que as heurísticas de Rajendran (1994) e de Aldowaisan e Allahverdi (2003). A heurística proposta não foi comparada à de Li, Wang e Wu (2008).

Recentemente, Framinan, Nagano e Moccellin (2010) apresentaram uma heurística construtiva para o problema de minimização do *total flowtime* baseada na analogia do problema com duas máquinas. O método primeiro seleciona o candidato a ser alocado na sequência parcial. Então, a vizinhança da sequência parcial é examinada com buscas locais de inserção e de permutação, a fim de melhorar o resultado. Na experimentação computacional, os autores compararam a heurística proposta com as melhores heurísticas construtivas para o caso: as duas de Rajendran e Chaudhuri (1990), a de Bertolissi (2000), CHINS de Fink e Voβ (2003) e PH1(p) de Aldowaisan e Allahverdi (2004). Os resultados mostraram que a heurística proposta é superior em termos de qualidade de solução e do tempo computacional.

A Tabela 6 a seguir resume as pesquisas com métodos heurísticos construtivos no ambiente *no-wait flowshop* básico.

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Bonney e Gundry (1976)	m	C_{max}	-	Heurística
King e Spachis (1980)	m	C_{max}	-	Heurísticas
Rajendran e Chaudhuri (1990)	m	$\sum F_j$	-	2 Heurísticas de duas fases
Gangadharan e Rajendran (1993)	m	C_{max}	-	2 Heurísticas de duas fases
Rajendran (1994)	m	C_{max}	-	Heurística de duas fases
Bertolissi (1999)	m	$\sum F_j$	-	Heurística
Bertolissi (2000)	m	$\sum F_j$	-	Heurística de duas fases
Fink e Voβ (2003)	m	$\sum F_j$	Heurísticas desenvolvidas foram utilizadas para inicialização de meta-heurísticas	3 Heurísticas
Aldowaisan e Allahverdi (2004a)	m	$\sum F_j$	-	4 Heurísticas Construtivas de três fases Método de Melhoria
Scardoelli (2006)	m	$\sum F_j$	-	3 Heurísticas, sendo uma de duas fases
Branco (2006)	m	C_{max}	-	Heurística de duas fases
Li, Wang e Wu (2008)	m	C_{max}	-	Heurística de inserção
Framinan e Nagano (2008)	m	C_{max}	-	Heurística de duas fases

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
Laha e Chakraborty (2009)	m	C_{max}	-	Heurística de inserção
Framinan, Nagano e Moccellin (2010)	m	$\sum F_j$	-	Heurística

Tabela 6 - Resumo das pesquisas com métodos heurísticos construtivos para *no-wait flowshop* básico

A partir da Tabela 6, observa-se um equilíbrio entre as pesquisas com critério de minimização do *total flowtime* e do *makespan.* Nota-se também que há uma forte tendência de intensificação de esforços nas pesquisas com métodos heurísticos construtivos, ao contrário dos métodos de solução exata. A seguir, no Tópico 3.3, serão detalhadas as pesquisas com métodos meta-heurísticos para esse mesmo problema.

3.3 Pesquisas com Métodos Meta-Heurísticos

Chen, Neppalli e Aljaber (1996) desenvolveram uma meta-heurística baseada em algoritmo genético para o problema de minimização do *total flowtime*, com dois diferentes procedimentos: um gera os membros da população inicial aleatoriamente e o outro através da heurística de Rajendran e Chaudhuri (1990). A experimentação computacional mostrou que a meta-heurística proposta obtém melhores resultados com o segundo procedimento.

Aldowaisan e Allahverdi (2003) abordaram o problema com *m*-máquinas e critério de minimização do *makespan* e desenvolveram seis meta-heurísticas baseadas no *simulated annealing* e no algoritmo genético, denominadas SA, SA-1, SA-2, GEN, GEN-1 e GEN-2. As heurísticas de Rajendran (1994) e a melhor heurística de Gangadharan e Rajendran (1993) foram comparadas às heurísticas propostas. As meta-heurísticas baseadas no *simulated annealing* se mostraram superiores.

Posteriormente, Allahverdi e Aldowaisan (2004) propuseram dois algoritmos híbridos para a otimização bi-critério ($C_{\max} + L_{\max}$): um algoritmo baseado no *simulated annealing* e um algoritmo genético. Ambos os algoritmos híbridos obtiveram desempenhos semelhantes, sendo adequados para a otimização bi-critério ou para um único critério.

Grabowski e Pempera (2005, 2007) desenvolveram e compararam cinco algoritmos de busca local para o problema de minimização do *makespan*, sendo eles de busca tabu ou de busca decrescente. Com o objetivo de diminuir o tempo computacional dos métodos, eles utilizaram multimovimentos, ou seja, em uma única iteração de cada algoritmo, executam-se vários movimentos simultaneamente, guiando assim o processo de busca a áreas mais promissoras do espaço de solução, onde boas soluções poderiam ser encontradas. Na experimentação computacional, compararam os algoritmos criados com a heurística de Rajendran (1994) e com as soluções ótimas para os problemas, mostrando que quase todos os métodos propostos obtêm melhores soluções do que o método de Rajendran (1994), obtendo, na maioria das vezes, a solução ótima.

Kumar et al. (2006) apresentaram um método meta-heurístico para o problema com objetivo de minimização do *total flowtime*, chamado de algoritmo psico-clonal. A experimentação computacional mostrou que o algoritmo psico-clonal é superior ao algoritmo genético GA de Chen, Neppalli e Aljaber (1996) e aos métodos de Aldowaisan e Allahverdi (2004a).

Pan, Tasgetiren e Liang (2007) apresentaram um algoritmo de *Discrete Particle Swarm Otimization* (DPSO) para resolver o *no-wait flowshop* com critérios de minimização do tempo total de fluxo e do *makespan*. Os resultados computacionais mostraram que o algoritmo desenvolvido supera a busca tabu de Grabowski e Pempera (2005, 2007) para o critério do *makespan* e as meta-heurísticas de Fink e Voβ (2003) para o critério de tempo total de fluxo.

Ribeiro Filho, Nagano e Lorena (2007) apresentaram um algoritmo evolutivo híbrido, utilizando como heurística de busca o algoritmo *cluster search* de Oliveira e Lorena (2004) para resolver o problema com critério de minimização do *total flowtime*. A experimentação computacional mostrou que o método obtém melhores resultados do que os métodos apresentado por Pan, Tasgetiren e Liang (2007) e Fink e Voβ (2003).

A Tabela 7 a seguir resume as pesquisas com métodos meta-heurísticos realizadas no ambiente *no-wait flowshop* básico.

Referências	**# de Estágios**	**Critérios**	**Condições Adicionais**	**Abordagem**
Chen, Neppalli e Aljaber (1996)	m	$\sum F_j$	-	Algoritmo Genético
Fink e Voβ (2003)	m	$\sum F_j$	-	*Steepest Descent* Busca Tabu

Referências	# de Estágios	Critérios	Condições Adicionais	Abordagem
				Simulated Annealing
Aldowaisan e Allahverdi (2003)	m	C_{max}	-	3 *Simulated Annealing* 3 Algoritmos Genéticos
Allahverdi e Aldowaisan (2004)	m	$C_{max} + L_{max}$	-	*Simulated Annealing* Algoritmo Genético
Grabowski e Pempera (2005, 2007)	m	C_{max}	Algoritmos de busca tabu ou busca decrescente	5 Algoritmos de Busca Local
Kumar et al. (2006)	M	$\sum F_j$	-	Algoritmo Psico-Clonal
Pan, Tasgetiren e Liang (2007)	M	C_{max} e $\sum F_j$	-	Algoritmo DPSO
Ribeiro Filho, Nagano e Lorena (2007)	M	$\sum F_j$	-	Algoritmo Evolutivo Híbrido

Tabela 7 - Resumo das pesquisas com algoritmos meta-heurísticos para *no-wait flowshop* básico

Através da Tabela 7 acima, observa-se que a abordagem meta-heurística para o problema *no-wait flowshop* básico é bastante recente, tendo surgido em meados dos anos 1990. Nota-se um equilíbrio entre as pesquisas com critério de minimização do *makespan* e do *total flowtime.* A seguir serão realizadas considerações acerca da revisão bibliográfica deste capítulo.

3.4 Considerações

A partir da revisão bibliográfica realizada (vide Tabelas 5, 6 e 7), nota-se que a maioria das pesquisas em ambiente *no-wait flowshop* básico considera o ambiente geral de *m*-máquinas, como pode ser observado na Tabela 8 a seguir.

	Ambiente de Programação		
Abordagem	***F2***	F_m	**Total**
Exata	1	9	10
Construtiva	0	15	15
Meta-Heurística	0	8	8
Total	1	32	33

Tabela 8 - Quantidade de pesquisas encontradas por ambiente de programação e abordagem

Ao todo foram encontrados 31 trabalhos em *no-wait flowshop* básico. O total de 33 pesquisas na Tabela 8 se deve ao fato de um mesmo trabalho abordar diferentes métodos de solução. Dessas 33 pesquisas, 32 abordam o *flowshop* com *m*-máquinas e apenas 1 com duas máquinas.

O Gráfico 2 abaixo mostra uma interessante tendência observada nas pesquisas. Até os anos 1990, a maioria delas considerava como critério de minimização o *makespan*. A partir desta década, as pesquisas com o critério de minimização do *flowtime* cresceram bastante, chegando ao mesmo número das pesquisas com *makespan*. Provavelmente, esse aumento das pesquisas com

critério do *flowtime* se deveu à propagação das idéias dos conceitos modernos de produção, como a produção enxuta, que preconiza a diminuição do estoque em processamento (WIP) e o aumento da utilização da capacidade.

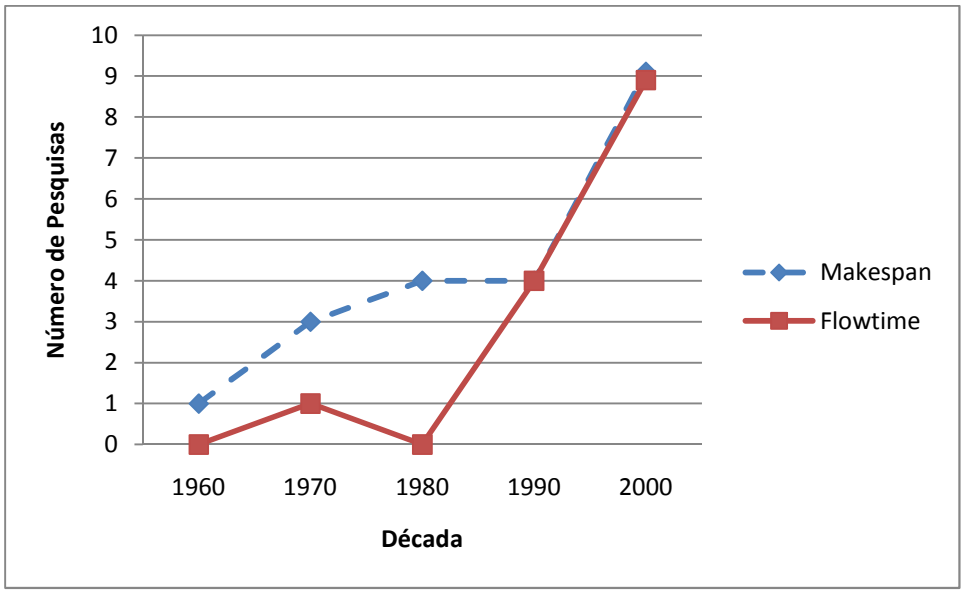

Gráfico 2 – *Makespan vs. Flowtime*

Dentre as três abordagens consideradas na revisão, aquela que possui mais pesquisas é a construtiva. Através do Gráfico 3, observa-se um grande aumento das pesquisas com métodos heurísticos (construtivos e meta-heurísticos), e uma brusca diminuição nas pesquisas com métodos de solução exata. Assim, pode-se inferir que, apesar da melhoria na tecnologia, que gera computadores mais rápidos a cada instante, a abordagem exata para este problema ainda é inviável, sendo preferível a utilização de métodos heurísticos.

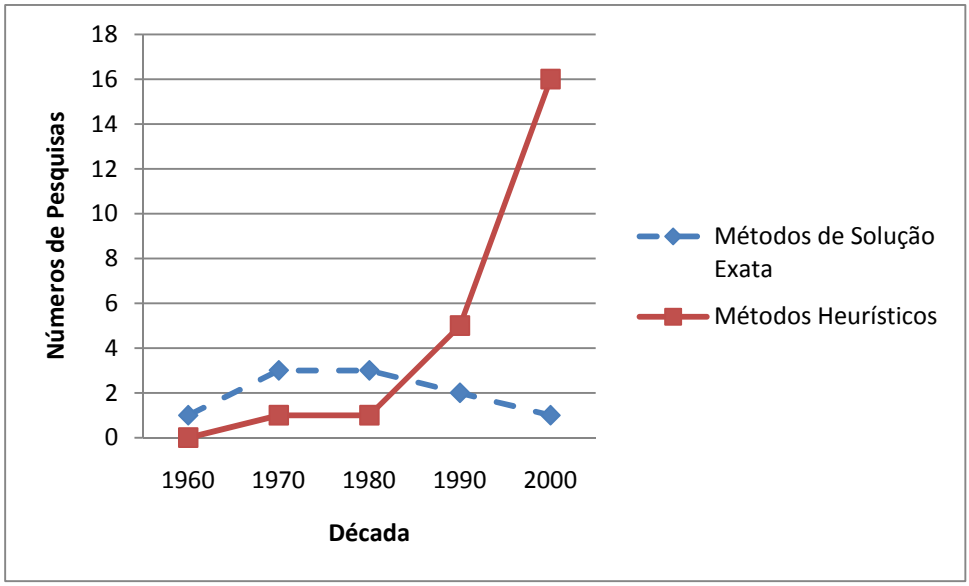

Gráfico 3 – Métodos de Solução Exata *vs.* Métodos Heurísticos

A seguir, no Capítulo 4, o problema em ambiente *no-wait flowshop* com tempos de *setup* será abordado.

CAPÍTULO 4

NO-WAIT FLOWSHOP COM TEMPOS DE *SETUP*

Quando ambas as características de *no-wait* e tempos de *setup* separados dos tempos de processamento são consideradas simultaneamente, os *setups* devem ser realizados nas máquinas antes da tarefa chegar, para satisfazer a primeira condição. Isso significa que o *setup* na máquina subseqüente ($m = i+1$) deve ser feito enquanto a tarefa está em operação na máquina precedente ($m = i$). Adicionalmente, a condição *no-wait* pode forçar atrasos na primeira máquina para garantir que a tarefa seja processada sem interrupções.

Neste trabalho, o problema de programação em ambiente *flowshop* com m máquinas será pesquisado com restrições *no-wait* e tempos de *setup* separados dos tempos de processamento e dependentes da seqüência, sendo considerado o critério de minimização da duração total da programação (*makespan*) e o critério de minimização do tempo de fluxo total (*total flowtime)*. Conforme a notação de Graham et al. (1979), estes problemas podem ser classificados respectivamente como $Fm/ST_{sd}, no-wait/C_{\max}$ e $Fm/ST_{sd}, no-wait/\sum C_j$ para o caso geral de m máquinas.

A motivação para o estudo do problema *no-wait flowshop* com tempos de *setup* dependentes da sequência vem do desenvolvimento e das possíveis aplicações de métodos heurísticos em situações reais em sistemas de produção, principalmente na indústria farmacêutica e de química fina. O caso mais simples que pode ser considerado, por exemplo, é de duas máquinas, onde cada tarefa

requer o processamento na primeira máquina (misturador e enchimento) e a embalagem na segunda máquina. Como o componente misturado pode reagir e deteriorar com a exposição ao ar, o processamento na segunda máquina deve respeitar a condição *no-wait*. Além disso, a produção de um componente químico diferente na mesma máquina possui um tempo de *setup* para a limpeza e preparação da máquina, que varia de acordo com o componente precedente, sendo assim dependente da sequência. O mesmo raciocínio se aplica a casos mais complexos de *flowshop* com *m* máquinas (BIANCO; DELL'OLMO; GIORDANI, 1999).

A seguir, nos Tópicos 4.1, 4.2 e 4.3 serão revisadas as pesquisas em ambiente *no-wait flowshop* com tempos de *setup* com métodos de solução exata, métodos heurísticos construtivos e métodos meta-heurísticos, respectivamente.

4.1 Pesquisas com Métodos de Solução Exata

Srikar e Ghosh (1986) propuseram um importante modelo MILP para o problema de *flowshop* com *setups* dependentes da sequência. Entretanto, eles cometeram alguns erros na formulação MILP, que foram posteriormente corrigidos por Stafford e Tseng (1990). Além das correções, Stafford e Tseng (1990) desenvolveram três novos modelos MILP baseados no modelo de Srikar e Ghosh (1986), com o objetivo de minimizar o tempo médio de fluxo ou o *makespan* para três tipos de problema: *flowshop* com *setups* independentes da sequência, *no-*

wait flowshop com *setups* independentes da sequência e *no-wait flowshop* com *setups* dependentes da sequência.

Gupta, Strusevich e Zwaneveld (1997) estudaram o problema de programação da produção com duas máquinas e tempos de *setup* independentes da sequência, considerando como função objetivo a minimização do *makespan.* Eles modelaram o problema como o TSP, conseguindo assim solucioná-lo em tempo polinomial.

Aldowaisan e Allahverdi (1998) abordaram o problema *no-wait flowshop* com duas máquinas e *setups* independentes da sequência com o objetivo de minimizar o *total flowtime*. Eles apresentaram um critério de eliminação, soluções ótimas para dois casos especiais e uma heurística para o caso geral, que será mencionada no próximo tópico.

Bianco, Dell'Olmo e Giordani (1999) abordaram o problema $Fm/ST_{sd},no-wait,r_j/C_{\max}$ e provaram que ele é equivalente ao problema assimétrico do caixeiro viajante com datas de liberação (*Asymmetric Travelling Salesman Problem with Ready Times* - ATSP-RT). Também mostraram que, mesmo quando as datas de liberação são iguais a zero, o problema é fortemente *NP-hard*.

Allahverdi e Aldowaisan (2000) consideraram o problema $F3/ST_{si},no-wait/\sum C_j$, encontraram soluções ótimas para três casos especiais e estabeleceram uma relação de dominância para o caso geral.

Allahverdi e Aldowaisan (2001) desenvolveram soluções ótimas para certos casos do problema $F2/ST_{sd},no-wait/\sum C_j$. Além disso, desenvolveram uma relação de dominância e apresentaram cinco heurísticas, que serão

detalhadas no tópico de meta-heurísticas. Para a experimentação computacional das heurísticas, ainda desenvolveram um algoritmo *branch-and-bound.*

Stafford e Tseng (2002) apresentaram dois modelos de programação linear inteira-mista para o problema $Fm/ST_{sd},no-wait/C_{\max}$, como já citado e detalhado na revisão sobre *setups* dependentes da sequência.

Brown, McGarvey e Ventura (2004) apresentaram dois métodos de solução em tempo não-polinomial e uma heurística em tempo polinomial para o problema $Fm/ST_{si},no-wait/\sum F_j$, também considerando o *makespan* como critério de minimização. A heurística será detalhada no tópico de heurísticas construtivas.

Dileepan (2004) adaptou para o problema $F2/ST_{si},no-wait/L_{\max}$ as condições especiais apresentadas por Aldowaisan e Allahverdi (1998) para encontrar a solução ótima do problema $F2/ST_{si},no-wait/\sum F_j$.

Fondrevelle, Allahverdi e Oulamara (2005) consideraram o mesmo problema, mas com tempos de remoção separados ($F2/ST_{si},R_{si},no-wait/L_{\max}$), e mostraram que é possível encontrar a solução ótima em tempo polinomial se algumas condições forem satisfeitas. Além disso, propuseram um algoritmo *branch-and-bound* que resolve problema de até 18 tarefas. A análise computacional mostrou que o algoritmo obtém melhores desempenhos quando os tempos de *setup* e de remoção não são muito grandes comparados ao tempo de processamento.

A Tabela 9 a seguir resume as pesquisas com métodos de solução exata do ambiente *no-wait flowshop* com tempos de *setup*.

Referências	**# de Estágios**	**Critérios**	**Condições Adicionais**	**Tempos de *Setup***	**Abordagem**
Stafford e Tseng (1990)	*m*	C_{max}, *F/*	-	ST_{sd} e ST_{si}	Correções ao artigo de Srikar e Ghosh (1986)
					Três formulações MILP
Gupta, Strusevich e Zwaneveld (1997)	2	C_{max}	-	ST_{si}	Modelo Matemático (TSP)
Aldowaisan e Allahverdi (1998)	2	$\sum F_j$	Modelos para dois casos especiais	ST_{si}	Critério de Eliminação
					Modelo Matemático
Bianco, Dell'Olmo e Giordani (1999)	*m*	C_{max}	Datas de Liberação	ST_{sd}	Provaram que o problema é *NP-hard*
					Modelo Matemático (ATSP)
Allahverdi e Aldowaisan (2000)	3	$\sum C_j$	Modelos para três casos especiais	ST_{si}	Modelo Matemático
					Relação de Dominância
Allahverdi e Aldowaisan (2001)	2	$\sum C_j$	-	ST_{sd}	Relação de Dominância
					Modelo Matemático
					Branch-and-Bound

Referências	# de Estágios	Critérios	Condições Adicionais	Tempos de *Setup*	Abordagem
Sttaford e Tseng (2002)	*m*	C_{max}	-	ST_{sd}	2 Formulações MILP
Brown, McGarvey e Ventura (2004)	*m*	C_{max} *e* $\sum F_j$	-	ST_{si}	2 Métodos de Solução em tempo não-polinomial
Dileepan (2004)	2	L_{max}	Modelo para condições especiais	ST_{si}	Modelo Matemático
Fondrevelle, Allahverdi e Oulamara (2005)	2	L_{max}	Tempos de Remoção/ Modelo matemático para condições especiais/ *Branch-and-Bound* para até 18 tarefas	ST_{si}	*Branch-and-Bound* Modelo Matemático

Tabela 9 - Resumo das pesquisas com métodos de solução exata para *no-wait flowshop* com *setups*

Como pode ser observado na Tabela 9, as pesquisas considerando ambas as características de ambiente *no-wait* e tempos de *setup* começaram a surgir recentemente, em 1990. Observa-se um predomínio das pesquisas com *setups* independentes da sequência. A seguir, o Tópico 4.2 detalhará as pesquisas com métodos heurísticos construtivos para este mesmo problema.

4.2 Pesquisas com Métodos Heurísticos Construtivos

Como já mencionado anteriormente, Aldowaisan e Allahverdi (1998), além de abordarem o problema com duas máquinas e *setups* independentes da sequência com o objetivo de minimizar o *total flowtime* com um método de solução exata, também desenvolveram uma heurística construtiva para o caso, que foi avaliada com base na porcentagem de erro em relação à solução ótima para problemas com até 10 tarefas. Os resultados mostraram que o erro cresce proporcionalmente ao número de tarefas.

Bianco, Dell'Olmo e Giordani (1999) abordaram o problema $Fm / ST_{sd}, no-wait, r_j / C_{\max}$, como já citado no tópico de métodos de solução exata. Além do modelo matemático, eles também apresentaram dois limitantes inferiores: LB1, baseado na relaxação lagrangeana do modelo matemático do problema, e LB2, baseado na relaxação da matriz de distância do ATSP, que representa o tempo gasto para o caixeiro viajante ir de um vértice a outro. Além disso, apresentaram dois métodos heurísticos, o BAH – *Best Adding Heuristic*, em que a seqüência solução é construída através da adição de tarefas no fim de uma seqüência parcial, e o BIH – *Best Insertion Heuristic*, em que a seqüência solução é construída através da inserção de tarefas em uma seqüência parcial. Na experimentação computacional, mostraram que o método BIH apresenta resultados melhores do que o BAH, apesar de possuir maior tempo computacional. O comportamento dos limitantes inferiores depende da faixa de valores das datas de liberação e do tamanho dos problemas. Em geral, nos casos mais complexos, LB1 encontra valores melhores do que LB2, e vice-versa.

Sidney, Potts e Sriskandarajah (2000) estudaram o problema $F2/ST_{si}, no-wait/C_{\max}$, onde os tempos de *setup* na segunda máquina são antecipatórios e possuem duas fases. Durante a primeira fase do *setup*, a tarefa não pode estar presente na máquina, enquanto que na segunda fase ela pode ou não estar presente. Eles propuseram um método heurístico chamado de H e, através da experimentação computacional, mostraram que encontra bons resultados, com desvio máximo da solução ótima de 33%.

Allahverdi e Aldowaisan (2000), além de considerarem o problema $F3/ST_{si}, no-wait/\sum C_j$ com métodos de solução exata, também apresentaram cinco heurísticas construtivas. Dessas, uma atribui uma tarefa por vez, duas são baseadas em várias regras de inserção e as demais são baseadas em uma única regra de inserção, que é repetida continuamente. Eles avaliaram os desempenhos das heurísticas através de experimentos computacionais, mostrando que todas obtêm bons resultados. As melhores dentre as heurísticas propostas são as de única regra de inserção.

Posteriormente, Aldowaisan (2001) abordou o problema $F2/ST_{si}, no-wait/\sum C_j$ e apresentou uma relação de dominância e uma heurística. Com o objetivo de avaliar a eficiência da heurística, comparou-a com a heurística de Aldowaisan e Allahverdi (1998) e ainda desenvolveu um limitante inferior, que foi utilizado junto das relações de dominância em um algoritmo *branch-and-bound.* A experimentação computacional mostrou a superioridade da nova heurística e da relação de dominância local.

Aldowaisan e Allahverdi (2004b) apresentaram várias heurísticas e testaram a efetividade delas através de extensiva experimentação computacional para o problema $F3/ST_{si},R_{si},no-wait/\sum C_j$. Eles também desenvolveram uma relação de dominância e um limitante inferior para o problema.

Chang, Yan e Shao (2004) abordaram o problema $FF2/ST_{si},R_{si},no-wait/C_{\max}$, onde há apenas uma máquina no primeiro estágio e *m*-máquinas paralelas no segundo estágio. Eles desenvolveram relações de dominância e propuseram duas heurísticas construtivas: H1 e H2, sendo que a segunda é semelhante à primeira, acrescida das relações de dominância. A experimentação computacional mostrou que ambas as heurísticas encontram bons resultados em tempo computacional aceitável, e que a heurística H2 é superior à H1.

Como já mencionado anteriormente, Brown, McGarvey e Ventura (2004) desenvolveram uma heurística para o problema $Fm/ST_{si},no-wait$, considerando o *total flowtime* e o *makespan* como critérios de minimização. A heurística apresentada, chamada de TRIPS, tenta quebrar o problema grande em problemas menores que podem ser enumerados facilmente: examinando todas as combinações de três tarefas possíveis, escolhe a seqüência (j_w,j_x,j_y) que minimiza o critério adotado. Para testá-la com o critério do *makespan*, a meta-heurística LKH de Helsgaun (2000) foi utilizada, e para o critério do tempo total de fluxo, foi implementado um algoritmo de *simulated annealing*. A heurística TRIPS obteve melhores soluções do que o *simulated annealing* para o critério do *total flowtime*, mas obteve piores soluções do que a meta-heurística LKH para o critério do *makespan*.

Ruiz e Allahverdi (2007a) abordaram o problema $Fm / ST_{si}, no-wait / \sum C_j$ e desenvolveram uma heurística construtiva para gerar a solução inicial de quatro meta-heurísticas, que serão mencionadas no próximo tópico. Os resultados computacionais mostraram essa heurística construtiva, quando utilizada sozinha, não obtinha melhores resultados que a heurística TRIPS de Brown, McGarvey e Ventura (2004).

Posteriormente, Ruiz e Allahverdi (2007b) estudaram o problema $Fm / ST_{si}, no-wait / L_{\max}$ e apresentaram sete heurísticas e quatro algoritmos genéticos, que serão mencionados no próximo tópico. Eles também apresentaram uma relação de dominância para o caso especial de três máquinas. Na experimentação computacional, comparam as sete heurísticas propostas com a melhor heurística de Ruiz e Allahverdi (2007a) adaptada para o caso. Os resultados mostraram que as heurísticas propostas são inferiores à heurística de Ruiz e Allahverdi (2007a).

A Tabela 10 a seguir resume as pesquisas com métodos heurísticos construtivos realizadas no ambiente *no-wait flowshop* com tempos de *setup.*

Referências	**# de Estágios**	**Critérios**	**Condições Adicionais**	**Tempos de *Setup***	**Abordagem**
Aldowaisan e Allahverdi (1998)	2	$\sum F_j$	-	ST_{si}	Heurística
Bianco, Dell'Olmo e Giordani (1999)	m	C_{max}	Datas de Liberação	ST_{sd}	2 Limitantes Inferiores 2 Heurísticas

Referências	# de Estágios	Critérios	Condições Adicionais	Tempos de *Setup*	Abordagem
Sidney, Potts e Sriskandarajah (2000)	2	C_{max}	*Setups* na segunda máquina possuem 2 partes	ST_{si}	Heurística Construtiva
Allahverdi e Aldowaisan (2000)	3	$\sum C_j$	-	ST_{si}	5 Heurísticas
Aldowaisan (2001)	2	$\sum C_j$	-	ST_{si}	Relação de Dominância Limitante Inferior Heurística
Aldowaisan e Allahverdi (2004b)	3	$\sum C_j$	Tempos de Remoção	ST_{si}	Relação de Dominância Limitante Inferior Heurística
Chang, Yan e Shao (2004)	2	C_{max}	*Flowshop* Flexível/ Tempos de Remoção	ST_{si}	Relações de Dominância 2 Heurísticas
Brown, McGarvey e Ventura (2004)	m	C_{max} e $\sum F_j$	-	ST_{si}	Heurística
Ruiz e Allahverdi (2007a)	m	$\sum C_j$	-	ST_{si}	Heurística
Ruiz e Allahverdi (2007b)	m	L_{max}	-	ST_{si}	Relação de Dominância para m=3 7 Heurísticas

Tabela 10 - Resumo das pesquisas com métodos heurísticos construtivos para *no-wait flowshop* com *setups*

Através da Tabela 10 acima, pode-se perceber que o ambiente *no-wait flowshop* com tempos de *setup* recebeu atenção considerável recentemente. Dentre os trabalhos com métodos construtivos encontrados, apenas o de Bianco,

Dell´Olmo e Giordani (1999) considera tempos de *setup* dependentes da sequência. A seguir, o Tópico 4.3 detalhará as pesquisas com métodos meta-heurísticos para este mesmo problema.

4.3 Pesquisas com Métodos Meta-Heurísticos

Allahverdi e Aldowaisan (2001) desenvolveram soluções ótimas para certos casos do problema $F2/ST_{sd},no-wait/\sum C_j$. Além disso, desenvolveram uma relação de dominância e apresentaram cinco heurísticas com a complexidade computacional de $O(n^2)$ e $O(n^3)$. As heurísticas são constituídas de duas fases; na primeira fase, uma sequência inicial é desenvolvida, e na segunda uma técnica de inserção é aplicada repetidamente para melhorar a solução encontrada. Os experimentos computacionais demonstraram que o conceito de inserções repetidas é bastante útil, e que as soluções para todas as sequências iniciais convergem para aproximadamente o mesmo valor após algumas iterações.

Shyu, Lin e Yin (2004) estudaram o problema $F2/ST_{si},no-wait/\sum C_j$ e apresentaram duas versões de um método meta-heurístico que encontra soluções aproximadas do problema baseados no método *Ant Colony Optimization* (ACO). As versões do método se diferenciam pela presença de otimizadores locais. Na experimentação computacional, compararam as duas versões do método proposto com as heurísticas propostas por Aldowaisan e Allahverdi (1998), Aldowaisan (2001) e com o modelo adaptado do TSP de

Fischetti, Laporte e Martello (1993). Os resultados mostraram a superioridade de ambas as versões do método. A versão com otimizadores locais encontra melhores resultados, porém possui tempo computacional muito superior à versão sem otimizadores.

Chang et al. (2004) derivaram duas relações de dominância para o problema $F2/ST_{si},R_{si},no-wait/\sum F_j$, onde todas as tarefas estão disponíveis no tempo zero. Eles também propuseram um algoritmo heurístico *greedy* para o problema, que se mostrou eficiente para resolver problemas com grande número de tarefas.

França, Tin e Buriol (2006) estudaram o problema em ambiente *no-wait flowshop* com *setups* dependentes da sequência e datas de liberação com o objetivo de minimizar o *makespan.* Eles desenvolveram um algoritmo genético híbrido que utiliza uma população composta por 13 agentes organizados em uma árvore ternária (i.e., estrutura de dados que se caracteriza por uma relação de hierarquia). Cada agente funciona como um recipiente que guarda duas soluções do problema. Eles ainda apresentaram um novo modelo recursivo de busca local, que é crucial ao bom desempenho do algoritmo proposto, já que representa aproximadamente 90% do tempo de processamento computacional. Na experimentação computacional, mostraram que o algoritmo proposto obtém melhores resultados do que a heurística BIH, de Bianco, Dell'Olmo e Giordani (1999), em menor tempo computacional.

Ruiz e Allahverdi (2007a) abordaram o problema $Fm/ST_{si},no-wait/\sum C_j$ e apresentaram uma relação de dominância para o caso de quatro máquinas, que também poderia ser adaptado para o caso de *m*

máquinas. Além disso, como já mencionado no tópico anterior, desenvolveram um método heurístico para gerar a solução inicial de quatro diferentes meta-heurísticas, que utilizam dois métodos de busca local estocástica, sendo um deles baseado em busca local iterativa (*Iterated Local Search* - ILS). Na experimentação computacional, testaram as cinco heurísticas propostas e os dois métodos de busca local juntamente com o algoritmo de otimização da colônia de formigas de Shyu, Lin e Yin (2004) adaptado para o caso de *m* máquinas e com a heurística TRIPS de Brown, McGarvey e Ventura (2004). Como resultado, observaram que três das cinco meta-heurísticas propostas se mostraram superiores aos algoritmos de Shyu, Lin e Yin (2004) e Brown, McGarvey e Ventura (2004), sendo que uma delas se mostrou superior às demais, principalmente quando utilizada conjuntamente ao método ILS.

Como já mencionado, Ruiz e Allahverdi (2007b) desenvolveram quatro algoritmos genéticos para o problema $Fm/ST_{si}, no-wait/L_{\max}$. Para testá-los, os autores compararam-nos com os dois métodos de busca local estocástica propostos por Ruiz e Allahverdi (2007a) adaptados para o caso. Os quatro métodos propostos se mostraram superiores aos de Ruiz e Allahverdi (2007a).

A Tabela 11 a seguir resume as pesquisas realizadas no ambiente *no-wait flowshop* com tempos de *setup* dependentes da sequência.

Referências	# de Estágios	Critérios	Condições Adicionais	Tempos de *Setup*	Abordagem
Allahverdi e Aldowaisan (2001)	*2*	$\sum C_j$	-	ST_{sd}	Relação de Dominância 5 Heurísticas (busca de inserção iterativa)
Shyu, Lin e Yin (2004)	2	$\sum C_j$	-	ST_{si}	2 Heurísticas ACO
Chang et al. (2004)	2	$\sum F_j$	Tempos de Remoção	ST_{si}	2 Relações de Dominância Algoritmo *Greedy*
França, Tin e Buriol (2006)	*m*	C_{max}	Datas de Liberação	ST_{sd}	Algoritmo Genético Híbrido Procedimento de Busca Local
Ruiz e Allahverdi (2007a)	*m*	$\sum C_j$	-	ST_{si}	2 Algoritmos de Busca Local Estocástica 4 Heurísticas Relação de Dominância
Ruiz e Allahverdi (2007b)	*m*	L_{max}	-	ST_{si}	4 Algoritmos Genéticos

Tabela 11 - Resumo das pesquisas com métodos meta-heurísticos para *no-wait flowshop* com *setups*

Através da Tabela 11 acima, observa-se um pequeno número de pesquisas com métodos meta-heurísticos, comparando-se com as pesquisas com métodos construtivos (Tabela 10). Isso pode ser justificado pelo fato da primeira abordagem meta-heurística para esse problema ser muito recente, do início dos anos 2000. Ademais, observa-se que há mais pesquisas com tempos de *setup* independentes do que dependentes da sequência. A seguir, no Tópico 4.4 serão realizadas considerações acerca da revisão bibliográfica deste capítulo.

4.4 Considerações

A partir da revisão bibliográfica realizada (vide Tabelas 9, 10 e 11), nota-se que, diferentemente dos ambientes *flowshop* com tempos de *setup* dependentes da sequência (vide Tabela 4) e *no-wait flowshop* básico (vide tabela 8), há empate entre as pesquisas com duas e *m*-máquinas, como pode ser observado na Tabela 12 a seguir. Ademais, os critérios de minimização mais abordados são o *makespan* e o *total flowtime*.

	Ambiente de Programação			
Abordagem	***F2***	***F3***	***Fm***	**Total**
Exata	5	1	4	10
Construtiva	4	1	5	10
Meta-Heurística	3	0	3	6
Total	12	2	12	26

Tabela 12 - Quantidade de pesquisas encontradas por ambiente de programação e abordagem

Ao todo foram encontrados 19 trabalhos em *no-wait flowshop* com *setups* dependentes da sequência. Na Tabela 12, o total de 26 pesquisas se deve ao fato de um mesmo trabalho abordar diferentes métodos de solução. Dessas 26 pesquisas, 12 abordam o *flowshop* com *m*-máquinas, 12 com duas máquinas e 2 com três máquinas. Comparando-se com os outros ambientes estudados, observa-se que o *flowshop* com ambas características de *no-wait* e tempo de *setup* possui poucos trabalhos, o que se deve ao fato do tratamento científico desse ambiente

ser recente. Os conceitos modernos de gestão da produção, tais como o JIT e a produção enxuta, concomitante ao atual crescimento da indústria de química fina provavelmente despertaram o interesse científico nesse ambiente. Através do Grafico 4 abaixo, pode-se verificar o crescimento no interesse por esse ambiente, além de uma tendência no emprego dos métodos heurísticos (construtivos e meta-heurísticos).

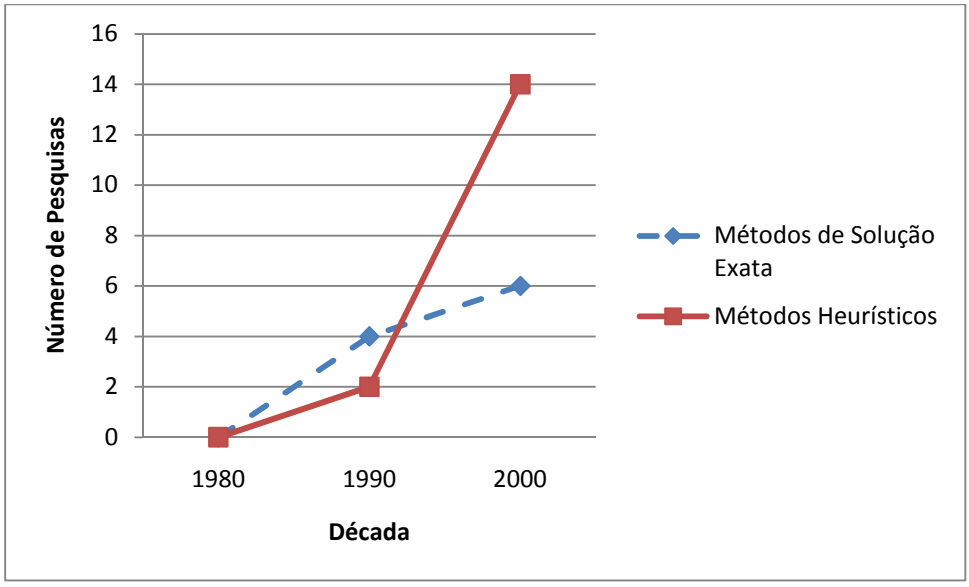

Gráfico 4 - Número de Pesquisas *vs.* Década

Dentre as três abordagens consideradas na revisão, estão empatadas em número as pesquisas com métodos exatos e heurísticos construtivos. As pesquisas com métodos meta-heurísticos possuem menor número. Entretanto, observa-se no Gráfico 4 uma tendência no aumento de pesquisas heurísticas, mostrando que as pesquisas com métodos exatos eram em maior número do que as pesquisas com métodos heurísticos em 1990, tendo essa situação se invertido em 2000.

É também interessante notar que há um total de 20 pesquisas que estudaram o ambiente *no-wait flowshop* com tempo de *setup* independente da sequência, enquanto há apenas 7 para tempo de *setup* dependente da sequência. A Tabela 13 abaixo mostra a relação entre o tipo do *setup* e a abordagem das pesquisas. A abordagem exata é a que mais possui pesquisas para o ambiente *no-wait* com *setups* dependentes da sequência. A abordagem construtiva possui apenas um trabalho com *setups* dependentes.

	Tempo de *Setup*	
Abordagem	***STsi***	***STsd***
Exata	7	4
Construtiva	9	1
Meta-Heurística	4	2
Total	20	7

Tabela 13 – Tempo de *Setup vs.* Abordagem

A seguir, no Capítulo 5, será definido o problema abordado nesta dissertação, *no-wait flowshop* com tempos de *setup* dependentes da sequência com critérios de minimização do *total flowtime* e do *makespan*.

CAPÍTULO 5

NOTAÇÃO E DEFINIÇÃO DO PROBLEMA

Um problema de programação *no-wait flowshop* com tempos de *setup* dependentes da sequência consiste em um conjunto J= $\{j_1, j_2, j_3, \dots, j_n\}$ de n tarefas a serem processadas em um conjunto $M = \{m_1, m_2, m_3, \dots, m_m\}$ de m máquinas, sendo que cada máquina processa apenas uma tarefa por vez. Uma determinada tarefa j_i é composta por m operações $(op_{1i}, \dots, op_{ki}, op_{k+1i}, \dots, op_{mi})$, a serem executadas em sequência e sem interrupção, para satisfazer a condição *no-wait*. A operação op_{ki} deve ser executada na máquina k, com tempo de processamento p_{ki}.

Para executar a operação op_{kj} na máquina k, existe um tempo de *setup* dependente da sequência (s_{ij}^{k}), se a operação op_{ki} for processada imediatamente antes da operação op_{kj}.

Neste trabalho, duas medidas de desempenho serão consideradas, o *makespan* e o *total flowtime*, detalhadas a seguir.

5.1 Critério de Minimização do *Makespan*

O tempo gasto para processar todas as tarefas desde o início da primeira operação da primeira tarefa até o fim da última operação da última tarefa

processada na seqüência é chamado de *makespan*. Esta medida de desempenho está relacionada à eficiente utilização dos recursos, ou seja, à redução do tempo da duração total da programação.

A expressão para o cálculo do *makespan* (C_{max}), segundo Bianco, Dell'Olmo e Giordani (1999), é ilustrada a seguir:

$$C_{max}=t_{[n]}+\sum_{k=1}^{m} p_{k[n]} \tag{1}$$

Onde *[n]* é a tarefa de *J* que ocupa a posição *n* em σ e $t_{[i]}$ é a data mais cedo de início da tarefa *[i]*, calculada através da expressão:

$$t_{[i]} = t_{[i-1]} + \max_{1\le k\le m}\left[s^{[k]}_{[i-1][i]} + \sum_{h=1}^{k}\left(p_{h[i-1]} - p_{h[i]}\right) + p_{k[i]}\right] \tag{2}$$

A Figura 3 a seguir mostra um exemplo do problema $F_m/ST_{sd}, no-wait/C_{max}$. Em particular, representa-se o problema com 2 máquinas e 3 tarefas, com duas soluções factíveis, sendo uma delas a solução ótima.

s^1_{ij}	j_1	j_2	j_3
j_1	-	2	4
j_2	3	-	1
j_3	3	2	-
p_{1i}	5	5	10

s^2_{ij}	j_1	j_1	j_1
j_1	-	1	2
j_2	2	-	3
j_3	4	3	-
p_{2i}	7	5	8

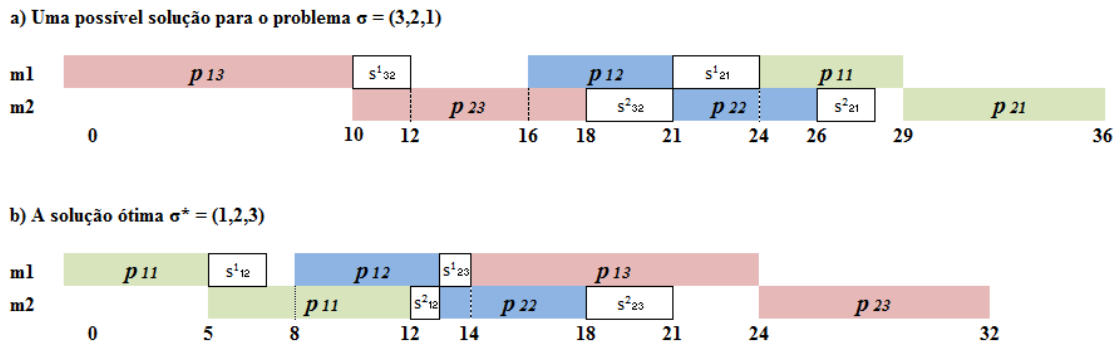

Figura 3 - Um exemplo do problema de programação $F2/ST_{sd}, no-wait/C_{max}$

5.2 Critério de Minimização do *Total Flowtime*

O *flowtime* de uma tarefa j_i é o tempo entre o momento que a tarefa está disponível para ser processada e o momento em que ela é completada, correspondendo, então, ao tempo em que a tarefa permanece no centro de trabalho.

A expressão para o cálculo do *flowtime* (F_i) em ambiente *no-wait* é ilustrada a seguir:

$$F_{[i]} = t_{[i]} + \sum_{k=1}^{m} p_{k[i]} \tag{3}$$

onde $t_{[i]}$ é a hora mais cedo de início da tarefa *[i]*, calculada através da expressão (2) acima.

O *total flowtime* $\left(\sum F_j\right)$ está relacionado à rápida resposta à demanda e é calculado de acordo com a expressão (4), que equivale à expressão (5).

$$\sum F_j = \sum_{i=1}^{n} F_{[i]} \tag{4}$$

$$\sum F_j = \sum_{i=2}^{n} t_{[i]} + \sum_{i=1}^{n} \sum_{k=1}^{m} p_{k[i]} \tag{5}$$

A Figura 4 a seguir mostra um exemplo do problema $F_m/ST_{sd}, no-wait/\sum F_j$. Em particular, representa-se o problema com 3 máquinas e 5 tarefas, com duas soluções factíveis, sendo uma delas a solução ótima.

s^1_{ij}	j_1	j_2	j_3	j_4	j_5
j_1	-	1	3	2	3
j_2	1	-	2	4	2
j_3	2	2	-	5	1
j_4	3	2	3	-	7
j_5	1	2	3	4	-
p_{1i}	5	8	4	5	6

s^2_{ij}	j_1	j_2	j_3	j_4	j_5
j_1	-	1	3	2	3
j_2	1	-	2	4	2
j_3	2	2	-	5	1
j_4	3	2	3	-	7
j_5	1	2	3	4	-
p_{2i}	6	7	3	7	5

s^3_{ij}	j_1	j_2	j_3	j_4	j_5
j_1	-	1	3	2	3
j_2	1	-	2	4	2
j_3	2	2	-	5	1
j_4	3	2	3	-	7
j_5	1	2	3	4	-
p_{3i}	7	2	8	6	5

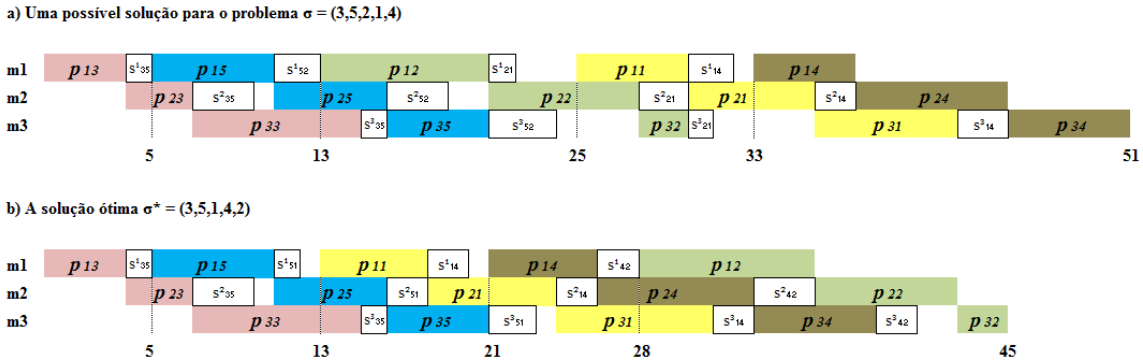

Figura 4 - Um exemplo do problema de programação $F3/ST_{sd}, no-wait/\sum F_j$

CAPÍTULO 6

ANÁLISE DOS MÉTODOS HEURÍSTICOS

Nesta pesquisa serão analisados os métodos BAH e BIH, de Bianco, Dell'Olmo e Giordani (1999) e o método TRIPS, de Brown, McGarvey e Ventura (2004). Como já mencionado, os métodos BAH e BIH são os únicos métodos construtivos encontrados na literatura para o problema NWFS com *setups* dependentes da sequência, e por isso serão utilizados na experimentação computacional. Já o método TRIPS é o melhor método construtivo encontrado na literatura para o NWFS com *setups* independentes da sequência, e por isso será adaptado para o NWFS com *setups* dependentes da sequência e utilizado na experimentação computacional. É importante ressaltar que, na experimentação computacional do método TRIPS, ele se mostrou superior a um *simulated annealing* para o critério de *total flowtime*, mas se mostrou inferior à meta-heurística LKH para o critério do *makespan* em relação à qualidade das soluções. Porém, não é usual a comparação de heurísticas construtivas com meta-heurísticas, já que essas normalmente encontram melhores soluções, mas requerem tempo computacional muito maior, por possuírem estruturas diferentes. A meta-heurística LKH, por exemplo, se mostrou 20 vezes mais lenta do que a heurística TRIPS. Assim, a heurística TRIPS será utilizada na experimentação computacional tanto para o critério do *total flowtime* quanto do *makespan*.

6.1 Métodos Heurísticos BAH e BIH

Este é o único trabalho presente na literatura que apresenta métodos construtivos para o problema *no-wait flowshop* com tempos de *setup* dependentes da seqüencia com o objetivo de minimizar o *makespan*: o BAH – *Best Adding Heuristic* e o BIH – *Best Insertion Heuristic*. Para fins de comparação, ambos métodos serão adpatados também para o critério de minimização do *total flowtime*.

6.1.1 Heurística BAH

O método BAH encontra uma sequência factível em *n* iterações. Em cada iteração, dada uma sequência parcial de tarefas computadas na iteração anterior, ele examina as tarefas ainda não programadas na última posição da sequência parcial, alocando a que resultar no menor *makespan*.

A seguir são apresentados seus respectivos passos:

Dado um conjunto $J = \{j_1, j_2, j_3, \dots, j_n\}$ de n tarefas, σ o conjunto das tarefas programadas e U o conjunto das tarefas não-programadas.

Passo 1: U←J; σ ←∅.

Passo 2: Enquanto U ≠ ∅, fazer:

Passo 2.1: Escolha a tarefa $j_i \in U$ que, quando programada na última posição da sequência σ, obtém o menor *makespan* da sequência parcial.

Passo 2.2: Insira a tarefa j_i na última posição da sequência σ .

Passo 2.3: $U \leftarrow U - j_i$.

6.1.2 Heurística BIH

O método BIH encontra uma sequência de *n* tarefas em *n* iterações. A cada iteração ele considera uma sequência das tarefas já programadas e encontra a melhor sequência obtida inserindo cada tarefa não-programada em todas posições possíveis da sequência.

A seguir são apresentados seus respectivos passos:

Dado um conjunto $J = \{j_1, j_2, j_3, \dots, j_n\}$ de n tarefas, σ o conjunto das tarefas programadas, U o conjunto das tarefas não-programadas e *h* a posição relativa de inserção.

Passo 1: $U \leftarrow J$; $\sigma \leftarrow \emptyset$; $h \leftarrow 0$.

Passo 2: Enquanto $U \neq \emptyset$, fazer:

Passo 2.1: Escolha a tarefa $j_i \in U$ que, quando inserida na sequência σ, obtém o menor *makespan* da sequência parcial, sendo *h* a posição relativa de inserção.

Passo 2.2: Insira a tarefa j_i na posição *h* da sequência σ .

Passo 2.3: U←U-j_i.

6.2 Heurística TRIPS

Os autores Brown, McGarvey e Ventura (2004) apresentaram a heurística construtiva TRIPS para o problema *no-wait flowshop* com tempos de *setup* independentes da sequência, para minimização do *total flowtime* ou do *makespan*. Essa heurística, para fins de comparação, será adaptada para o caso de tempos de *setup* dependentes da sequência.

A heurística TRIPS realiza uma tentativa de quebrar o problema maior em problemas menores que podem ser enumerados facilmente. O método examina todas as combinações de três tarefas possíveis do conjunto de tarefas *J* e escolhe a sequência $\{j_w j_x j_y\}$ que minimiza a medida de performance. A tarefa j_w é então colocada na última posição vazia de uma nova seqüência σ e é removida do conjunto de tarefas não-sequenciadas. Repetindo este processo, programa-se uma nova tarefa em σ a cada trio examinado, até que restem somente três tarefas. Seleciona-se a sequência ótima para essas tarefas e as mesmas são colocadas nas posições finais da sequência σ.

O método TRIPS é composto pelos seguintes passos:

Dado um conjunto J = $\{j_1, j_2, j_3, \dots, j_n\}$ de n tarefas, σ o conjunto das tarefas programadas e *U* o conjunto das tarefas não-programadas.

Passo 1: U←J; σ ←∅; *h*←0.

Passo 2: Enquanto $h<n-2$, fazer:

Passo 2.1: Dado que as primeiras h tarefas são designadas na seqüência σ, compare todos os trios de tarefas ordenados de U;

Passo 2.2: Escolha o trio $\{j_w, j_x, j_y\}$ que, inserido nas posições $h+1$, $h+2$, $h+3$, respectivamente, da seqüência σ, minimize a medida de performance (*makespan* ou *flowtime*) e insira j_w na posição $h+1$ de σ.

Passo 2.3: $h \leftarrow h+1$; $U \leftarrow U-j_w$;

Passo 3: Insira j_x e j_y nas duas últimas posições, respectivamente, de σ.

CAPÍTULO 7

MÉTODOS HEURÍSTICOS PROPOSTOS

7.1 Heurística para Minimização do *Makespan*

Wismer (1972) apresentou um algoritmo para o problema *no-wait flowshop* que buscava minimizar os *delays*, ou seja, os atrasos necessários na primeira máquina para satisfazer a condição *no-wait*. Baseando-se nesse algoritmo e retrabalhando as expressões (1) e (2), demonstradas por Bianco, Dell'Olmo e Giordani (1999), foi aplicada uma propriedade estrutural, apresentada a seguir para sua aplicação no método proposto.

7.1.1 Análise da propriedade estrututal para a nova heurística

Seja σ uma sequência de *J*, $j_{[i]}$ é a tarefa de *J* que ocupa a *i*-ésima posição em σ. O intervalo de tempo entre o início da tarefa $j_{[i+1]}$ e o início da tarefa $j_{[i]}$ na máquina k é $\Delta t^{k}_{[i][i+1]}$, calculado conforme as expressões (6) e (7) abaixo:

$$\Delta t^{1}_{[i][i+1]} = \max_{1 \le k \le m} [s^{k}_{[i][i+1]} + \sum_{h=1}^{k} (p_{h[i]} - p_{h[i+1]}) + p_{k[i+1]}] \qquad (6)$$

$$\Delta t^{k}_{[i][i+1]} = \Delta t^{1}_{[i][i+1]} + \sum_{h=1}^{k-1} \left(p_{h[i+1]} - p_{h[i]} \right) \tag{7}$$

Definindo $GAP^{k}_{[i][i+1]}$ como o intervalo de tempo entre o término da tarefa $j_{[i]}$ e o início da tarefa $j_{[i+1]}$ na máquina k, o mesmo pode ser obtido pela expressão (8) abaixo.

$$GAP^{k}_{[i][i+1]} = \Delta t^{k}_{[i][i+1]} - p_{k[i]} \tag{8}$$

O *GAP* da primeira tarefa da sequência na máquina k é definido pela expressão (9):

$$GAP^{k}_{[0][1]} = \sum_{h=1}^{k-1} p_{h[1]} \tag{9}$$

A Figura 5 ilustra o $GAP^{2}_{[2][3]}$, ou seja, o intervalo de tempo entre o término da tarefa j_2 e o início da tarefa j_3 na máquina 2.

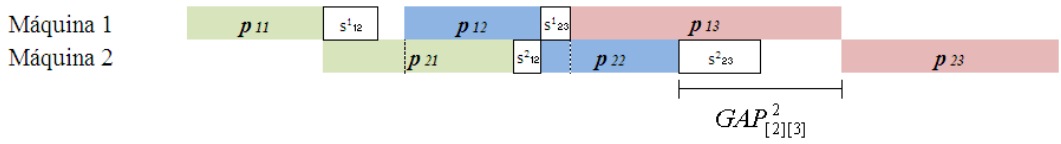

Figura 5 - Exemplo do cálculo do *GAP*

7.1.2 Método heurístico proposto

No processo de criação da heurística, verificou-se que o objetivo de minimizar o *makespan* coincidia com a minimização dos *GAP*s apresentados na

propriedade estrutural. Assim, quanto menores os *GAPs* de uma programação, menor será sua duração total. Então, a partir dessa idéia de minimizar os *GAPs*, foram desenvolvidos e testados vários métodos, até que se chegou a um método de quatro fases, sendo descrito a seguir.

Para a apresentação do método, faz-se necessário definir alguns conceitos. Define-se a duração da programação em uma máquina *k* como o "custo total da programação na máquina *k*". Assim, o custo total da programação compreende o somatório dos *GAP*s da máquina *k* e dos tempos de processamento das operações sequênciadas na máquina *k*. Nota-se que o custo total da programação na última máquina é equivalente ao *makespan* (ver Figura 6). Define-se o "*GAP* total" como a soma dos *GAPs* em todas as máquinas (ver Figura 7).

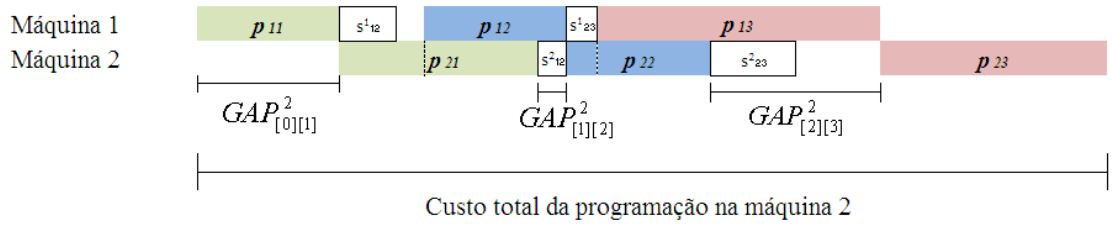

Figura 6 - Exemplo do cálculo do custo total da programação

O método proposto será denominado GAPH, sendo composto pelos seguintes passos:

Dado um conjunto J= $\{j_1, j_2, j_3, \ldots, j_n\}$ de n tarefas, U o conjunto das tarefas não-programadas e $\sigma_x = (j_{[1]}, j_{[2]}, \ldots, j_{[n]})$ a sequência-solução, onde $x = \{1,2,3,4\}$, calcule o $GAP^k_{[i][j]}$ de cada tarefa $i = 1, ..., n$ para cada tarefa $j = 1, ..., n$, se $i \neq j$, em todas as m máquinas.

Passo 1: $U \leftarrow J$; $\sigma_1 \leftarrow \emptyset$;

Passo 2: Enquanto $U \neq \emptyset$, fazer:

Passo 2.1: Calcule o custo total na última máquina para todas as possíveis inserções de cada tarefa $j_i \in U$ em σ_1, sendo h a posição relativa de inserção.

Passo 2.2: Escolha a tarefa $j_i \in U$ que forneça o menor custo total na última máquina.

Passo 2.3: Insira esta tarefa na posição h da sequência σ_1

Passo 2.4: $U \leftarrow U - j_i$;

Passo 3: $U \leftarrow J$; ; $\sigma_2 \leftarrow J$;

Passo 4: Enquanto $U \neq \emptyset$, fazer:

Passo 4.1: Calcule o GAP total para todas as possíveis inserções de cada tarefa $j_i \in U$ em σ_2, sendo h a posição relativa de inserção.

Passo 4.2: Escolha a tarefa $j_i \in U$ que forneça o menor GAP total.

Passo 4.3: Insira esta tarefa na posição h da sequência σ_2.

Passo 4.4: $U \leftarrow U - j_i$;

Passo 5: $U \leftarrow J$; $\sigma_3 \leftarrow J$;

Passo 6: Enquanto $U \neq \emptyset$, fazer:

Passo 6.1: Calcule o GAP na última máquina para todas as possíveis inserções de cada tarefa $j_i \in U$ em σ_3, sendo h a posição relativa de inserção.

Passo 6.2: Escolha a tarefa $j_i \in U$ que forneça o menor GAP na última máquina.

Passo 6.3: Insira esta tarefa na posição h da sequência σ_3.

Passo 6.4: $U \leftarrow U - j_i$

Passo 7: $U \leftarrow J$; $\sigma_4 \leftarrow J$;

Passo 8: Enquanto $U \neq \emptyset$, fazer:

Passo 8.1: Calcule, para todas as possíveis inserções de cada tarefa $j_i \in U$ em σ_4, a soma dos *GAPs* com a duração da tarefa na última máquina, sendo h a posição relativa de inserção.

Passo 8.2: Escolha a tarefa $j_i \in U$ que forneça a menor soma dos *GAP*s com a duração da tarefa na última máquina.

Passo 8.3: Insira esta tarefa na posição h da sequência σ_4.

Passo 8.4: $U \leftarrow U - j_i$

Passo 9: Calcular o *makespan* para as sequências $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ e escolher a sequência de menor *makespan*.

As Figuras 7 e 8 ilustram os cálculos do método GAPH.

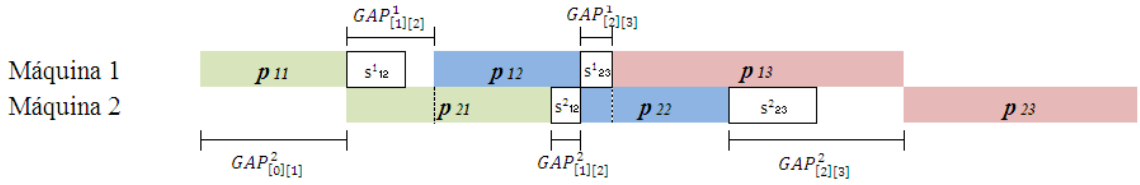

Figura 7 - Ilustração de todos os GAPs da programação

Na Figura 7, o *GAP* total é: $GAP^1_{[1][2]} + GAP^1_{[2][3]} + GAP^2_{[0][1]} + GAP^2_{[1][2]} + GAP^2_{[2][3]}$.

A soma dos *GAPs* na última máquina, calculada no Passo 6.1, é: $GAP^2_{[0][1]} + GAP^2_{[1][2]} + GAP^2_{[2][3]}$.

A soma dos *GAPs* na última máquina com o tempo de processamento da tarefa a ser inserida na última máquina (Passo 8.1) é: $GAP^2_{[0][1]} + GAP^2_{[1][2]} + GAP^2_{[2][3]} + p_{23}$.

A Figura 8 ilustra o funcionamento do método. No exemplo, as tarefas j_1 e j_2 já foram programadas ($\sigma = (j_1, j_2)$) e a tarefa j_3 é programada nas possíveis posições da sequência (h).

Como pode-se observar na Figura 8, quando a tarefa j_3 foi inserida na posição $h = 3$, obteve-se o menor *makespan*. Logo, a sequência selecionada pelo método seria a terceira ($\sigma = (j_1, j_2, j_3)$).

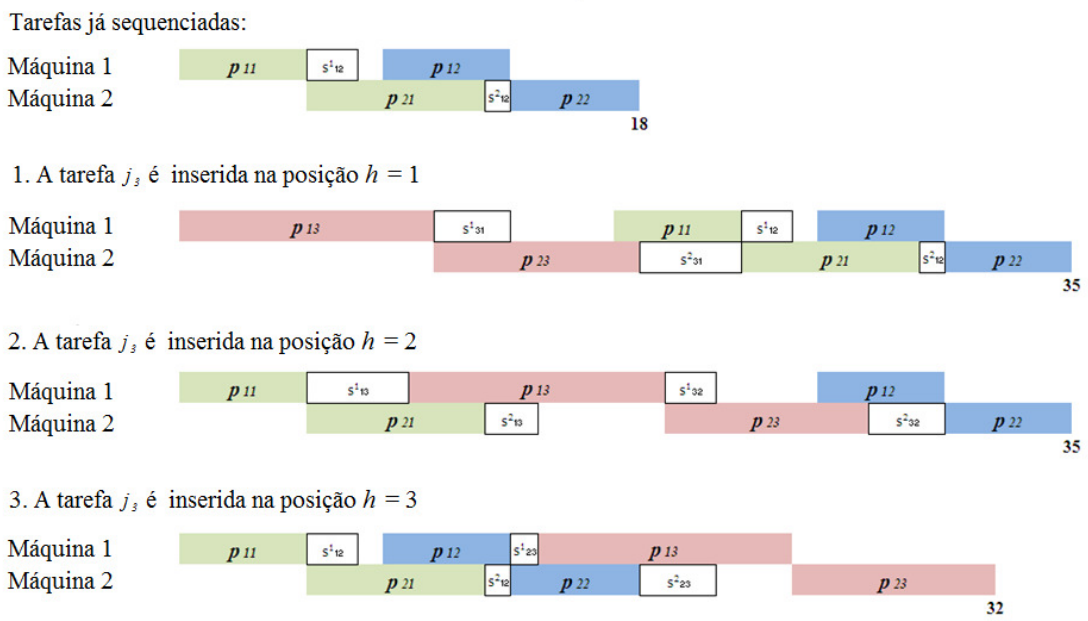

Figura 8 - Exemplo numérico do funcionamento do método

7.2 Heurística para Minimização do *Total Flowtime*

Brown, McGarvey e Ventura (2004) apresentaram a heurística TRIPS, detalhada anteriormente, que quebra o problema em trios, i.e., conjunto de três

tarefas, diminuindo assim o esforço computacional necessário para obtenção da solução. Baseado nessa idéia de tentar quebrar o problema grande em problemas menores, de mais fácil solução, foram testados métodos que quebravam o problema em duas, quatro e cinco tarefas. A partir do *trade-off* "qualidade de solução *vs.* esforço computacional", foi selecionado o método que quebrava o problema em quatro tarefas. Então, foi desenvolvida uma nova heurística que utilizava esse método, que recebeu o nome de QUARTS – abreviação para "quartetos", que remete ao nome da heurística TRIPS – abrevição para "*triplets*" (trios em inglês). A heurística QUARTS será descrita a seguir.

7.2.1 Método heurístico proposto

Para escolher as duas primeiras tarefas da sequência, a heurística QUARTS examina todas as combinações de quatro tarefas possíveis do conjunto de tarefas *J* e escolhe a sequência $\{j_w, j_x, j_y, j_z\}$ que minimiza o *total flowtime*. As tarefa j_w e j_x são então colocadas na primeira e segunda posições da sequência σ, respectivamente, e são removidas do conjunto de tarefas não-sequenciadas. Na próxima iteração, a tarefa sequenciada na segunda posição de σ será a primeira posição (j_w) de todas as combinações de tarefas do quarteto $\{j_w, j_x, j_y, j_z\}$. A tarefa na segunda posição (j_x) do quarteto que obtiver menor *total flowtime* é sequenciada na próxima posição vazia da sequência σ e é removida do conjunto de tarefas não-sequenciadas. Repetindo este processo, programa-se uma nova tarefa em σ a cada iteração, sendo a tarefa sequenciada na iteração anterior sempre a primeira posição dos quartetos da iteração atual, até que restem somente três tarefas.

Seleciona-se a sequência ótima para estas tarefas e as mesmas são colocadas nas posições finais da sequência σ. Em seguida, o método busca melhorar a sequência através da combinação de duas buscas nas vizinhanças, sendo uma busca de inserção e uma busca de permutação, descritas a seguir:

- Busca de inserção na vizinhança:

A busca de inserção procura inserir cada tarefa j_i da sequência σ nas $(n\text{-}1)^2$ posições, sendo n o número de tarefas do problema. A inserção que trouxer o menor *total flowtime* é então escolhida.

A Figura 9 a seguir apresenta um exemplo de inserção: dada a sequência $\sigma = \{j_3, j_5, j_2, j_1, j_4\}$, a tarefa j_2 é inserida na sua quinta posição, resultando na sequência $\sigma' = \{j_3, j_5, j_1, j_4, j_2\}$.

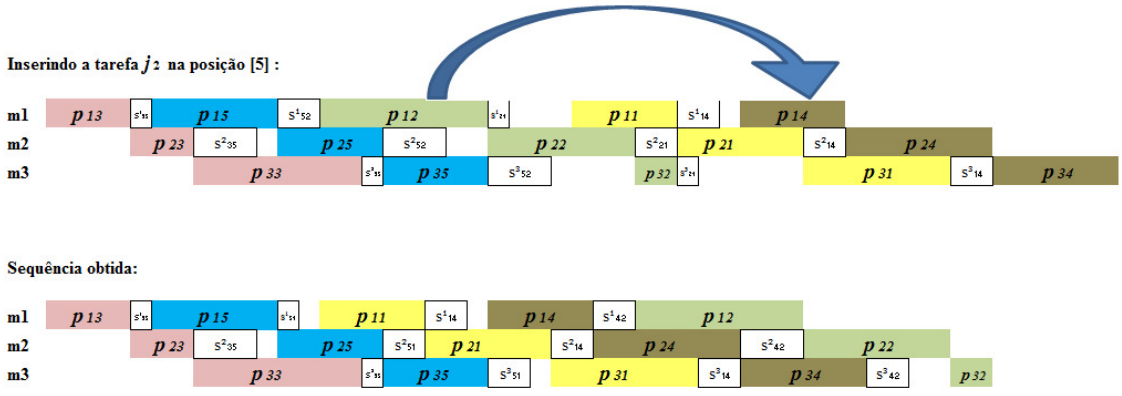

Figura 7 – Exemplo da busca de inserção

- Busca de permutação na vizinhança:

Esta busca realiza trocas de pares de tarefas da sequência σ de forma ordenada, nas $\frac{n(n-1)}{2}$ posições, sendo n o número de tarefas do problema. A troca que trouxer o menor *total flowtime* será escolhida.

A Figura 10 a seguir apresenta um exemplo de busca de permutação: dada a sequência $\sigma = \{j_3, j_5, j_2, j_1, j_4\}$, a tarefa j_2 é trocada com a tarefa j_1, resultando na sequência $\sigma' = \{j_3, j_5, j_1, j_2, j_4\}$.

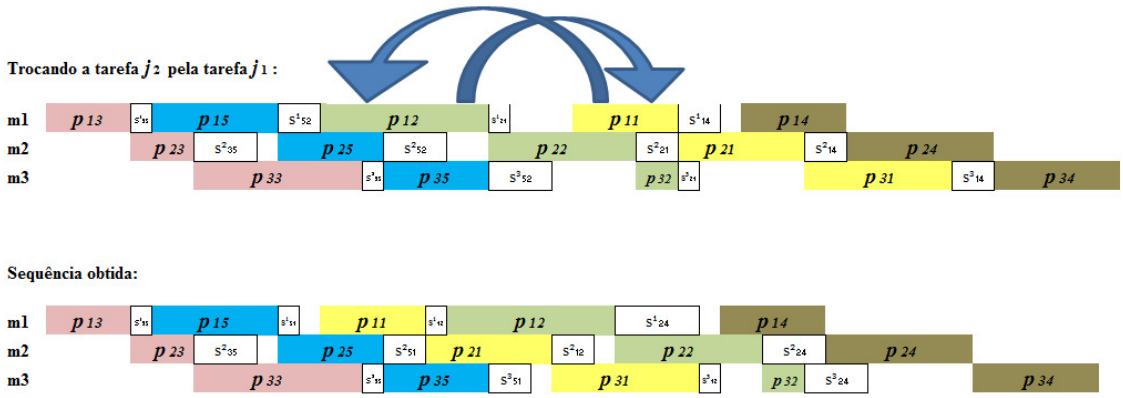

Figura 8 – Exemplo da busca de permutação

A seguir são apresentados os passos da heurística QUARTS:

Dado um conjunto $J = \{j_1, j_2, j_3, \ldots, j_n\}$ de n tarefas, σ o conjunto das tarefas programadas e U o conjunto das tarefas não-programadas.

Passo 1: U←J; σ, σ',σ''←∅ ; h←1.

Passo 2:

Passo 2.1: Calcule o *total flowtime* para todos os quartetos de tarefas ordenados de *U*;

Passo 2.2: Escolha o quarteto $\{j_w, j_x, j_y, j_z\}$ de menor *flowtime* e insira j_w e j_x nas posições 1 e 2 de σ, respectivamente;

Passo 2.3: $h \leftarrow h+2$; $U \leftarrow U\text{-}j_w$; $U \leftarrow U\text{-}j_x$.

Passo 3: Enquanto h<n-2, fazer:

Passo 3.1: $j_w \leftarrow j_{[h-1]}$;

Passo 3.2: Calcule o *total flowtime* para todos os quartetos $\{j_w, j_x, j_y, j_z\}$ de tarefas de U;

Passo 3.3: Escolha o quarteto de menor *flowtime* e insira j_x na posição h de σ;

Passo 3.4: $h \leftarrow h+1$; $U \leftarrow U\text{-}j_x$;

Passo 4: Insira j_y e j_z nas duas últimas posições, respectivamente, de σ. Compute o *total flowtime*.

Passo 5: Busca de inserção: considerando toda a vizinhança de inserção da sequência σ, determine a sequência σ' que minimize o *total flowtime*. Caso o *total flowtime* da sequência σ' seja menor do que o da sequência σ, então σ← σ'.

Passo 6: Busca de permutação: considerando toda a vizinhança de permutação da sequência σ, determine a sequência σ'' que minimize o *total flowtime*. Caso o *total flowtime* da sequência σ'' seja menor do que o da sequência σ, então σ← σ''.

Baseando-se no exemplo apresentado pela Figura 4, as Figuras 11, 12 e 13 a seguir ilustrarão o funcionamento do método proposto.

Primeiramente, calcula-se o *total flowtime* de todos os quartetos $\{j_w, j_x, j_y, j_z\}$. Em seguida, seleciona-se o de menor *total flowtime*. A Figura 11 mostra o quarteto selecionado ($\{j_3, j_5, j_1, j_2\}$).

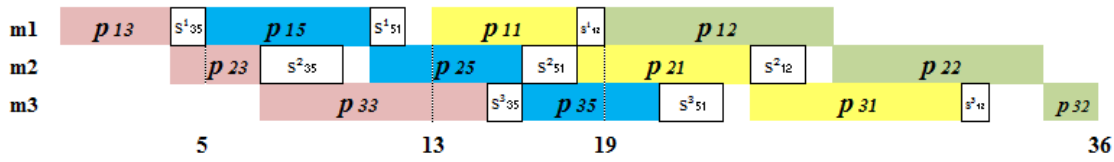

Figura 9 – Primeiro quarteto selecionado

A primeira e a segunda tarefa do quarteto $\{j_3,j_5,j_1,j_2\}$ são então inseridas na primeira e segunda posição da sequência de tarefas programadas (σ), e retiradas do conjunto de tarefas de não programadas (U). Na próxima iteração, a tarefa j_5 será a primeira posição dos quartetos a serem calculados.

Após calcular o *total flowtime* de todos os quartetos $\{j_5,j_x,j_y,j_z\}$, seleciona-se o de menor *total flowtime.* A Figura 12 mostra o quarteto selecionado ($\{j_5,j_1,j_4,j_2\}$).

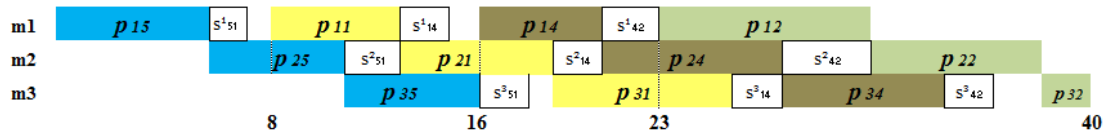

Figura 10 – Segundo quarteto selecionado

Como restam apenas três tarefas a serem programadas, todas elas serão inseridas nas últimas posições de σ. Assim, j_1, j_4 *e* j_2 serão programadas na terceira, quarta e quinta posições de σ, respectivamente, e retiradas de U. Logo, a sequência final será $\{j_3,j_5,j_1,j_4,j_2\}$, ilustrada na Figura 13.

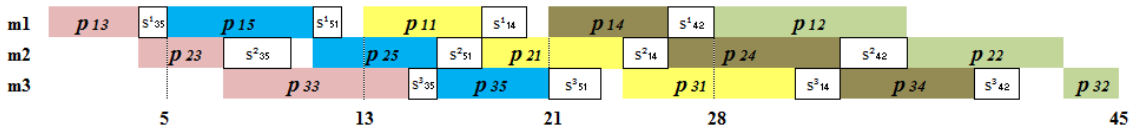

Figura 11 - Sequência final

Em seguida, o método realiza as buscas de inserção e de permutação (vide Figuras 9 e 10). Neste caso particular, as buscas não encontram solução melhor do que a sequência $\{j_3, j_5, j_1, j_4, j_2\}$, e esta fica sendo a sequência final.

CAPÍTULO 8

EXPERIMENTAÇÃO COMPUTACIONAL

A experimentação computacional neste trabalho tem por objetivo avaliar o desempenho dos métodos propostos GAPH e QUARTS em relação aos métodos BAH, BIH e TRIPS. Como já mencionado, o método TRIPS será adaptado para o problema com tempos de *setup* dependentes da sequência, tanto para o critério do *makespan* quanto do *total flowtime*, e os métodos BAH e BIH serão utilizados na sua forma original, i.e., para minimização do *makespan,* e também serão adaptados para o critério de minimização do *total flowtime*.

8.1 Delineamento do Experimento

A experimentação computacional será dividida em duas partes:

- Parte 1: serão avaliados os métodos GAPH, BAH, BIH e TRIPS com o critério de minimização do *makespan*.
- Parte 2: serão avaliados os métodos QUARTS, BAH, BIH e TRIPS com o critério de minimização do *total flowtime*.

As heurísticas serão avaliadas com o banco de dados de Ruiz, Maroto e Alcaraz (2005). Esse banco de dados contém 12 conjuntos de problemas (n x m), com combinações entre $n \in \{20, 50, 100, 200, 500\}$ e $m \in \{5, 10, 20\}$.

Para cada conjunto de problemas, há 4 diferentes distribuições dos tempos de *setup*, com 10 problemas cada: SSD-10, SSD-50, SSD-100 e SSD-125. Os tempos de processamento são uniformemente distribuídos no intervalo de [1,99] e os tempos de *setup* são distribuídos uniformemente nos intervalos [1,9], [1,49], [1,99] e [1,124], para as classes de setup SSD-10, SSD-50, SSD-100 e SSD-125, respectivamente. Por exemplo, na classe SSD-10 o tempo máximo de *setup* corresponde a 10% do tempo de processamento, enquanto na classe SSD-125 ele corresponde a 125% do tempo de processamento.

Como há 12 grupos de problemas (*n* x *m*) com 4 classes de 10 problemas cada, totaliza-se 480 problemas. Entretanto, devido ao conjunto de problemas com 500 tarefas ser muito grande para fins práticos, optou-se por retirar esse problema (composto por 500 tarefas e 20 máquinas). Assim, totaliza-se 440 problemas.

8.2 Processo de Análise

Os resultados obtidos na experimentação computacional serão analisados com relação à Porcentagem de Sucesso, ao Desvio Relativo Médio e ao Tempo Médio de Computação.

A Porcentagem de Sucesso é definida pelo quociente entre o número de problemas para os quais um determinado método obteve a melhor solução (menor *makespan/total flowtime*) e o número total de programas resolvidos. Logo,

quando os métodos obtêm a melhor solução para um mesmo problema, suas Porcentagens de Sucesso são simultaneamente melhoradas.

O Desvio Relativo (DR_h) quantifica o desvio que o método h obtém em relação ao melhor *makespan/flowtime* obtido para o mesmo problema, sendo calculado conforme segue:

$$DR_h(\%) = \frac{D_h - D*}{D*} x100 \tag{10}$$

onde:

D_h é o *makespan/total flowtime* obtido pelo método h;

$D*$ é o melhor *makespan/total flowtime* obtido pelo(s) método(s) para um determinado problema.

O Tempo Médio de Computação de um método é obtido pela soma dos tempos de computação de cada problema dividida pelo número total de problemas resolvidos. Na experimentação computacional, o tempo médio de computação será medido em segundos (s).

CAPÍTULO 9

ANÁLISE DOS RESULTADOS

Objetivando uma melhor visualização e, consequentemente, análise dos resultados obtidos, estes estão apresentados em tabelas e gráficos de linhas e de colunas. Os resultados estão divididos nas duas partes detalhadas na experimentação computacional, sendo que o Tópico 9.1 apresenta os resultados com relação ao critério *makespan* e o Tópico 9.2 apresenta os do *total flowtime*. Dentro de cada tópico, há subtópicos para a porcentagem de sucesso, desvio relativo médio e tempo médio de computação.

Para a construção dos gráficos, os resultados obtidos na experimentação computacional foram agrupados em relação ao número de máquinas. Por exemplo, o resultado para 20 tarefas em cada distribuição do tempo de *setup* é uma média entre os resultados obtidos para 20 tarefas e 5 máquinas, 20 tarefas e 10 máquinas e 20 tarefas e 20 máquinas para a respectiva distribuição.

9.1 Critério de Minimização do *Makespan*

Na primeira parte da experimentação computacional, foram testados os métodos BAH, BIH, TRIPS e GAPH com o critério de minimização do *makespan*. Eles foram codificados em linguagem *Python* e processados conjuntamente em um microcomputador *Pentium IV 3.00GHz, 512MB de RAM*.

9.1.1 Análise da Porcentagem de Sucesso

As Tabelas 14, 15, 16 e 17 e os Gráficos 5, 6, 7 e 8 mostram os resultados da porcentagem de sucesso para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,00	60,00	0,00	100,00
	10	0,00	50,00	0,00	100,00
	20	0,00	60,00	0,00	100,00
	Média	0,00	56,67	0,00	100,00
50	**5**	0,00	70,00	0,00	100,00
	10	0,00	90,00	0,00	100,00
	20	0,00	80,00	0,00	100,00
	Média	0,00	80,00	0,00	100,00
100	**5**	0,00	70,00	0,00	100,00
	10	0,00	40,00	0,00	100,00
	20	0,00	60,00	0,00	100,00
	Média	0,00	56,67	0,00	100,00
200	**10**	0,00	50,00	0,00	100,00
	20	0,00	70,00	0,00	100,00
	Média	0,00	60,00	0,00	100,00

Tabela 14 – Porcentagem de sucesso para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,00	60,00	0,00	100,00
	10	0,00	70,00	0,00	100,00
	20	0,00	70,00	0,00	100,00
	Média	0,00	66,67	0,00	100,00
50	**5**	0,00	50,00	0,00	100,00
	10	0,00	80,00	0,00	100,00
	20	0,00	90,00	0,00	100,00

	Média	0,00	73,33	0,00	100,00
100	**5**	0,00	60,00	0,00	100,00
	10	0,00	30,00	0,00	100,00
	20	0,00	80,00	0,00	100,00
	Média	0,00	56,67	0,00	100,00
200	**10**	0,00	80,00	0,00	100,00
	20	0,00	80,00	0,00	100,00
	Média	0,00	80,00	0,00	100,00

Tabela 15 – Porcentagem de sucesso para a classe SSD-50

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,00	20,00	0,00	100,00
	10	0,00	50,00	0,00	100,00
	20	0,00	60,00	0,00	100,00
	Média	0,00	43,33	0,00	100,00
50	**5**	0,00	50,00	0,00	100,00
	10	0,00	70,00	0,00	100,00
	20	0,00	80,00	0,00	100,00
	Média	0,00	66,67	0,00	100,00
100	**5**	0,00	30,00	0,00	100,00
	10	0,00	60,00	0,00	100,00
	20	0,00	70,00	0,00	100,00
	Média	0,00	53,33	0,00	100,00
200	**10**	0,00	40,00	0,00	100,00
	20	0,00	60,00	0,00	100,00
	Média	0,00	50,00	0,00	100,00

Tabela 16 – Porcentagem de sucesso para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,00	60,00	10,00	90,00
	10	0,00	50,00	0,00	100,00
	20	0,00	50,00	0,00	100,00
	Média	0,00	53,33	3,33	96,67
50	**5**	0,00	30,00	0,00	100,00
	10	0,00	60,00	0,00	100,00

	20	0,00	90,00	0,00	100,00
	Média	0,00	60,00	0,00	100,00
100	**5**	0,00	20,00	0,00	100,00
	10	0,00	70,00	0,00	100,00
	20	0,00	90,00	0,00	100,00
	Média	0,00	60,00	0,00	100,00
200	**10**	0,00	30,00	0,00	100,00
	20	0,00	60,00	0,00	100,00
	Média	0,00	45,00	0,00	100,00

Tabela 17 – Porcentagem de sucesso para a classe SSD-125

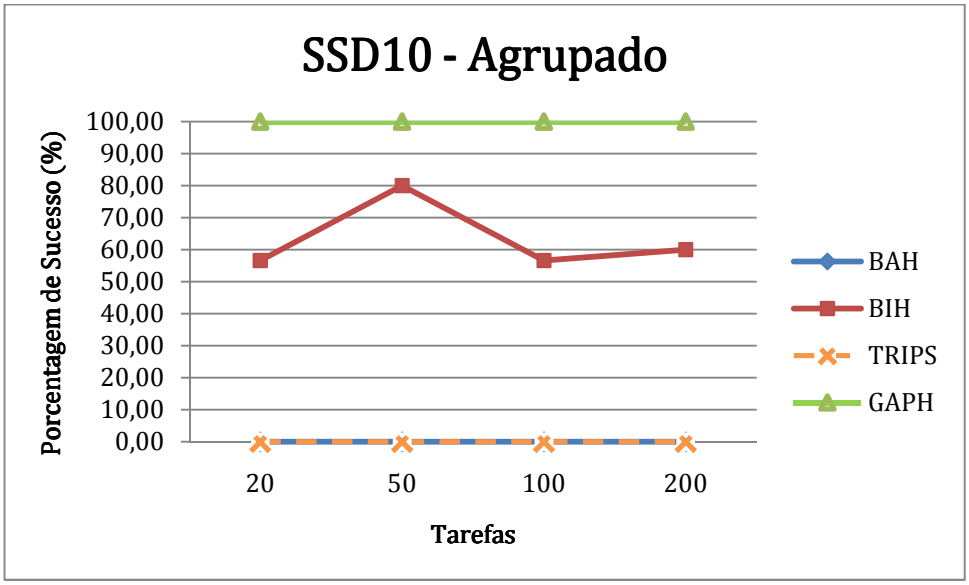

Gráfico 5 – Porcentagem de sucesso por máquinas agrupadas para a classe SSD-10

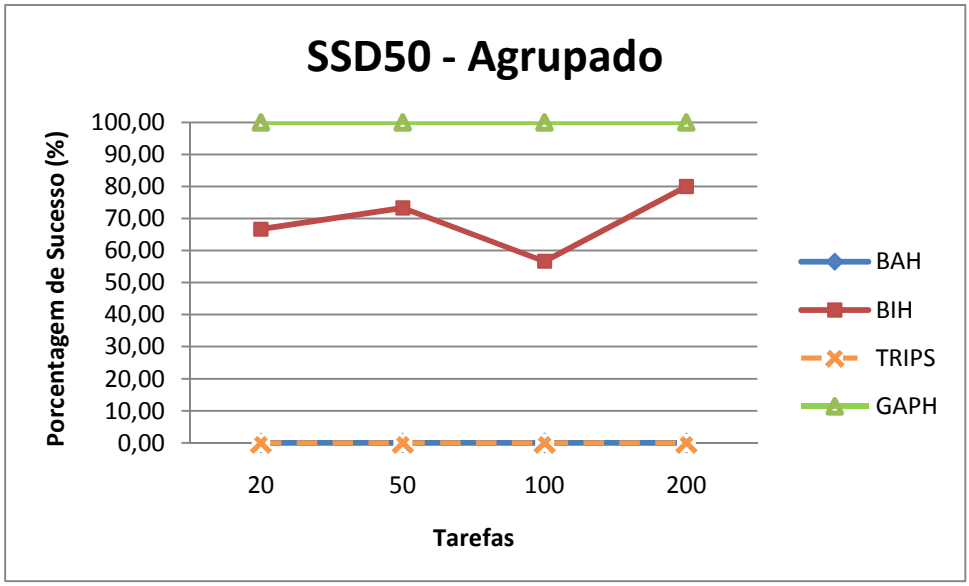

Gráfico 6 - Porcentagem de sucesso por máquinas agrupadas para a classe SSD-50

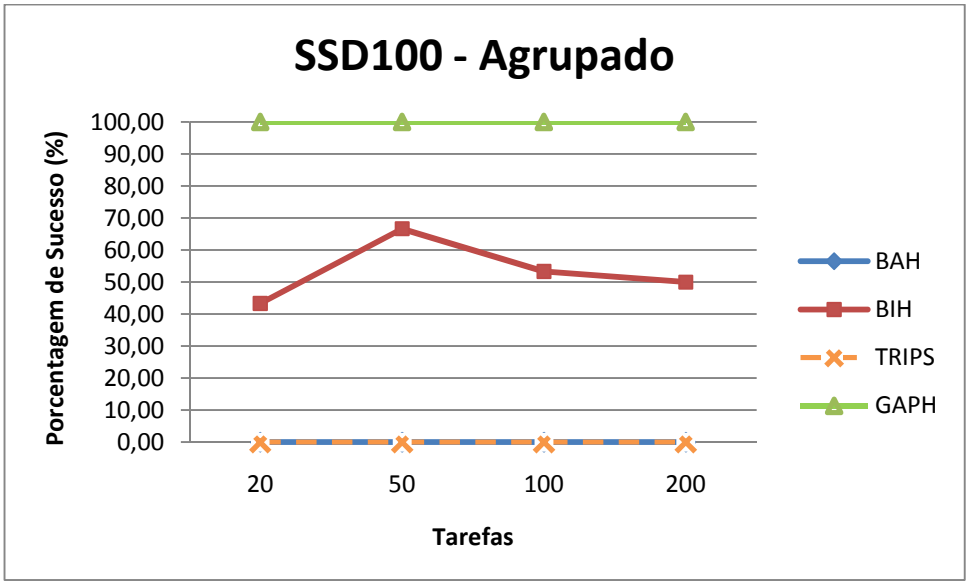

Gráfico 7 - Porcentagem de sucesso por máquinas agrupadas para a classe SSD-100

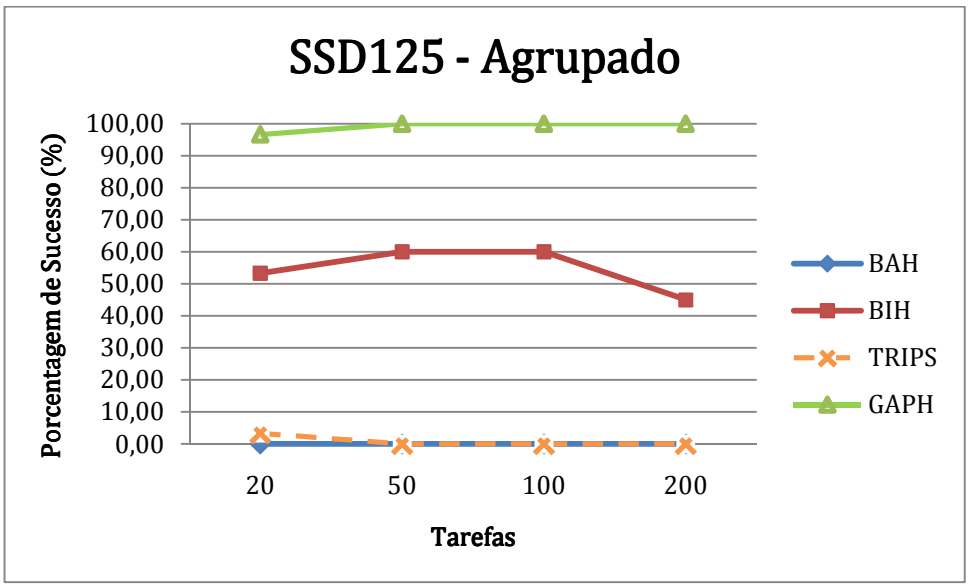

Gráfico 8 - Porcentagem de sucesso por máquinas agrupadas para a classe SSD-125

Uma primeira análise é realizada com relação aos desempenhos dos métodos BAH, BIH, TRIPS e GAPH com o critério de minimização do *makespan*. Observando-se as Tabelas 14, 15, 16 e 17 e os Gráficos 5, 6, 7 e 8, percebe-se que o método GAPH sempre obtém a melhor solução, ou seja, sua porcentagem de sucesso é 100% em todos os casos, salvo um único caso com 20 tarefas e 5 máquinas, em que o método TRIPS obteve a melhor solução. O método BIH possui uma porcentagem de sucesso média de 60%. Observa-se que não existe uma relação entre o desempenho do BIH em termos da porcentagem de sucesso e a quantidade de tarefas. O método BAH obteve uma porcentagem de sucesso nula, ou seja, em nenhuma das vezes conseguiu encontrar o melhor resultado, enquanto o método TRIPS conseguiu chegar ao melhor resultado em apenas um dos 440 problemas testados, obtendo uma porcentagem de sucesso média de 0,23%. Assim, conclui-se que os métodos BAH e TRIPS não são competitivos para o critério de minimização do *makespan*. Ademais, conclui-se que o método GAPH é superior aos

outros, já que obteve a melhor solução em 439 dos 440 problemas testados, enquanto o método BIH obteve em 264 deles.

Uma característica interessante da experimentação computacional é que os métodos parecem não ser afetados pela distribuição dos tempos de *setup*, e nem pela quantidade de tarefas e de máquinas. Ou seja, não existe uma tendência em nenhum dos métodos, considerando-se esses três aspectos. Através dos Gráficos 5, 6, 7 e 8, observa-se que os métodos GAPH, BAH e TRIPS possuem uma tendência linear praticamente em todos eles, e o método BIH ora aumenta, ora diminui a porcentagem de sucesso, independente da quantidade de tarefas e da distribuição dos tempos de processamento.

9.1.2 Análise do Desvio Relativo Médio

As Tabelas 18, 19, 20 e 21 e os Gráficos 9, 10, 11 e 12 mostram os resultados do desvio relativo médio para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	15,54	1,83	9,27	0,00
	10	15,87	2,48	8,23	0,00
	20	13,42	0,42	8,82	0,00
	Média	14,94	1,58	8,77	0,00
50	**5**	11,74	0,41	5,77	0,00
	10	13,42	0,08	6,67	0,00
	20	13,92	0,10	8,41	0,00
	Média	13,03	0,20	6,95	0,00

100	**5**	10,11	0,73	6,08	0,00
	10	11,16	0,85	7,07	0,00
	20	11,42	0,26	7,54	0,00
	Média	10,89	0,61	6,90	0,00
200	**10**	8,11	0,33	6,08	0,00
	20	8,68	0,11	5,52	0,00
	Média	8,39	0,22	5,80	0,00

Tabela 18 – Desvio relativo médio para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	13,13	1,27	7,23	0,00
	10	14,21	1,59	8,09	0,00
	20	13,21	0,88	8,37	0,00
	Média	13,51	1,25	7,90	0,00
50	**5**	9,80	0,98	5,44	0,00
	10	11,31	0,08	5,57	0,00
	20	12,22	0,03	7,60	0,00
	Média	11,11	0,37	6,20	0,00
100	**5**	8,27	0,36	5,32	0,00
	10	8,68	0,43	6,68	0,00
	20	9,81	0,14	6,74	0,00
	Média	8,92	0,31	6,25	0,00
200	**10**	6,68	0,12	4,58	0,00
	20	7,43	0,02	4,40	0,00
	Média	7,05	0,07	4,49	0,00

Tabela 19 – Desvio relativo médio para a classe SSD-50

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	9,95	1,08	5,71	0,00
	10	12,20	1,09	6,82	0,00

	20	12,15	0,35	7,57	0,00
	Média	11,44	0,84	6,70	0,00
50	**5**	9,51	0,80	4,38	0,00
	10	8,98	0,13	4,01	0,00
	20	9,95	0,04	5,53	0,00
	Média	9,48	0,32	4,64	0,00
100	**5**	7,32	0,71	4,40	0,00
	10	7,38	0,17	4,88	0,00
	20	8,05	0,37	5,37	0,00
	Média	7,58	0,42	4,89	0,00
200	**10**	6,37	0,59	4,21	0,00
	20	6,55	0,09	3,45	0,00
	Média	6,46	0,34	3,83	0,00

Tabela 20 – Desvio relativo médio para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	10,54	0,85	5,08	0,14
	10	11,46	1,03	5,62	0,00
	20	11,85	0,93	6,63	0,00
	Média	11,28	0,94	5,78	0,05
50	**5**	9,21	1,32	5,70	0,00
	10	9,97	0,93	4,99	0,00
	20	9,07	0,01	5,44	0,00
	Média	9,41	0,75	5,38	0,00
100	**5**	7,14	1,02	4,44	0,00
	10	7,05	0,24	3,81	0,00
	20	6,82	0,08	4,47	0,00
	Média	7,00	0,44	4,24	0,00
200	**10**	5,59	0,56	3,44	0,00
	20	5,70	0,09	3,15	0,00
	Média	5,64	0,32	3,29	0,00

Tabela 21 – Desvio relativo médio para a classe SSD-125

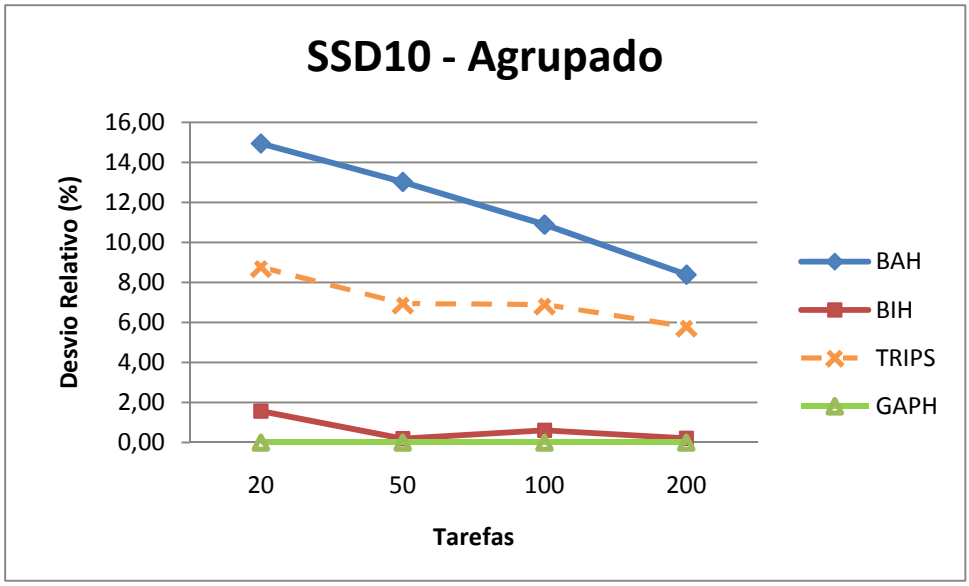

Gráfico 9 – Desvio relativo médio por máquinas agrupadas para a classe SSD-10

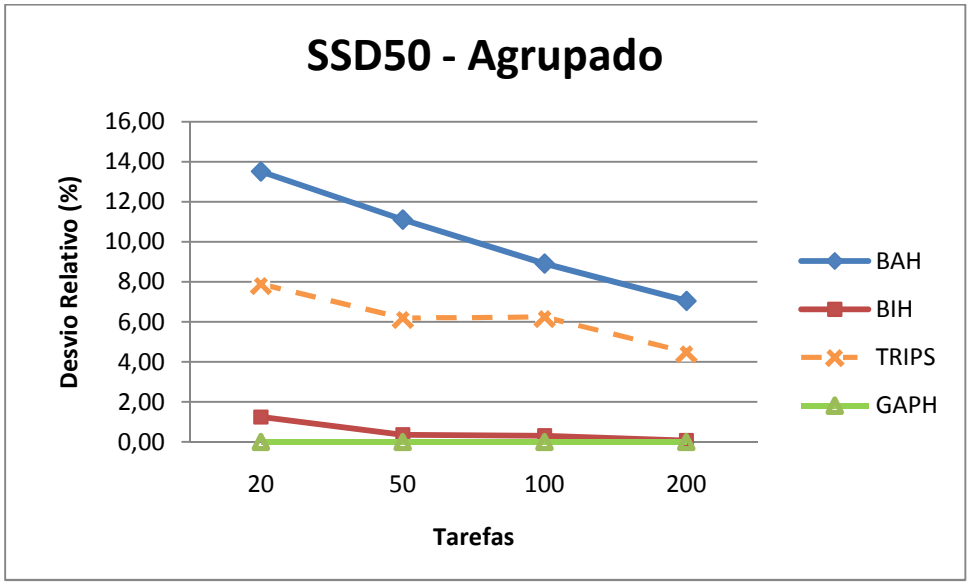

Gráfico 10 - Desvio relativo médio por máquinas agrupadas para a classe SSD-50

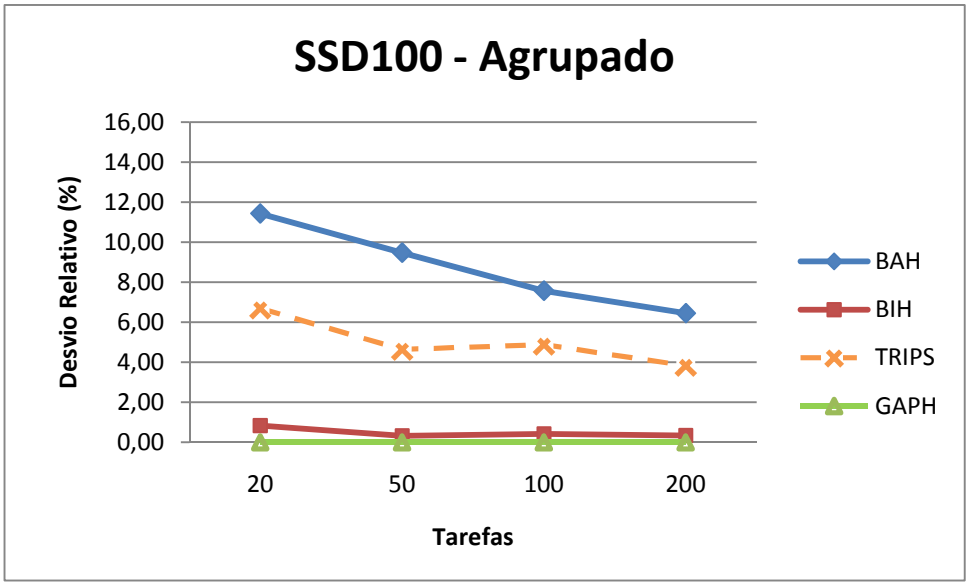

Gráfico 11 - Desvio relativo médio por máquinas agrupadas para a classe SSD-100

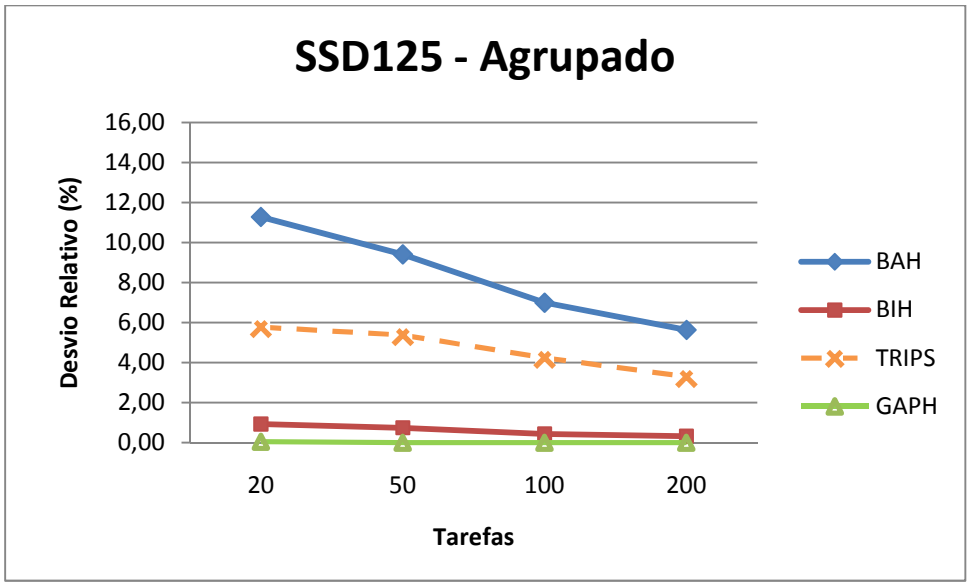

Gráfico 12 - Desvio relativo médio por máquinas agrupadas para a classe SSD-125

Através da observação dos gráficos e tabelas acima, percebe-se que os desvios relativos médios dos métodos possuem faixas de valores distintas e espaçadas. O método BAH, que já havia sido considerado não competitivo, possui os maiores desvios. O desvio relativo médio deste método chega a quase 16% em alguns problemas, com uma média geral de aproximadamente 10%. O segundo

método com piores resultados é o TRIPS, com desvio relativo médio de 6%. O maior desvio relativo médio calculado para este método chega a quase 10%. Ambos os métodos possuem uma tendência a diminuir o percentual do desvio relativo médio com o aumento do número de tarefas, o que indica que eles obtêm melhores resultados em problemas de grande porte. Além disso, observa-se uma relação entre o desvio relativo e a distribuição dos tempos de *setup* para ambos os métodos. Parece haver uma tendência entre o aumento da distribuição dos tempos de *setup* e a diminuição do desvio relativo, indicando que os métodos são melhores em ambientes com altos tempos de *setup*, em comparação aos tempos de processamento. O BAH possui um desvio relativo médio de 12,13% no SSD-10, diminuindo para 10,43% no SSD-50, 8,95% no SSD-100 e 8,58% no SSD-125. O TRIPS segue a mesma linha, diminuindo de 7,22% no SSD-10 para 6,37% no SSD-50, para 5,12% no SSD-100 e para 4,80% no SSD-125.

Os quatro métodos avaliados não apresentam tendência crescente ou decrescente em relação à quantidade de máquinas.

Como os desvios relativos dos métodos BIH e GAPH são muito inferiores aos dos métodos BAH e TRIPS, fez-se necessário apresentar gráficos de desvio para os dois somente, por uma questão de escala. Os Gráficos 13, 14, 15 e 16 a seguir apresentam tais desvios relativos.

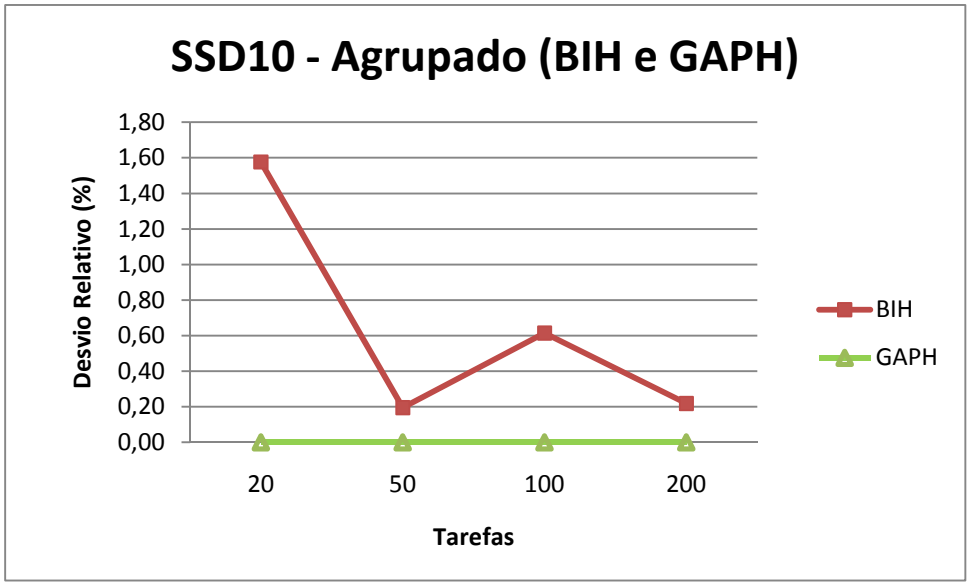

Gráfico 13 – Desvio relativo médio por máquinas agrupadas do BIH e do GAPH para a classe SSD-10

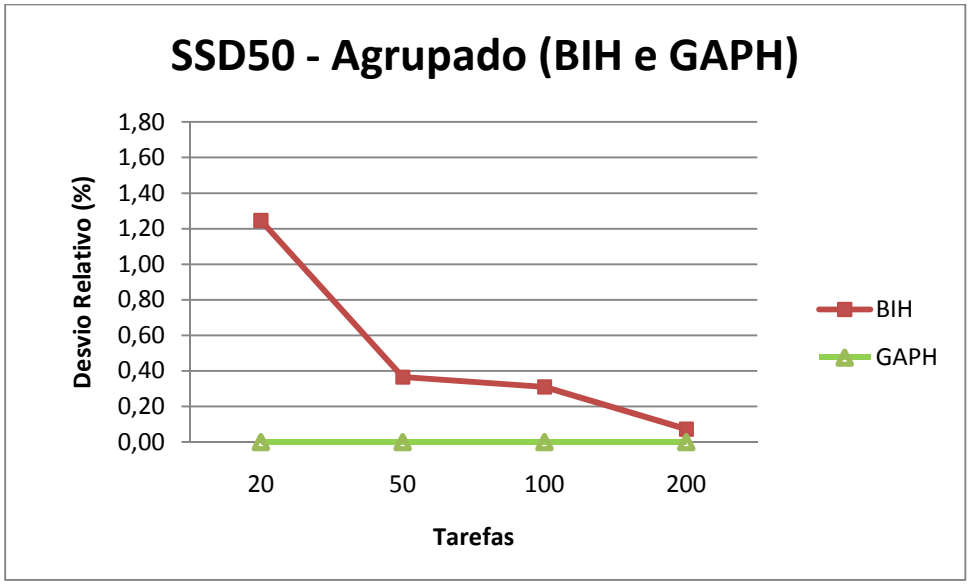

Gráfico 14 - Desvio relativo médio por máquinas agrupadas do BIH e do GAPH para a classe SSD-50

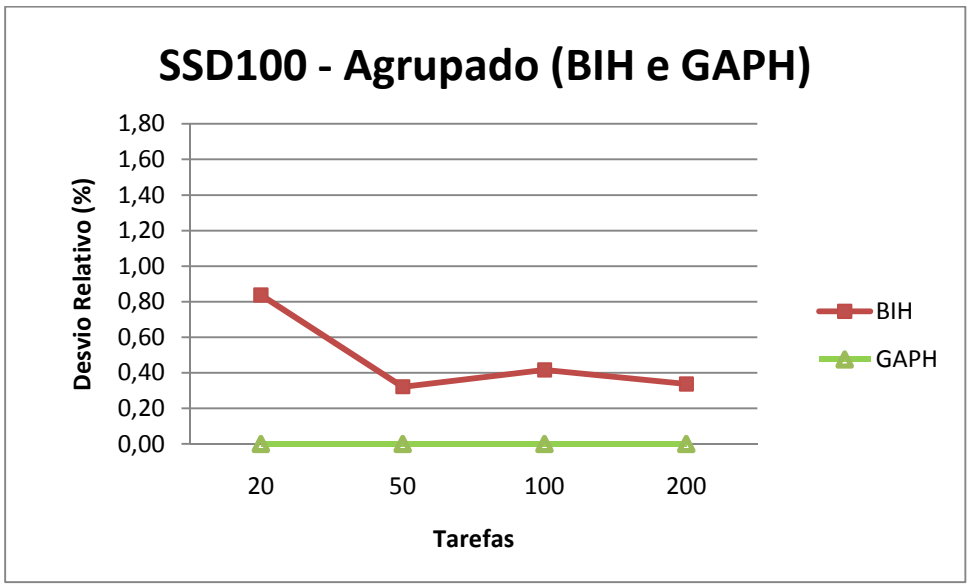

Gráfico 15 - Desvio relativo médio por máquinas agrupadas do BIH e do GAPH para a classe SSD-100

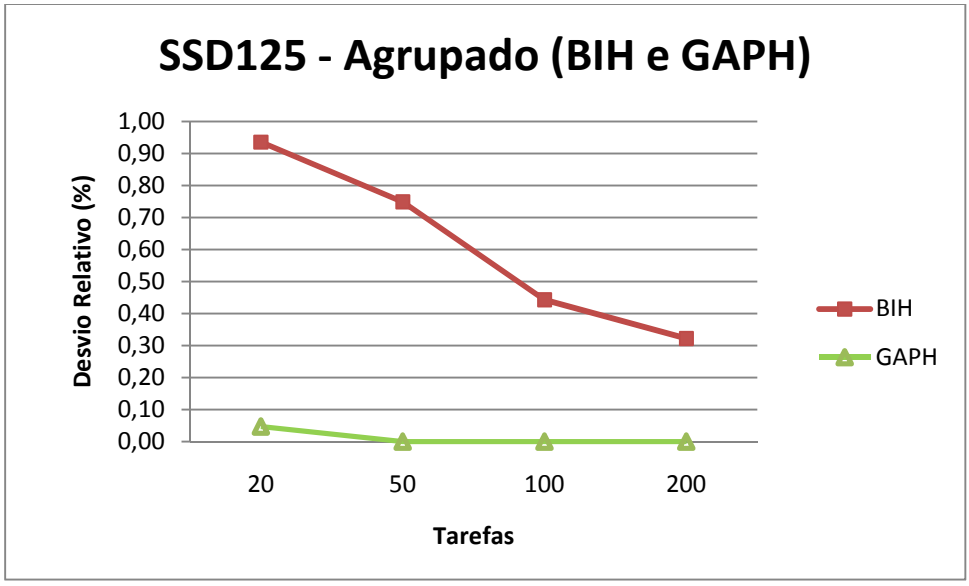

Gráfico 16 - Desvio relativo médio por máquinas agrupadas do BIH e do GAPH para a classe SSD-125

Para o método BIH, o maior desvio relativo médio é de aproximadamente 2,5%, e a média entre os desvios relativos de todos os 440 problemas é de 0,56%. Observa-se uma tendência de diminuição dos desvios relativos médios com o aumento do número de tarefas. Para o SSD-50, por

exemplo, o desvio relativo médio deste método para 20 tarefas é de 1,25%, caindo para 0,37% para 50 tarefas, 0,31% para 100 tarefas e para apenas 0,07% para 200 tarefas. Essa tendência mostra que, apesar da porcentagem de sucesso do método não aumentar com o aumento do número de tarefas, ele encontra soluções melhores (mais próximas do melhor valor encontrado) com problemas de grande porte. Ademais, o método parece ser influenciado pela distribuição dos tempos de *setup*, mas a natureza desta relação não pôde ser identificada, já que o método diminuiu os desvios relativos médios com o aumento dos tempos de *setup* de SSD-10 para SSD-50 e de SSD-50 para SSD-100, mas aumentou novamente seus desvios relativos com o aumento dos tempos de *setup* de SSD-100 para SSD-125. Logo, não se pode afirmar se o método BIH funciona melhor com tempos de *setup* pequenos ou grandes em relação aos tempos de processamento.

Como a análise de porcentagem de sucesso mostrou que o método GAPH obtém em praticamente todos os problemas o melhor resultado, era esperado que seu desvio relativo médio fosse nulo. Através da observação dos Gráficos 13, 14, 15 e 16, nota-se que o desvio relativo deste método coincide com o eixo indicativo do desvio relativo zero, salvo em um ponto. Assim, a análise do desvio relativo confirmou a superioridade deste método.

9.1.3 Análise do Tempo Médio de Computação

As Tabelas 22, 23, 24 e 25 e os Gráficos 17, 18, 19 e 20 mostram os resultados do tempo médio de computação para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,03	0,11	0,34	0,27
	10	0,06	0,15	0,37	0,31
	20	0,18	0,26	0,43	0,43
	Média	0,09	0,17	0,38	0,34
50	**5**	0,17	2,87	14,30	7,92
	10	0,40	3,09	14,45	8,30
	20	1,16	3,84	14,85	9,10
	Média	0,58	3,27	14,53	8,44
100	**5**	0,83	40,70	226,25	117,27
	10	1,67	39,97	226,09	118,98
	20	5,60	43,73	225,70	122,87
	Média	2,70	41,47	226,01	119,71
200	**10**	6,88	611,50	3799,28	1837,43
	20	21,63	616,54	3681,30	1834,39
	Média	14,25	614,02	3740,29	1835,91

Tabela 22 – Tempo médio de computação para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,03	0,11	0,34	0,27
	10	0,06	0,15	0,36	0,31
	20	0,17	0,26	0,42	0,43
	Média	0,09	0,17	0,37	0,34
50	**5**	0,16	2,89	14,11	7,96
	10	0,40	3,14	14,22	8,35
	20	1,17	3,88	14,66	9,15
	Média	0,57	3,30	14,33	8,49
100	**5**	0,67	40,75	225,78	117,62
	10	1,69	40,17	226,26	118,94
	20	5,07	43,82	226,89	123,65
	Média	2,48	41,58	226,31	120,07
200	**10**	6,81	612,89	3759,58	1839,95
	20	20,81	617,86	3653,72	1843,07
	Média	13,81	615,37	3706,65	1841,51

Tabela 23 – Tempo médio de computação para a classe SSD-50

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,02	0,11	0,34	0,27
	10	0,07	0,14	0,36	0,31
	20	0,18	0,26	0,42	0,43
	Média	0,09	0,17	0,37	0,34
50	**5**	0,17	2,88	14,11	8,01
	10	0,39	3,12	14,27	8,37
	20	1,19	3,85	14,71	9,13
	Média	0,58	3,28	14,36	8,50
100	**5**	0,68	41,16	223,30	118,19
	10	1,75	40,67	226,42	119,35
	20	5,18	43,87	228,77	123,35
	Média	2,54	41,90	226,16	120,30

200	**10**	6,87	613,21	3780,10	1853,66
	20	21,19	619,73	3657,28	1846,33
	Média	14,03	616,47	3718,69	1850,00

Tabela 24 – Tempo médio de computação para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**GAPH**
20	**5**	0,03	0,11	0,33	0,27
	10	0,06	0,15	0,36	0,31
	20	0,18	0,26	0,42	0,44
	Média	0,09	0,17	0,37	0,34
50	**5**	0,17	2,88	14,11	8,02
	10	0,40	3,11	14,30	8,34
	20	1,16	3,85	14,74	9,13
	Média	0,58	3,28	14,38	8,50
100	**5**	0,67	40,93	223,37	118,93
	10	2,03	40,36	227,38	119,32
	20	5,19	44,09	229,54	123,40
	Média	2,63	41,80	226,76	120,55
200	**10**	7,03	612,99	3793,94	1836,08
	20	21,13	617,28	3649,61	1844,22
	Média	14,08	615,14	3721,78	1840,15

Tabela 25 – Tempo médio de computação para a classe SSD-125

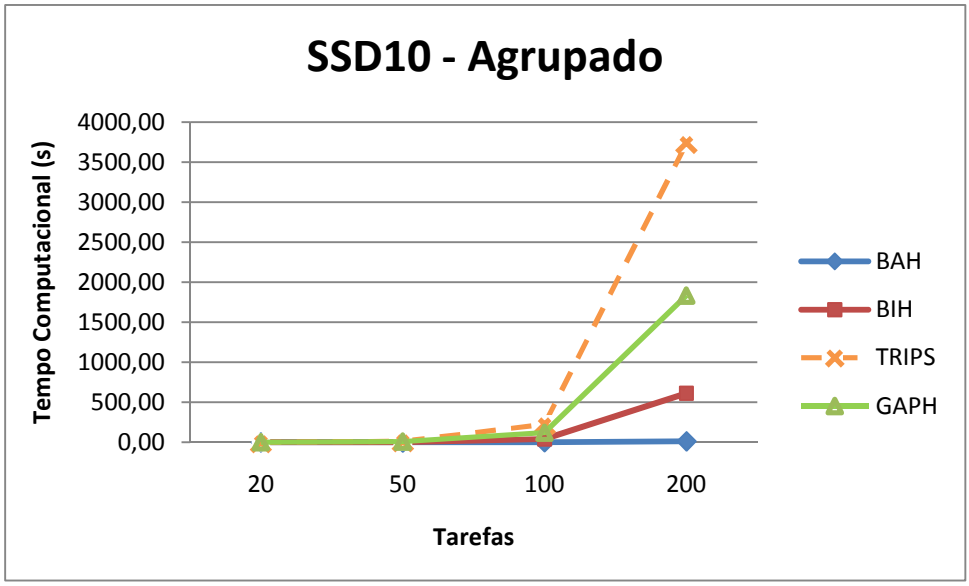

Gráfico 17 –Tempo médio de computação por máquinas agrupadas para a classe SSD-10

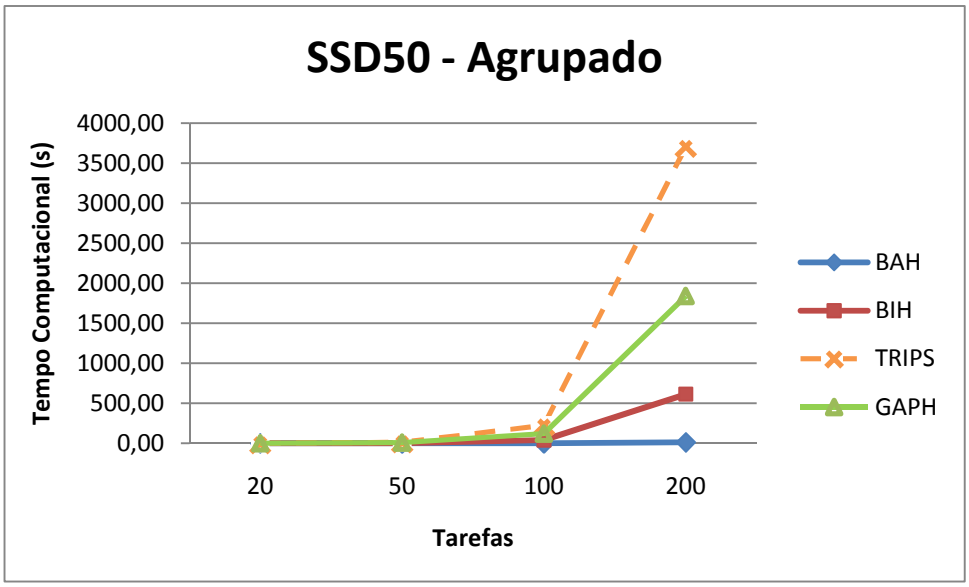

Gráfico 18 - Tempo médio de computação por máquinas agrupadas para a classe SSD-50

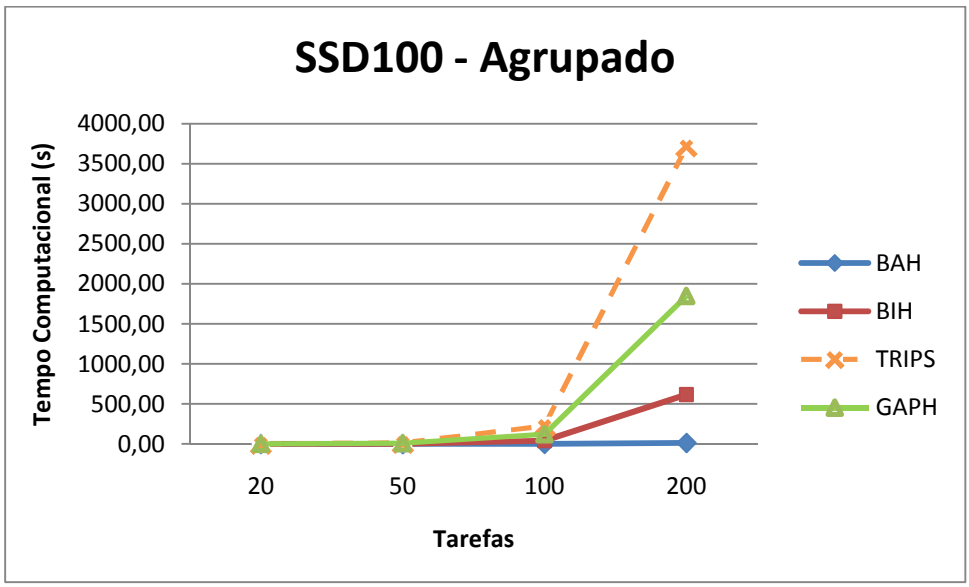

Gráfico 19 - Tempo médio de computação por máquinas agrupadas para a classe SSD-100

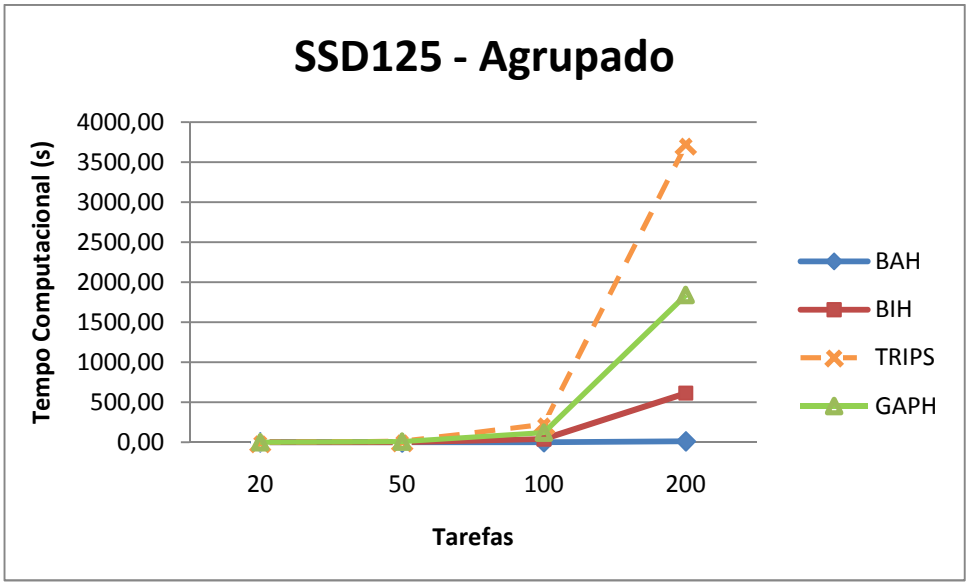

Gráfico 20 - Tempo médio de computação por máquinas agrupadas para a classe SSD-125

Em termos do tempo médio de computação, percebe-se que o BAH é o método mais rápido. No pior caso (200 tarefas e 20 máquinas), seu tempo é de aproximadamente 21s, enquanto o tempo do BIH, o segundo método mais rápido, é de aproximadamente 618s.

Comparando-se o BIH com o GAPH, observa-se que o BIH é mais rápido do que o GAPH. Para o problema com 20 tarefas, o método BIH é quase duas vezes mais rápido do que o GAPH. Já para maior problema, com 200 tarefas, o BIH é quase três vezes mais rápido do que o GAPH. Comparando o GAPH com o TRIPS, observa-se que o GAPH é aproximadamente duas vezes mais rápido que o TRIPS. Logo, apesar do tempo computacional do método proposto ser mais alto do que o do método BIH, pode-se concluir que ele é um método eficiente, por ser mais rápido do que o TRIPS.

Através das tabelas, percebe-se que o método BAH é o único que aparenta ser afetado pela variação do número de máquinas. De um problema com 10 máquinas para um de 20 máquinas, ele tem seu tempo computacional aumentado em aproximadamente três vezes. Observando os gráficos, nota-se que o tempo computacional de todos os métodos é influenciado pelo número de tarefas do problema. Existe uma relação proporcional entre o tempo e o número de tarefas, ou seja, quanto maior o número de tarefas, maior o tempo computacional de cada método. Não foi observada nenhuma relação entre o tempo computacional e as distribuições dos tempos de *setup*.

De forma geral, dentre todos os métodos avaliados nesta experimentação, verifica-se que o método proposto GAPH é o que apresenta melhores resultados, tanto em termos de porcentagem de sucesso quanto em termos de desvio relativo médio. Fazendo uma média entre os 440 problemas testados, sua porcentagem de sucesso é de 99,77% e seu desvio relativo médio é de 0%, enquanto que os métodos BIH, TRIPS e BAH possuem porcentagem de sucesso de 60,23%, 0,23% e 0%, e desvio relativo médio de 0,59%, 5,88% e 10,02%, respectivamente. Com respeito ao tempo médio de computação, o método

GAPH é mais lento que o BIH e mais rápido do que o TRIPS. Assim, pode-se concluir que a heurística GAPH é a mais indicada para o critério de minimização do *makespan*.

9.2 Minimização do *Total Flowtime*

Na segunda parte da experimentação computacional, foram testados os métodos BAH, BIH, TRIPS e QUARTS com o critério de minimização do *total flowtime*. Eles foram codificados em linguagem *Python* e processados conjuntamente em um microcomputador *Intel Core 2 Duo 2,20GHz/2,20GHz 4,00GB de RAM*.

9.2.1 Análise da Porcentagem de Sucesso

As Tabelas 26, 27, 28 e 29 e os Gráficos 21, 22, 23 e 24 mostram os resultados da porcentagem de sucesso para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,00	0,00	20,00	90,00
	10	0,00	0,00	20,00	100,00
	20	0,00	0,00	30,00	90,00
	Média	0,00	0,00	23,33	93,33
50	**5**	0,00	0,00	0,00	100,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	0,00	100,00
100	**5**	0,00	0,00	0,00	100,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	10,00	90,00
	Média	0,00	0,00	3,33	96,67
200	**10**	0,00	0,00	0,00	100,00
	20	0,00	0,00	10,00	90,00
	Média	0,00	0,00	5,00	95,00

Tabela 26 – Porcentagem de sucesso para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,00	0,00	30,00	90,00
	10	0,00	0,00	10,00	100,00
	20	0,00	10,00	20,00	80,00
	Média	0,00	3,33	20,00	90,00
50	**5**	0,00	0,00	0,00	100,00
	10	0,00	0,00	20,00	80,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	6,67	93,33
100	**5**	0,00	0,00	0,00	100,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	10,00	90,00
	Média	0,00	0,00	3,33	96,67

	10	0,00	0,00	0,00	100,00
200	**20**	0,00	0,00	0,00	100,00
	Média	0,00	0,00	0,00	100,00

Tabela 27 – Porcentagem de sucesso para a classe SSD-50

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,00	0,00	30,00	100,00
	10	0,00	0,00	30,00	90,00
	20	0,00	0,00	40,00	90,00
	Média	0,00	0,00	33,33	93,33
50	**5**	0,00	0,00	10,00	100,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	3,33	100,00
100	**5**	0,00	0,00	0,00	100,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	0,00	100,00
200	**10**	0,00	0,00	20,00	80,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	10,00	90,00

Tabela 28 - Porcentagem de sucesso para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,00	0,00	70,00	100,00
	10	0,00	0,00	30,00	90,00
	20	0,00	0,00	10,00	100,00
	Média	0,00	0,00	36,67	96,67

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
50	**5**	0,00	0,00	60,00	70,00
	10	0,00	0,00	0,00	100,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	20,00	90,00
100	**5**	0,00	0,00	30,00	100,00
	10	0,00	0,00	10,00	90,00
	20	0,00	0,00	0,00	100,00
	Média	0,00	0,00	13,33	96,67
200	**10**	0,00	0,00	0,00	100,00
	20	0,00	0,00	20,00	80,00
	Média	0,00	0,00	10,00	90,00

Tabela 29 – Porcentagem de sucesso para a classe SSD-125

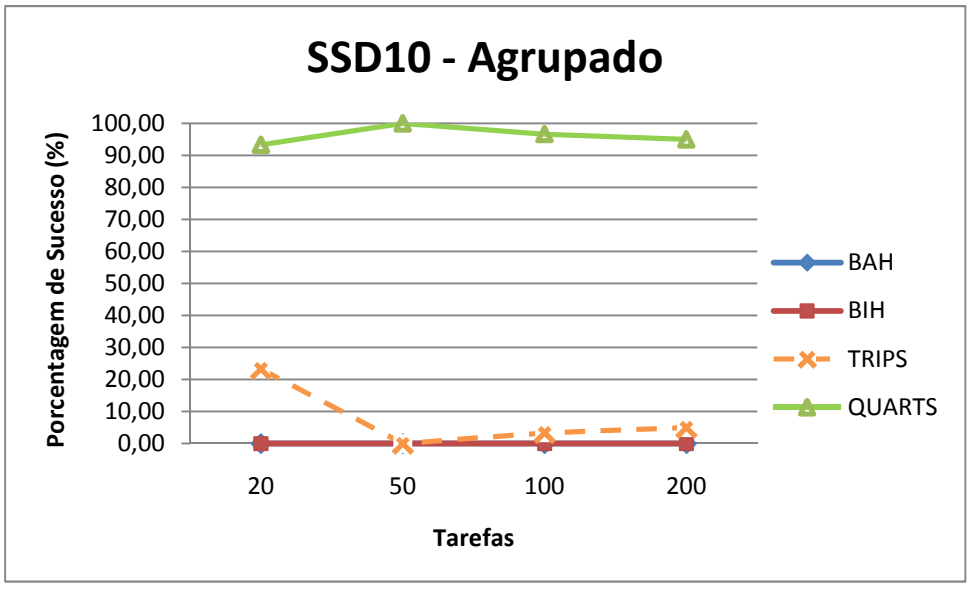

Gráfico 21 – Porcentagem de sucesso por máquinas agrupadas para a classe SSD-10

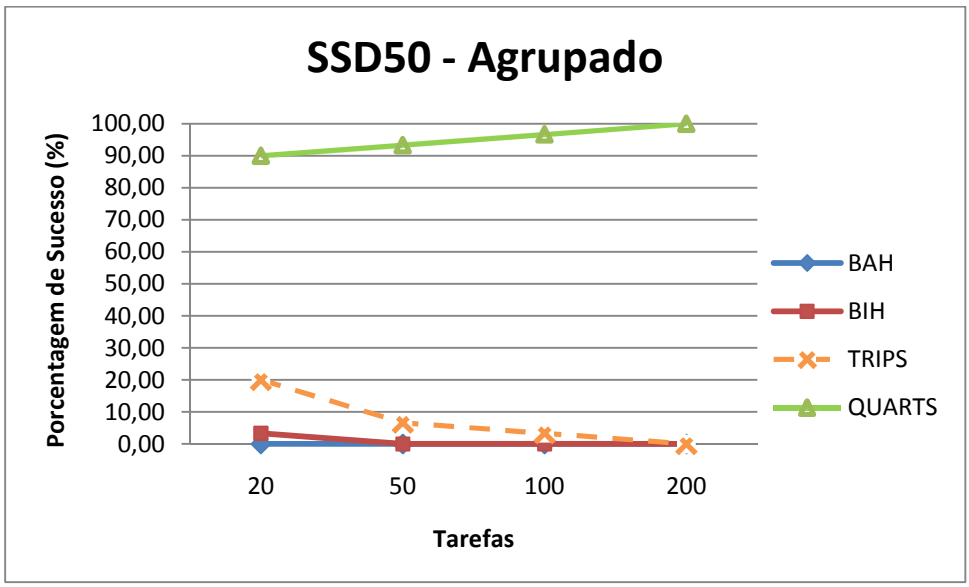

Gráfico 22 – Porcentagem de sucesso por máquinas agrupadas para a classe SSD-50

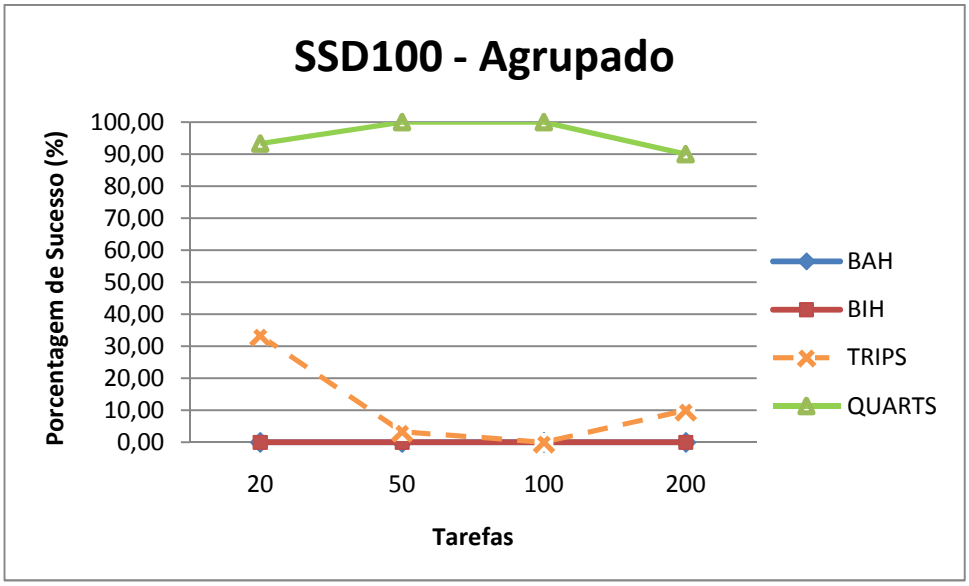

Gráfico 23 - Porcentagem de sucesso por máquinas agrupadas para a classe SSD-100

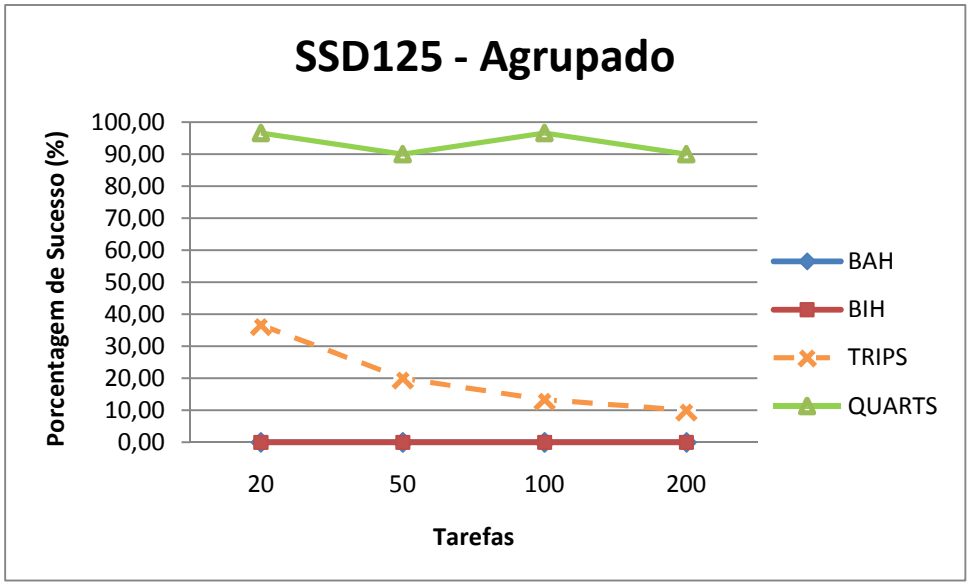

Gráfico 24 - Porcentagem de sucesso por máquinas agrupadas para a classe SSD-125

Uma primeira análise é realizada com relação aos desempenhos dos métodos BAH, BIH, TRIPS e QUARTS com o critério de minimização do *total flowtime*. Pode-se verificar, através das Tabelas 26, 27, 28 e 29 e dos Gráficos 21, 22, 23 e 24, que a porcentagem de sucesso do método QUARTS é superior a todos os outros métodos, em todos os problemas testados. O método obteve a melhor solução em 95,23% dos 440 problemas testados, ou seja, em 419 problemas. O método TRIPS obteve a melhor solução em 54 problemas (12,27%), e os métodos BAH e BIH obtiveram uma porcentagem de sucesso média de 0% e 0,23%, respectivamente, ou seja, o BAH não encontrou a melhor solução nenhuma vez, e o BIH apenas uma. Logo, pode-se concluir que os métodos BAH e BIH não são competitivos para o critério do *total flowtime*. Como já citado, esses métodos foram desenvolvidos por Bianco, Dell´Olmo e Giordani (1999) a fim de minimizarem o *makespan*, e foram adaptados neste trabalho para a minimização do *total flowtime*, já que são os únicos métodos heurísticos construtivos para NWFS com SDST.

Os gráficos acima mostram que há uma relação entre a porcentagem de sucesso do método TRIPS e o número de tarefas. Para o problema com 20 tarefas, a porcentagem de sucesso média do método é de 28,33%, decaindo para 7,50% para 50 tarefas, 5,00% para 100 tarefas e aumentando para 6,25% para 200 tarefas. Considerando as quatro classes analisadas (SSD-10, SSD-50, SSD-100 e SSD-125), essa relação possui como semelhança a diminuição brusca na porcentagem de sucesso (aproximadamente 20%) quando o problema vai de 20 para 50 tarefas. De 50 para 100 e de 100 para 200 tarefas, em geral, a porcentagem de sucesso não possui alterações significantes (aproxidamente 1,5%). Nas classes SSD-50 e SSD-125, a relação entre a porcentagem de sucesso e o número de tarefas é sempre inversamente proporcional. Já nas classes SSD-10 e SSD-100, essa relação se inicia inversamente proporcional, mas se altera em 50 tarefas e 100 tarefas, respectivamente. Assim, pode-se concluir que o método TRIPS obtém melhores resultados para problemas de pequeno porte (i.e., até 20 tarefas), obtendo porcentagem de sucesso próxima de 6% para problemas de médio e grande porte.

Considerando os métodos BAH, BIH e GAPH, não foi observada relação entre suas porcentagens de sucesso e o número de tarefas de cada problema. A porcentagem de sucesso dos métodos BIH e BAH possuem uma tendência linear em todas as classes, ou seja, os métodos obtêm a mesma porcentagem de sucesso, independente do número de tarefas, e não há relação visível entre a porcentagem de sucesso e o número de tarefas do método GAPH. As porcentagens de sucesso de todos os métodos parecem não ser afetadas pela distribuição dos tempos de processamento, e nem pela quantidade de máquinas.

9.2.2. Análise do Desvio Relativo Médio

As Tabelas 30, 31, 32 e 33 e os Gráficos 25, 26, 27 e 28 mostram os resultados do desvio relativo médio para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	12,26	7,55	0,71	0,09
	10	11,45	6,22	0,53	0,00
	20	10,18	4,87	0,44	0,00
	Média	11,29	6,21	0,56	0,03
50	**5**	17,32	7,72	0,34	0,00
	10	20,24	7,90	0,39	0,00
	20	18,51	6,95	0,68	0,00
	Média	18,69	7,52	0,47	0,00
100	**5**	22,93	8,11	0,27	0,00
	10	24,09	7,20	0,52	0,00
	20	25,07	6,06	0,36	0,21
	Média	24,03	7,12	0,39	0,07
200	**10**	28,80	6,27	0,21	0,00
	20	31,86	6,04	0,13	0,04
	Média	30,33	6,16	0,17	0,02

Tabela 30 – Desvio relativo médio para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	11,49	6,69	0,41	0,17
	10	9,04	5,01	0,62	0,00
	20	8,44	5,23	0,47	0,14
	Média	9,65	5,64	0,50	0,10
50	**5**	18,44	7,02	0,40	0,00
	10	17,64	6,90	0,24	0,11
	20	17,08	6,07	0,64	0,00
	Média	17,72	6,66	0,43	0,04
100	**5**	24,94	6,85	0,17	0,00
	10	21,25	6,14	0,46	0,00
	20	20,88	5,07	0,26	0,04
	Média	22,36	6,02	0,30	0,01
200	**10**	26,12	5,47	0,34	0,00
	20	27,83	5,83	0,09	0,00
	Média	26,97	5,65	0,22	0,00

Tabela 31 - Desvio relativo médio para a classe SSD-50

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	15,47	5,21	0,38	0,00
	10	10,57	5,79	0,55	0,02
	20	8,57	4,66	0,34	0,00
	Média	11,54	5,22	0,43	0,01
50	**5**	25,77	7,76	0,29	0,00
	10	19,1	6,58	0,24	0,00
	20	16,28	4,78	0,30	0,00
	Média	20,38	6,37	0,28	0,00

100	**5**	30,5	6,37	0,13	0,00
	10	23,51	5,69	0,16	0,00
	20	19,67	4,71	0,22	0,00
	Média	24,56	5,59	0,17	0,00
200	**10**	27,88	5,17	0,07	0,04
	20	24,30	5,27	0,09	0,00
	Média	26,09	5,22	0,08	0,02

Tabela 32 - Desvio relativo médio para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	17,89	7,11	0,29	0,00
	10	11,05	5,80	0,34	0,20
	20	8,96	4,54	0,32	0,00
	Média	12,63	5,82	0,32	0,07
50	**5**	27,32	8,15	0,24	0,46
	10	19,69	6,80	0,30	0,00
	20	15,37	4,83	0,24	0,00
	Média	20,79	6,59	0,26	0,15
100	**5**	33,78	6,68	0,05	0,00
	10	23,67	6,08	0,12	0,06
	20	18,91	4,27	0,16	0,00
	Média	25,45	5,68	0,11	0,02
200	**10**	28,71	5,69	0,05	0,00
	20	24,30	5,27	0,09	0,00
	Média	26,50	5,48	0,07	0,00

Tabela 33 - Desvio relativo médio para a classe SSD-125

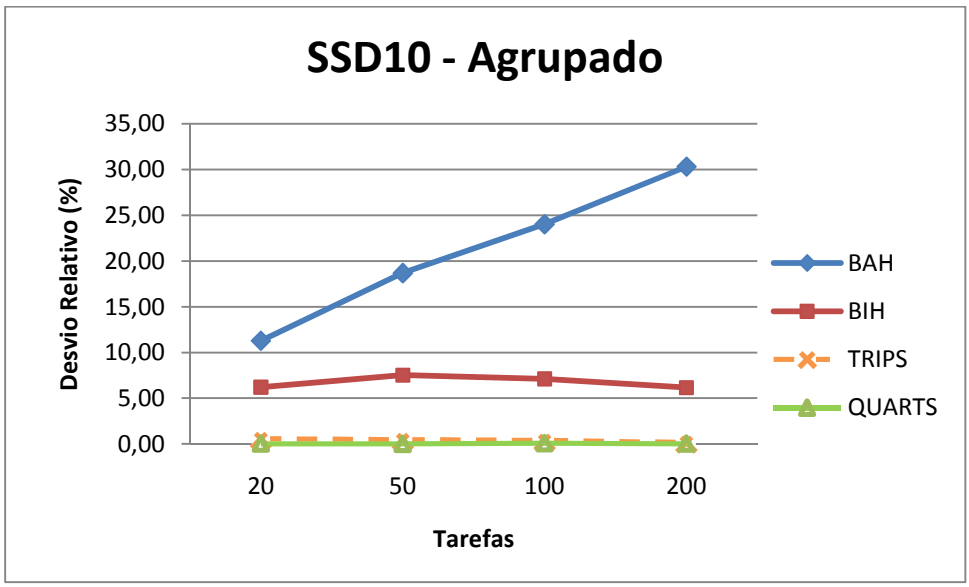

Gráfico 25 – Desvio relativo médio por máquinas agrupadas para a classe SSD-10

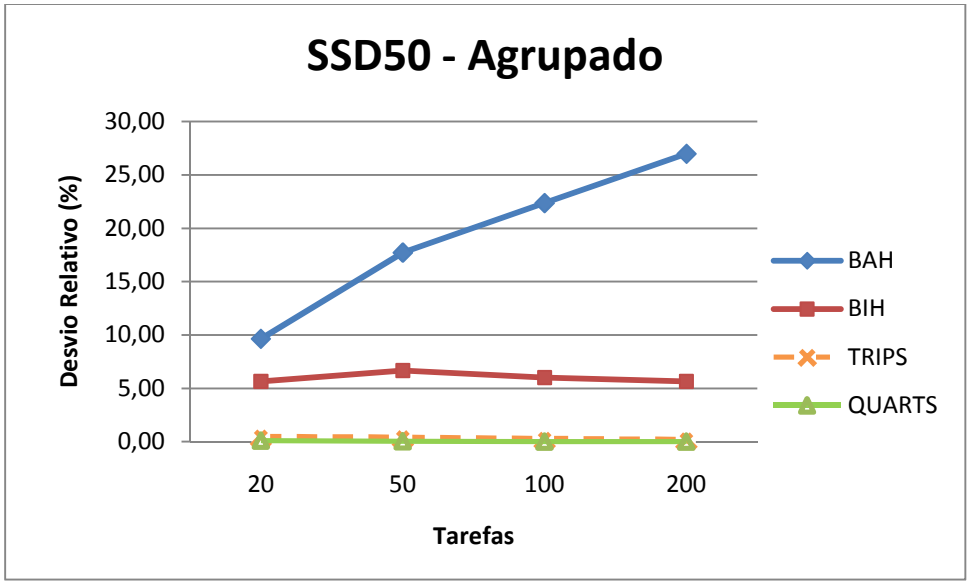

Gráfico 26- Desvio relativo médio por máquinas agrupadas para a classe SSD-50

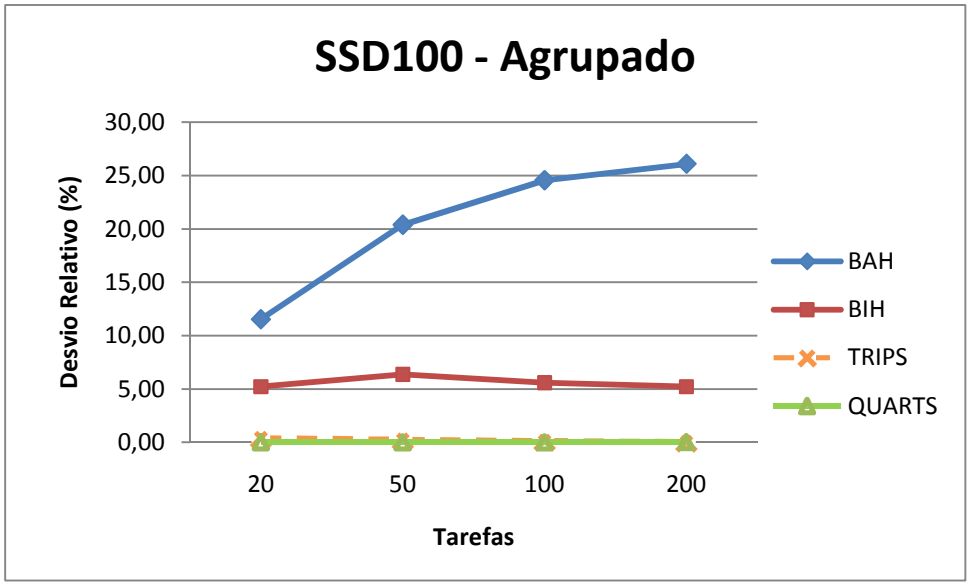

Gráfico 27- Desvio relativo médio por máquinas agrupadas para a classe SSD-100

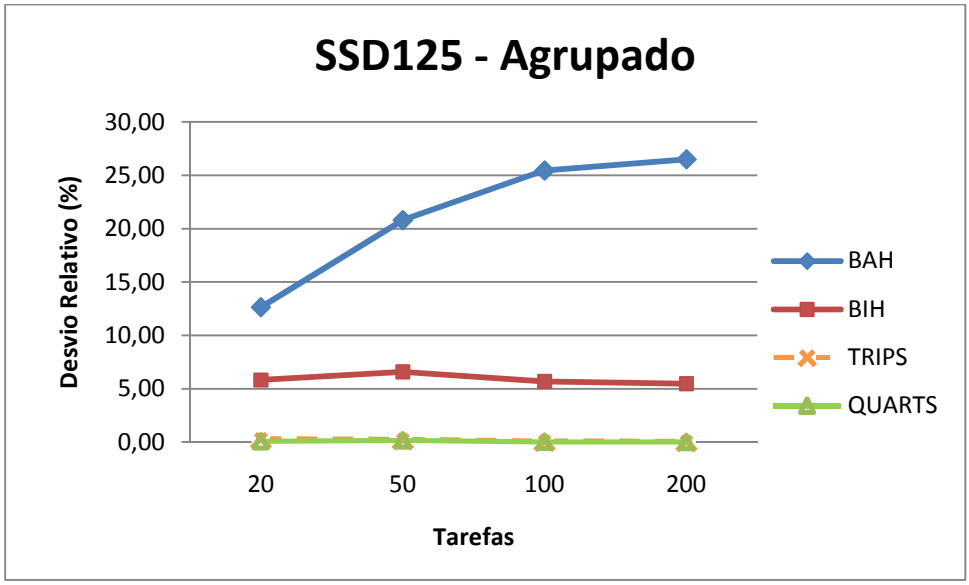

Gráfico 28 - Desvio relativo médio por máquinas agrupadas para a classe SSD-125

Através da observação dos gráficos e tabelas acima, nota-se que os desvios relativos médios dos métodos possuem faixas de valores distintas e espaçadas. Os métodos BAH e BIH, que já haviam sido considerados não competitivos, possuem os maiores desvios. O desvio relativo do BAH chega a 32% em alguns problemas, com uma média entre todos os 440 problemas de 19,93%.

Observa-se uma tendência nesse método em aumentar os valores dos desvios com o aumento do número de tarefas, indicando que o método funciona melhor para problemas de pequeno porte. Para 20 tarefas, o desvio relativo médio é de 11,28%, aumentando para 19,40% em 50 tarefas, 24,10% em 100 tarefas e 27,47% em 200 tarefas. Já os desvios relativos do método BIH encontram-se bem próximos do eixo 5,00 em todas as distribuições do tempo de *setup.* O maior desvio relativo médio desse método, de 8,11%, ocorreu no SSD-10 com 100 tarefas e 5 máquinas. Observando os gráficos, parece haver uma relação entre o número de tarefas e o desvio relativo, embora os valores variem pouco. Os desvios relativos aumentam de 20 para 50 tarefas, diminuindo com a mudança de 50 para 100 tarefas e diminuindo novamente com a mudança de 100 para 200 tarefas. Os desvios relativos médios são de 5,72%, 6,79%, 6,10% e 5,63%, respectivamente, para 20, 50, 100 e 200 tarefas. Logo, não é possível concluir se esse método funciona melhor para problemas de pequeno ou grande porte. Considerando-se a distribuição dos tempos de *setup* e a quantidade de máquinas, parece não haver relação entre elas e os desvios relativos dos métodos BAH e BIH.

Como os desvios relativos dos métodos TRIPS e QUARTS são muito inferiores aos dos métodos BAH e BIH, fez-se necessário apresentar gráficos de desvio para os dois somente, por uma questão de escala. Os Gráficos 29, 30, 31 e 32 a seguir apresentam tais desvios relativos.

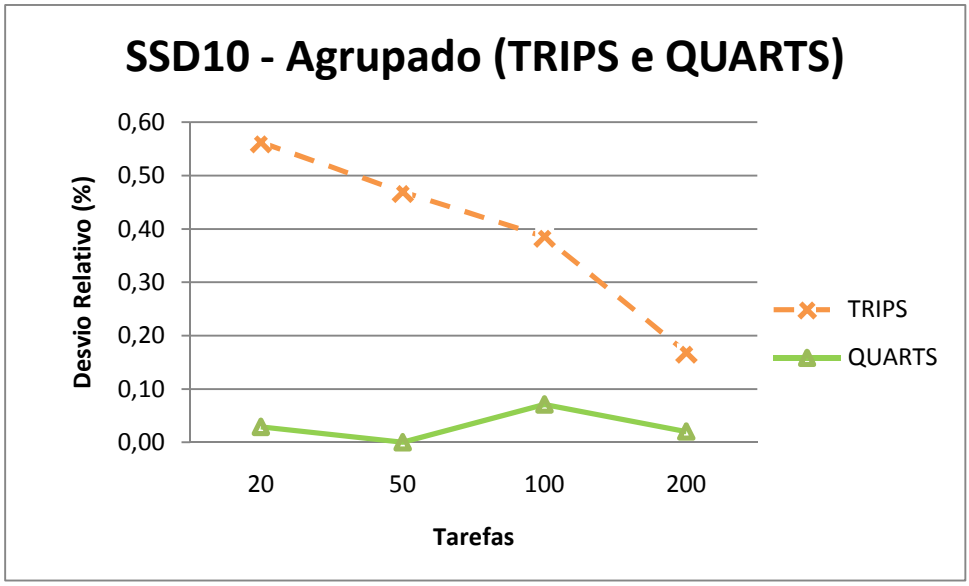

Gráfico 29 - Desvio relativo médio por máquinas agrupadas do TRIPS e do QUARTS para a classe SSD-10

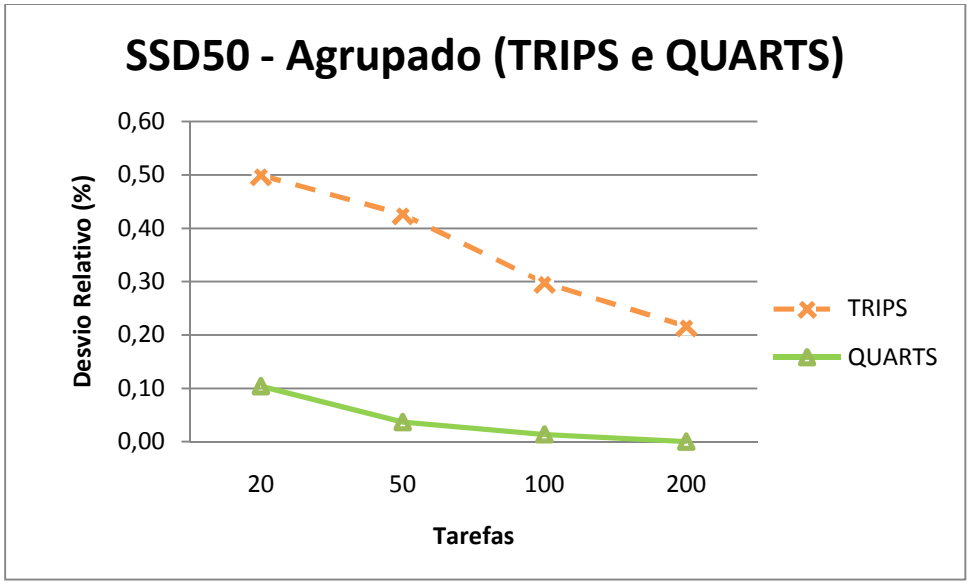

Gráfico 30 - Desvio relativo médio por máquinas agrupadas do TRIPS e do QUARTS para a classe SSD-50

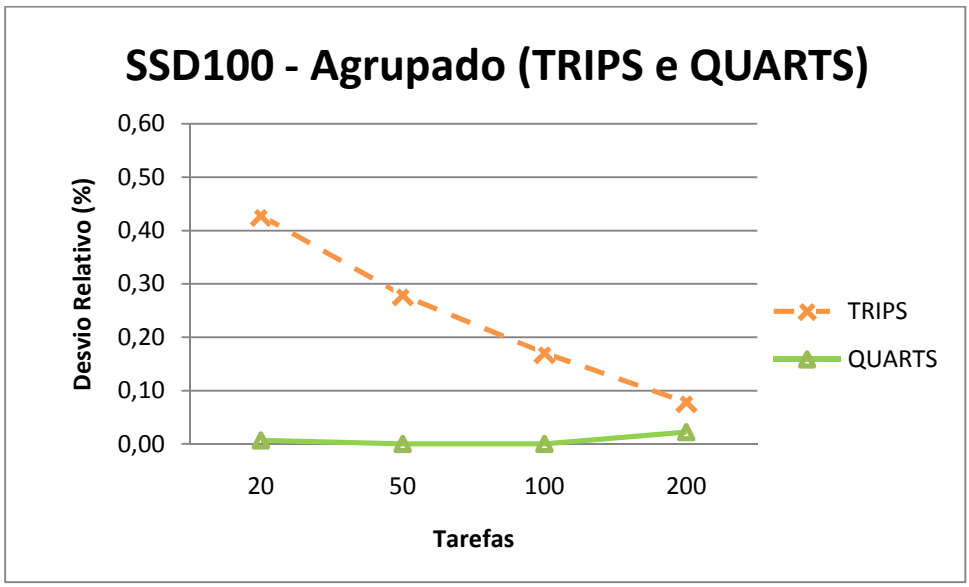

Gráfico 31 - Desvio relativo médio por máquinas agrupadas do TRIPS e do QUARTS para a classe SSD-100

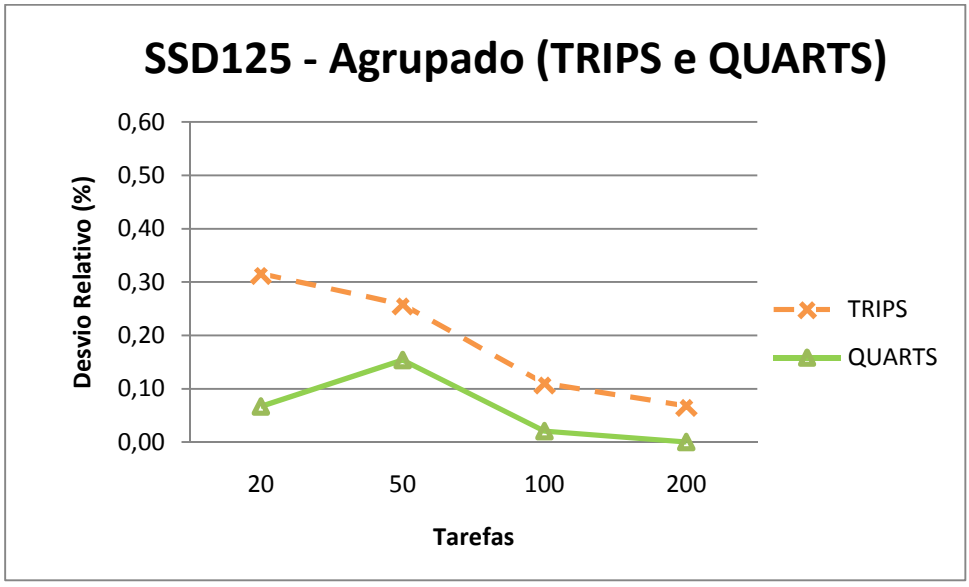

Gráfico 32 - Desvio relativo médio por máquinas agrupadas do TRIPS e do QUARTS para a classe SSD-125

Para o método TRIPS, o maior desvio relativo é de 0,71% (referente à média para o SSD-10 com 20 tarefas e 5 máquinas), e a média entre os desvios

relativos de todos os 440 problemas é de 0,31%. Observa-se uma tendência de diminuição dos desvios relativos com o aumento do número de tarefas, o que mostra que o método encontra soluções mais próximas do melhor valor encontrado quanto maior o número de tarefas. Para 20 tarefas, o desvio relativo médio desse método é de 0,45%, caindo para 0,36% para 50 tarefas, 0,24% para 100 tarefas e para 0,13% para 200 tarefas. Além disso, observa-se uma relação entre os desvios relativos e as distribuições do tempo de *setup*. O desvio relativo parece diminuir com o aumento da distribuição dos tempos de *setup.* No SSD-10, o desvio relativo médio é de 0,42%, caindo para 0,37% no SSD-50, 0,25% no SSD-100 e 0,20% no SSD-125. Isso indica que o método obtém melhores resultados quando os tempos de *setup* são altos comparados aos tempos de processamento.

Os Gráficos 29, 30, 31 e 32 ratificam a superioridade do método QUARTS, já mostrada na análise da porcentagem de sucesso. Seus desvios relativos chegam muito próximo do eixo indicativo de desvio relativo zero em todas as classes do problema, possuindo uma média geral de 0,04%. Parece não haver relação dos desvios relativos com número de tarefas, e nem com as distribuições dos tempos de *setup*.

9.2.3 Tempo Médio de Computação

As Tabelas 34, 35, 36 e 37 e os Gráficos 33, 34, 35 e 36 mostram os resultados do tempo médio de computação para as classes de tempo de *setup* (SSD-10, SSD-50, SSD-100 e SSD-125).

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,02	0,08	0,32	0,41
	10	0,05	0,09	0,32	0,43
	20	0,14	0,19	0,36	0,47
	Média	0,07	0,12	0,33	0,44
50	**5**	0,12	2,01	12,87	15,99
	10	0,31	2,35	13,00	16,65
	20	0,92	2,86	13,98	16,91
	Média	0,45	2,41	13,28	16,52
100	**5**	0,50	32,12	217,34	258,54
	10	1,25	33,01	216,13	265,60
	20	3,70	35,35	218,32	265,61
	Média	1,82	33,49	217,26	263,25
200	**10**	5,15	511,98	3494,11	4133,52
	20	14,90	515,71	3537,08	4164,64
	Média	10,03	513,85	3515,60	4149,08

Tabela 34 – Tempo médio de computação para a classe SSD-10

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,02	0,09	0,32	0,41
	10	0,05	0,10	0,32	0,43
	20	0,15	0,19	0,36	0,48
	Média	0,07	0,13	0,33	0,44
50	**5**	0,12	2,08	12,94	16,30
	10	0,31	2,27	13,07	16,75
	20	0,91	2,84	14,27	17,16
	Média	0,45	2,40	13,43	16,74

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
100	**5**	0,51	32,78	215,95	259,79
	10	1,26	32,48	216,72	265,86
	20	3,72	35,70	218,47	268,15
	Média	1,83	33,65	217,05	264,60
200	**10**	5,17	513,02	3490,31	4130,08
	20	14,82	515,82	3517,09	4158,26
	Média	10,00	514,42	3503,70	4144,17

Tabela 35 – Tempo médio de computação para a classe SSD-50

		MÉTODOS			
N	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,02	0,08	0,32	0,42
	10	0,05	0,09	0,33	0,43
	20	0,14	0,20	0,36	0,47
	Média	0,07	0,12	0,34	0,44
50	**5**	0,12	2,04	12,97	16,67
	10	0,31	2,29	13,11	16,82
	20	0,92	2,87	14,26	17,25
	Média	0,45	2,40	13,45	16,91
100	**5**	0,50	33,21	215,57	265,68
	10	1,27	32,70	217,09	269,60
	20	3,71	34,97	219,34	269,38
	Média	1,83	33,63	217,33	268,22
200	**10**	5,18	511,11	3502,32	4158,17
	20	14,91	514,98	3528,10	4166,21
	Média	10,05	513,05	3515,21	4162,19

Tabela 36 – Tempo médio de computação para a classe SSD-100

		MÉTODOS			
n	***m***	**BAH**	**BIH**	**TRIPS**	**QUARTS**
20	**5**	0,02	0,09	0,32	0,43
	10	0,05	0,09	0,32	0,44
	20	0,14	0,20	0,36	0,48
	Média	0,07	0,13	0,33	0,45
50	**5**	0,13	2,05	12,96	16,55
	10	0,31	2,22	13,10	16,93
	20	0,92	2,85	14,10	17,47
	Média	0,45	2,37	13,39	16,98
100	**5**	0,51	32,97	218,75	267,20
	10	1,27	32,78	217,73	271,06
	20	3,72	35,02	218,98	270,50
	Média	1,83	33,59	218,48	269,58
200	**10**	5,15	512,84	3522,19	4165,47
	20	14,87	515,03	3518,97	4178,01
	Média	10,01	513,94	3520,58	4171,74

Tabela 37 – Tempo médio de computação para a classe SSD-125

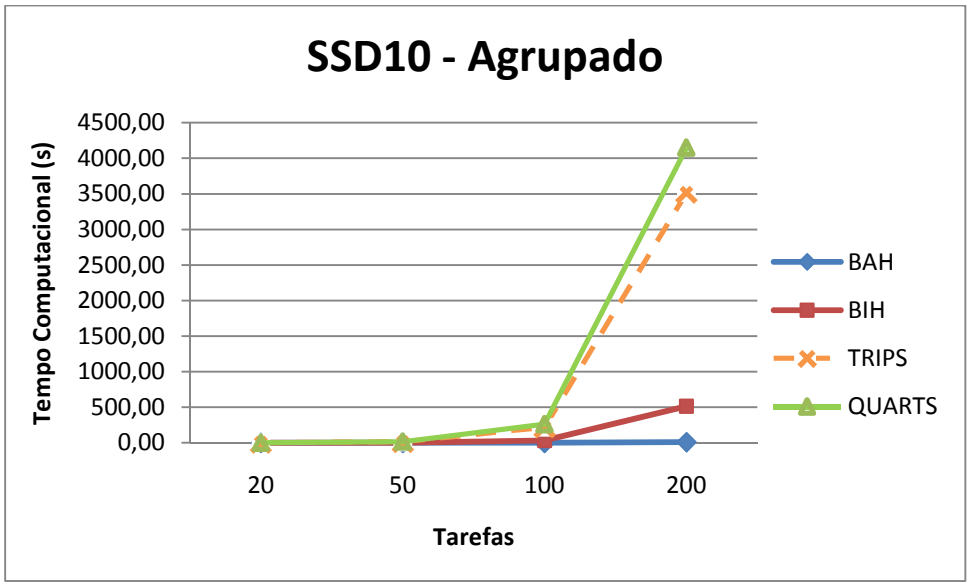

Gráfico 33 – Tempo médio de computação por máquinas agrupadas para a classe SSD-10

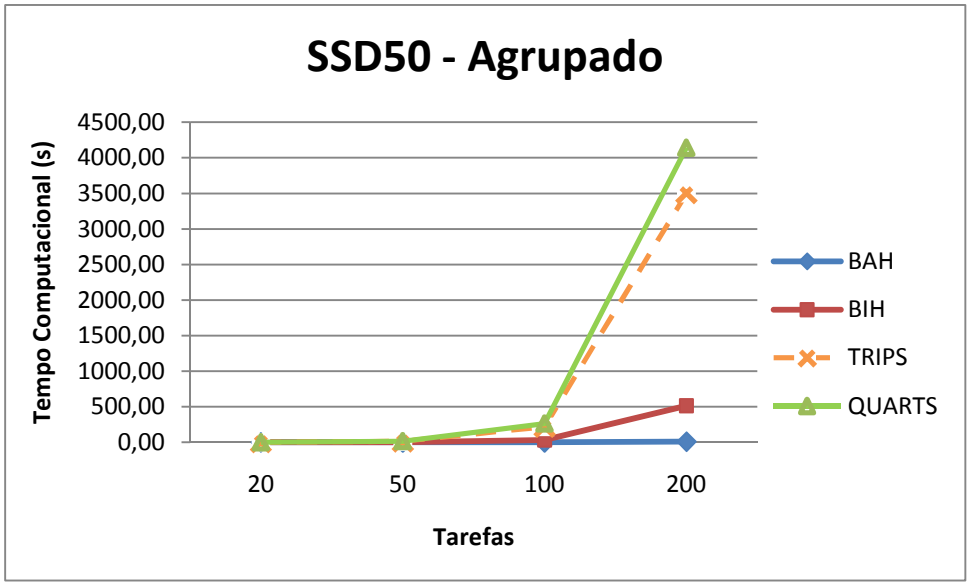

Gráfico 34 – Tempo médio de computação por máquinas agrupadas para a classe SSD-50

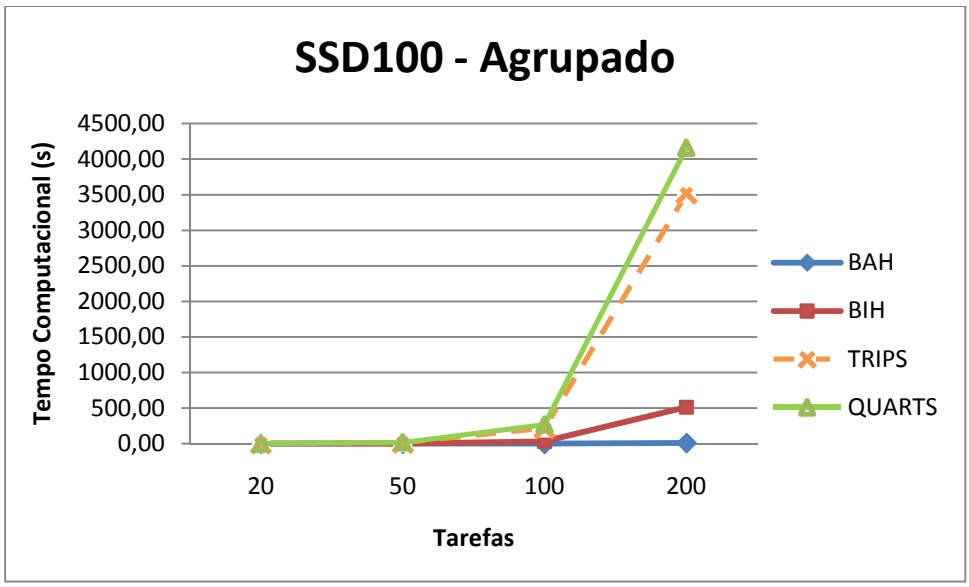

Gráfico 35 - Tempo médio de computação por máquinas agrupadas para a classe SSD-100

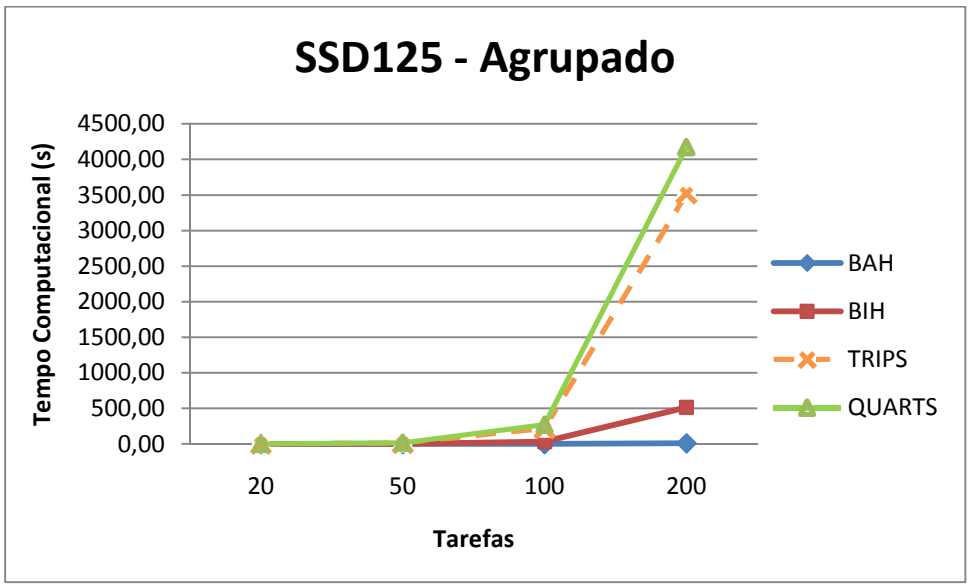

Gráfico 36 - Tempo médio de computação por máquinas agrupadas para a classe SSD-125

Em termos do tempo médio de computação, percebe-se que o BAH é o método mais rápido, seguido pelo BIH. O tempo de computação médio do BAH é de 2,46s, enquanto do BIH é de 103,27.

Comparando-se o TRIPS com o QUARTS, observa-se que o TRIPS é um pouco mais rápido do que o QUARTS. O tempo médio de computação do TRIPS, considerando-se todos os 440 problemas, é de aproximadamente 702s, enquanto do QUARTS é de 833s. Observando-se os gráficos e as tabelas acima, nota-se que o tempo computacional desses dois métodos é da mesma ordem, ou seja, a diferença entre eles não é significativa.

Através das tabelas, percebe-se que o método BAH é o único que aparenta ser afetado pela variação do número de máquinas. De um problema com 10 máquinas para um de 20 máquinas, ele tem seu tempo computacional aumentado em aproximadamente três vezes. Observando-se os gráficos, pode-se perceber que o tempo computacional de todos os métodos é influenciado pelo

número de tarefas do problema. Existe uma tendência de o tempo computacional ser proporcional ao número de tarefas, ou seja, quanto maior o número de tarefas, maior o tempo computacional de cada método. Não foi observada relação entre o tempo computacional e as distribuições dos tempos de *setup*.

Finalmente, dentre todos os métodos avaliados nesta experimentação para o critério do *total flowtime*, verifica-se que o método QUARTS é o que apresenta melhores resultados em relação à qualidade da solução. Fazendo uma média entre os 440 problemas testados, sua porcentagem de sucesso é de 95,10% e seu desvio relativo é de 0,04%, enquanto que os métodos BAH, BIH e TRIPS possuem porcentagens de sucesso de 0%, 0,23% e 12,27%, e desvio relativo de 19,93%, 6,10% e 0,31%, respectivamente. Com respeito ao tempo computacional, apesar do método TRIPS ser um pouco mais rápido do que o QUARTS, essa diferença é insignificante. Assim, pode-se concluir que a heurística QUARTS é a mais indicada para o critério de minimização do *total flowtime*.

CAPÍTULO 10

CONSIDERAÇÕES FINAIS

Este trabalho abordou o problema de programar *n* tarefas em *m* máquinas em um *no-wait flowshop* com tempos de *setup* dependentes da sequência. Os critérios de minimização utilizados foram o *makespan* e o *total flowtime*, ambos de grande importância nos sistemas de manufatura. O *makespan* relaciona-se à utilização eficiente dos recursos (máquinas) e o *total flowtime* busca minimizar o estoque em processamento (WIP).

Uma extensa revisão bibliográfica foi realizada, abordando os trabalhos com métodos de solução exata, heurísticos construtivos e meta-heurísticos, para o *flowshop* com tempos de *setup* dependentes da sequência, para o *no-wait flowshop* básico e para o *no-wait flowshop* com tempos de *setup*.

Os principais métodos existentes na literatura, BAH e BIH, de Bianco, Dell´Olmo e Giordani (1999) e TRIPS, de Brown, McGarvey e Ventura (2004) foram estudados e implementados para o problema. Em seguida, foram propostos dois métodos heurísticos construtivos, visto que o problema abordado é *NP-hard*.

O método para minimização do *makespan* foi chamado de GAPH, sendo desenvolvido a partir de uma propriedade estrutural derivada dos trabalhos de Wismer (1972) e Bianco, Dell´Olmo e Giordani (1999). Os resultados obtidos na experimentação computacional evidenciaram a qualidade desse método proposto, considerando-se todos os problemas testados, uma vez que ele apresentou porcentagem de sucesso média de 99,77%, e desvio relativo médio de 0%,

enquanto que os métodos BIH, TRIPS e BAH apresentaram porcentagens de sucesso médias de 60,23%, 0,23% e 0%, desvios relativos médios de 0,59%, 5,88% e 10,02%, respectivamente. Com respeito ao tempo computacional, o método GAPH se mostrou mais lento que o BIH e mais rápido do que o TRIPS.

O método para minimização do *total flowtime* foi desenvolvido baseado no método TRIPS, com algumas adaptações no intuito de proporcionar melhorias aos resultados. A principal modificação realizada foi que, ao invés de trios como o TRIPS, a heurística QUARTS quebra o problema em quartetos. A experimentação computacional mostrou a superioridade desse método em relação à qualidade de solução. Fazendo uma média entre os 440 problemas testados, o método apresentou uma porcentagem de sucesso de 95,10% e desvio relativo médio de 0,04%, enquanto que os métodos BAH, BIH e TRIPS apresentaram porcentagens de sucesso de 0%, 0,23% e 12,27%, e desvios relativos de 19,93%, 6,10% e 0,31%, respectivamente. Com respeito ao tempo computacional, apesar do método TRIPS ser um pouco mais rápido do que o QUARTS, ambos os métodos possuem resultados da mesma ordem, tornando essa diferença insignificante.

Assim, conclui-se que os métodos propostos GAPH e QUARTS são as heurísticas construtivas mais indicadas para o NWFS com SDST com critério de minimização do *makespan* e do *total flowtime*, respectivamente. Estas duas heurísticas são a principal contribuição deste trabalho.

Trabalhos futuros podem ser conduzidos na implementação dos métodos GAPH e QUARTS com o critério de minimização bi-critério, e também na adaptação desses métodos para o NWFS com tempo de *setup* independentes da sequência.

REFERÊNCIAS

ADIRI, T.; POHORYLES, D. Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. **Naval Research Logistics Quartely**, Hoboken, vol. 29, n. 3, p. 495-504, set. 1982.

AHN,S.H. A study on an algorithm to minimize makespan of sequence-dependent jobs. **Journal of the Korean Institute of Management Science**, Seoul, vol. 12, n. 1, p. 77-87, 1995.

ALDOWAISAN, T. A new heuristic and dominance relations for no-wait flowshops with setups. **Computers & Operations Research**, Oxford, vol. 28, n. 6, p. 563-584, mai. 2001.

_____.; ALLAHVERDI, A. New heuristics for m-machine no-wait flowshop to minimize total completion time. **Omega, The international Journal of Management Science**, Oxford, vol. 32, n. 5, p. 345-352, out. 2004a.

_____.; ALLAHVERDI, A. New heuristics for no-wait flowshops to minimize makespan. **Computers & Operations Research**, Oxford, vol. 30, n. 8, p. 1219-1231, jul. 2003.

_____.; ALLAHVERDI, A. Three-machine no-wait flowshop with separate setup and removal times to minimize total completion time. **International Journal of Industrial Engineering**, Houston, vol. 11, n. 2, 113–123., jun. 2004b.

_____.; ALLAHVERDI, A. Total flowtime in no-wait flowshops with separated setup times. **Computers & Operations Research**, Oxford, vol. 25, n. 9, p. 757-765, set. 1998.

ALLAHVERDI, A.; AL-ANZI, F. Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. **Computers & Operations Research**, Oxford, vol. 29, n. 8, p. 971–994, jul. 2002.

_____.; ALDOWAISAN, T. Minimizing total completion time in a no-wait flowshop with sequence-dependent additive changeover times. **Journal of the Operational Research Society**, Oxford, vol. 52, n. 4, p. 449-462, abr. 2001.

_____.; ALDOWAISAN, T. No-wait and separate setup three-machine flowshop with total completion time criterion. **International Transactions in Operational Research**, vol. 7, n. 3, p. 245-264, mai. 2000.

_____.; ALDOWAISAN, T. No-wait flowshops with bicriteria of makespan and maximum lateness. **European Journal of Operational Research**, Amsterdam, vol. 152, n. 1, p. 132-147, jan. 2004.

_____. et al. A survey of scheduling problems with setups times or costs. **European Journal of Operational Research**, Amsterdam, vol. 187, n. 3, p. 985-1032, jun. 2008.

_____.; GUPTA, J. N. D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **Omega, The international Journal of Management Science**, Oxford, vol. 27, n. 2, p. 219-239, abr. 1999.

_____.; SOROUSH, H. M. The significance of reducing setup times/setup costs. **European Journal of Operational Research**, Amsterdam, vol. 187, n. 3, p. 978-984, jun. 2008.

ANDRÉS, C. et al. A multi-objective genetic algorithm to solve the scheduling problem in flowshops with sequence dependent setup times. In: THE INTERNATIONAL CONFERENCE ON INDUSTRIAL ENGINEERING AND SYSTEMS MANAGEMENT, 2005, Marrakech. **Proceedings...** Marrakech: International Institute of Innovation, Industrial Engineering and Entrepreneurship, 2005, p. 630–635.

BERTOLISSI, E. A simple no-wait flowshop scheduling for the no-wait flowshop problem. In: THE INTERNATIONAL CONFERENCE ON COMPUTER-AIDED PRODUCTION ENGINEERING, 15., 1999, Durham. **Proceedings...** Durham: University of Durham Publishers, 1999, p. 750–755.

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

BIANCO, L.; DELL'OLMO, P.; GIORDANI, S. Flow shop no-wait scheduling with sequence dependent setup times and release dates. **INFOR Journal**, Toronto, vol. 37, n. 1, p. 3–19, 1999.

BONNEY, M. C.; GUNDRY, S. W. Solutions to constrained flowshop sequencing problem. **Operational Research Quarterly**, London, vol. 27, n. 4, p. 869-883, 1976.

BROWN, S. I.; MCGARVEY, R.; VENTURA, J. A. Total flowtime and makespan for a no-wait m-machine flowshop with set-up times separated. **Journal of the Operational Research Society**, Oxford, vol. 55, n. 6, p. 614-621, mai. 2004.

CAMPBELL, H. G.; DUDEK, R. A.; SMITH, M. L. A heuristic algorithm for the n job, m machine sequencing problem. **Management Science**, Rhode Island, vol. 16, n. 10, p. B630–B637, jun. 1970.

CHANG, J. et al. Scheduling a no-wait flowshop with separated setup and removal times. In: THE WORLD CONGRESS ON INTELLIGENT CONTROL AND AUTOMATION, 5., 2004, Hangzou. **Proceedings...** Hangzou: 2004, vol. 4, p. 2877–2880.

CHANG, J.; YAN, W.; SHAO, H. Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. In: AMERICAN CONTROL CONFERENCE, 2004, Boston. **Proceedings...** Boston: IEEE Control Systems Magazine, 2004, vol. 2, p. 1412–1416.

CHANG, P. C.; HSIEH, J. C.; WANG, Y. W. Genetic algorithms applied in BOPP film scheduling problems: minimizing total absolute deviation and setup times. **Applied Soft Computing**, Amsterdam, vol. 3, n. 2, p. 139–148, set. 2003.

CHEN, C.-L.; NEPPALLI, R. V.; ALJABER, N. Genetic algorithms applied to the continuous flow shop problem. **Computers & Industrial Engineering**, Oxford, vol. 30, n. 4, p. 919-929, set. 1996.

CHENG, T. C. E.; GUPTA, J. N. D.; WANG, G. A review of flowshop scheduling research with setup times. **Production and Operations Management**, vol. 9, n. 3, set. 2000.

CORWIN, B. D.; ESOGBUE, A. O. Two machine flow shop scheduling problems with sequence dependent setup times: a dynamic programming approach. **Naval Research Logistics Quarterly**, Hoboken, vol. 21, n. 3, p. 515–524, set. 1974.

CROWDER, B. **Minimizing the makespan in a flexible flowshop with sequence dependent setup times, uniform machines, and limited buffers.** 2006. 136f. Dissertação (Master of Science in Industrial Engineering) - Department of Industrial and Management Systems Engineering, West Virginia University, Morgantown, 2006.

DANNENBRING, D.G. An Evaluation of Flow-Shop Sequencing Heuristics. **Management Science**, Rhode Island, vol. 23, n. 11, p. 1174-1182, jul. 1977.

DAS, S. R.; GUPTA, J. N. D.; KHUMAWALA, B.M. A saving index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. **Journal of the Operational Research Society**, Oxford, vol. 46, n. 11, p. 1365-73, nov. 1995.

DEMAN, J. M. V.; BAKER, K. R. Minimizing mean flowtime in the flow shop with no intermediate queues. **IIE Transactions**, Norcross, vol. 6, n. 1, p. 28-34, mar. 1974.

DEMIRKOL, E.; UZSOY, R. Decomposition methods for reentrant flow shops with sequence-dependent setup times. **Journal of Scheduling**, Amsterdam, vol. 3, n. 3, p. 155–177, mai. 2000.

DILEEPAN, P. A note on minimizing maximum lateness in a two-machine no-wait flowshop. **Computers & Operations Research**, Oxford, vol. 31, n. 12, p. 2111-2115, out. 2004.

DUDEK, R. A.; PANWALKAR, S. S.; SMITH, M. L. The lessons of flowshop scheduling research. **Operations Research**, Hannover, vol. 40, n. 1, p. 7–13, jan./fev. 1992.

FINK, A.; VOβ, S. Solving the continuos flow-shop scheduling problem by metaheuristics. **European Journal of Operational Research**, Amsterdam, vol. 151, n. 2, p. 400-414, dez. 2003.

FISCHETTI, M.; LAPORTE, G.; MARTELLO, S. The Delivery Man Problem and Cumulative Matroids. **Operations Research**, Hannover, vol. 41, n. 6, p. 1055-1064, nov./dez. 1993.

FLYNN, B. B. The effects of setup time on output capacity in cellular manufacturing. **International Journal of Production Research**, London, vol. 25, n. 12, p. 1761–1772, dez. 1987.

FONDREVELLE, J.; ALLAHVERDI, A.; OULAMARA, A. Two-machine no-wait flowshop scheduling problem to minimize maximum lateness with separate setup and removal times. **International Journal of Agile Manufacturing**, vol. 8, p. 165–174, 2005.

FRAMINAN, J. M.; LEISTEN, R. An efficient constructive heuristic for flowtime minimization in permutation flow shops. **Omega, The International Journal of Management Science**, Oxford, vol. 31, n. 4, p. 311-317, jun. 2003.

_____.; NAGANO, M. S. Evaluating the performance for makespan minimisation in no-wait flowshop sequencing. **Journal of Materials Processing Technology**, Oxford, vol. 197, n. 1-3, p. 1-9, fev. 2008.

_____.; NAGANO, M. S.; MOCCELLIN, J. V. An efficient heuristic for total flowtime minimization in no-wait flowshops. **International Journal of Advanced Manufacturing Technology**, London, vol. 46, n. 9-12, p. 1049-1057, fev. 2010.

FRANÇA, P. M.; TIN JR, G.; BURIOL, L. S. Genetic Algorithms for the no-wait flowshop sequencing problem with time restrictions. **International Journal of Production Research**, London, vol. 44, n.5, p. 939-957, 2006.

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

GAREY, M. R.; JOHNSON, D. S. **Computers and intractability**: a guide to the theory of NP-completeness. W. H Freeman and Company, 1979.

GENDREAU, M.; LAPORTE, G.; GUIMARÃES, E. M. A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. **European Journal of Operational Research**, Amsterdam, vol. 133, n. 1, p. 183–189, ago. 2001.

GILMORE, P. C.; GOMORY, R. E. Sequencing a one-state variable machine: a solvable case of the traveling salesman problem. **Operations Research**, Hannover, vol. 12, n. 5, p. 655-679, set./out. 1964.

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

_____.; PEMPERA, J. The permutation flowshop problem with blocking. A tabu search approach. **Omega, The International Journal of Management Science**, Oxford, vol. 35, n. 3, p. 302-311, jun. 2007.

GRAHAM, R. L. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. **Annals of Discrete Mathematics**, vol. 5, n. 2, p. 287-326, 1979.

GUPTA, J.N.D. A general algorithm for the n x m flowshop scheduling problem. **International Journal of Production Research**, London, v**ol. 7,** n. 3, p. 241–247, 1968.

_____. A search algorithm for the generalized scheduling problem. **Computers & Operations Research**, Oxford, vol. 2, n. 2, p. 83-90, 1975.

_____. Flowshop schedules with sequence dependent setup times. **Journal of the Operational Research Society of Japan**, Tokyo, vol. 29, n. 3, p. 206–219, 1986.

_____.; DARROW, W. P. Approximate schedules for the two-machine flowshop with sequence dependent setup times. **Indian Journal of Management Systems**, Delhi, vol. 1, p. 6-11, 1985.

_____.; DARROW, W. P. The two-machine sequence dependent flowshop scheduling problem. **European Journal of Operational Research**, Amsterdam, vol. 24, n. 3, p. 439-446, mar. 1986.

_____.; DAS, S. R.; GHOSH, S. Flowshop **Scheduling with Sequence Dependent Setup Times**. Muncie: Department of Management - Ball State University, 1995. (Working paper, IN 47306).

GUPTA, J. N. D.; STRUSEVICH, V. A.; ZWANEVELD, C. M. Two-stage no-wait scheduling models with setup and removal times separated. **Computers & Operations Research**, Oxford, vol. 24, n. 11, p. 1025-1031, nov. 1997.

HALL, N. G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. **Operations Research**, Hannover, vol. 44, n. 3, p. 510-525, mai./jun. 1996.

HELSGAUN, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. **European Journal of Operational Research**, Amsterdam, vol. 126, n. 1, p. 106–120, out. 2000.

HWANG, H.; SUN, J. U. Production sequencing problem with reentrant work flows and sequence dependent setup times. **Computers & Industrial Engineering**, vol. 33, n. 3-4, p. 773–776, dez. 1997.

_____.; SUN, J. U. Production sequencing problem with reentrant work flows and sequence dependent setup times. **International Journal of Production Research**, London, vol. 36, n. 9, p. 2435-2450, set. 1998.

JOHNSON, L. A.; MONTGOMERY, D. C. **Operations research in production planning, scheduling, and inventory control**. New York: John Wiley & Sons, 1974.

JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly**, Hoboken, v. 1, n. 1, p. 61-68, mar. 1954.

KANG, B. S.; Markland, R. E. Evaluation of scheduling techniques for solving flowshop problems with no intermediate storage. **Journal of Operations Management**, vol. **7, n. 3-4,** p. 1–24, dez. 1988.

KARG, R. L.; THOMPSON, G. L. A Heuristic Approach to Solving Travelling Salesman Problems. **Management Science**, Rhode Island, vol. 10, n. 2, p. 225-248, jan. 1964.

KIM, D. W. et al. Unrelated parallel machine scheduling with setup times using simulated annealing. **Robotics and Computer-Integrated Manufacturing**, Panama, vol. 18, n. 4, p. 223–231, out. 2002.

KING, J. R.; SPACHIS, A. S. Heuristics for flowshop scheduling. **International Journal of Production Research***,* London, vol. 18, n. 3, p. 345-357, fev. 1980.

KRAJEWSKI, L. J. et al. Kanban, MRP, and shaping the manufacturing environment. **Faculty of Management Science**, London, vol. 33, n. 1, p. 39–57, 1987.

KUMAR, A. et al. Psycho-Clonal algorithm based approach to solve continuous flow shop scheduling problem. **Expert Systems with Applications**, Los Angeles, vol. 31, n. 3, p. 504-514, out. 2006.

LAGUNA, M. A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times. **IIE Transactions**, Norcross, vol. 31, n. 2, 125–134, fev. 1999.

LAHA, D.; CHAKRABORTY, U. K. A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. **International Journal of Advanced Manufacturing Technology**, London, vol. 41, n. 1-2, p. 97-109, mar. 2009.

LI, X.; WANG, Q.; WU, C. Heuristic for no-wait flow shops with makespan minimization. **International Journal of Production Research**, London, vol. 46, n. 9, p. 2519-2530, mai. 2008.

LIN, H. T.; LIAO, C. J. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. **International Journal of Production Economics**, Amsterdam, vol. 86, n. 2, p. 133–143, nov. 2003.

MACCARTHY, B. L.; LIU, J. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. **International Journal of Production Research**, London, v. 31, n.1, p. 59-79, jan. 1993.

MADDUX III, H. S.; GUPTA, J. N. D. Scheduling intermediate and finished products in a two-stage flowshop with sequence dependent setup times. In: ANNUAL MEETING OF THE DECISION SCIENCES INSTITUTE, 34., 2003, Washington DC. **Proceedings...** Washington DC: The Decision Sciences Institute, 2003, p. 1579–1585.

MOCCELLIN, J. V.; NAGANO, M. S. Flow shop com máquinas paralelas genéricas. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 35., 2003, Natal. **Anais...** Rio de Janeiro: Sociedade Brasileira de Pesquisa Operacional, 2003, 1 CD-ROM.

_____.; NAGANO, M. S. Uma propriedade estrutural do problema de programação da produção flow shop permutacional com tempos de setup. **Pesquisa Operacional**, Rio de Janeiro, vol. 27, n. 3, p. 487-515, set./dez. 2007.

MONKS, J. G. **Administração da produção.** São Paulo: MCGraw-Hill, 1987.

NAINI, S. H. H. **Scheduling a** flowshop **manufacturing cell with sequence dependent setup times.** 2007. 99f. Dissertação (Master of Science) - Departament of Mechanical and Manufacturing Engineering, University of Mantoba, Mantoba, 2007.

NAGANO, M. S.; MOCCELLIN, J. V. A high quality solution constructive heuristic for flow shop sequencing. **Journal of the Operational Research Society**, Oxford, v. 53, n. 12, p. 1374-1379, dez. 2002.

_____.; MOCCELLIN, J. V. Reducing mean flow time in permutation flow shop. **Journal of the Operational Research Society**, Oxford, vol. 59, n. 7, jul. 2008.

_____.; MOCCELLIN, J. V. Redução do estoque em processamento em sistemas de produção flowshop. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 37., 2005, Gramado. **Anais...** Gramado: Sociedade Brasileira de Pesquisa Operacional, 2005, CD-ROM.

NAWAZ, M.; ENSCORE JR., E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega, The International Journal of Management Science**, Oxford, vol. 11, n. 1, p. 91-95, 1983.

NORMAN, B. A.; BEAN, J. C. A genetic algorithm methodology for complex scheduling problems. **Naval Research Logistics**, Hoboken, vol. 46, n. 2, p. 199–211, mar. 1999.

OLIVEIRA, A.C.M.; LORENA, L.A.N. Detecting promising areas by evolutionary clustering search. In: BAZZAN, A.L.C., LABIDI, S. **SBIA 2004: Advances in Artificial Intelligence. São Luis:** Springer Heidelberg, 2004. LNAI 3171, p. 385–394.

OSMAN, I.H.; POTTS, C.N. Simulated annealing for permutation flow-shop scheduling. **Omega, The International Journal of Management Science**, Oxford, vol. 17, n. 6, p. 551–557, 1989.

PAN, Q.-P.; TASGETIREN, M. F.; LIANG, Y.-C. **A discrete differential evolution algorithm for the permutation** flowshop **scheduling problem. Computers & Industrial Engineering**, Oxford, vol. 55, n. 4, p. 795-816, nov. 2008.

PANWALKAR, S. S.; DUDEK, R. A.; SMITH, M. L. Sequencing research and the industrial scheduling problem. In: THEORY OF SCHEDULING AND ITS APPLICATIONS, 1973, New York. **Symposium...** New York: Springer-Verlag, 1973, p. 29–38.

PAPADIMITRIOU, C. H.; KANELLAKIS, P. C. Flowshop scheduling with limited temporary storage. **Journal of the ACM**, New York, vol. 27, n. 3, p. 533-549, jul. 1980.

PARTHASARATHY, S.; RAJENDRAN, C. A simulated annealing heuristic for scheduling to minimize weighted tardiness in a flowshop with sequence dependent setup times of jobs - a case study. **Production Planning & Control**, Abington, vol. 8, n. 5, p. 475-483, jul. 1997a.

_____.; RAJENDRAN, C. An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence dependent setup times of jobs. **International Journal of Production Research**, London, vol. 49, n. 3, p. 255-263, mai. 1997b.

PEKNY, J. F.; MILLER, D. L. Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristics methods. **Computers & Chemical Engineering**, Shannon, vol. 15, n. 11, p. 741-748, nov. 1991.

PINEDO, M. **Scheduling:** theory, algorithms, and systems. New Jersey: Prentice Hall, 2008.

PRESS, W. H. et al. **Numerical Recipes In C.** Cambridge: Cambridge University Press, 1988.

PROUST, C.; GUPTA, J. N. D.; DESCHAMPS, V. Flowshops scheduling with set-up, processing and removal times separated. **International Journal of Production Research**, London, v. 29, n. 3, p. 479-493, mar. 1991.

RAJAGOPALAN D.; KARIMI, I. A. Scheduling in serial mixed-storage multiproduct processes with transfer and set-up times. In: REKLAITIS, G. V., SPRIGGS, H. D. **Foundations of Computer Aided Process Operations**. New York: Elsevier Science Publishers, 1987. p. 679–686.

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

_____.; CHAUDHURI, D. Heuristic algorithms for continuos flow-shop problem. **Naval Research Logistics**, Hoboken, vol. 37, n. 5, p. 695-705, 1990.

_____.; ZIEGLER, H. Heuristics for scheduling in a flowshop with setup, processing and removal times separated. **Production Planning & Control**, Abington, vol. 8, n. 6, p. 568-576, set. 1997.

_____.; ZIEGLER, H. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. **European Journal of Operational Research**, Amsterdam, vol. 149, n. 3, p. 513–522, set. 2003.

REDDI, S. S.; RAMAMOORTHY, C. V. On the flow-shop sequencing problem with no-wait in process. **Operational Research Quarterly**, London, vol. 23, n. 3, p. 323-331, 1972.

RIBEIRO FILHO, G.; NAGANO, M. S.; LORENA, L.A.N. Hybrid Evolutionary Algorithm for Flowtime Minimisation in No-Wait Flowshop Scheduling. In: GELBUKH, A.; MORALES, A. F. K. **MICAI 2007: Advances in Artificial Intelligence.** Aguascalientes: Springer Berlin/Heidelberg, 2007. LNAI 4827, p. 1099-1109.

RICKEL, J. An intelligent tutoring framework for task-oriented domains. In: CONFERENCE ON INTELLIGENT TUTORING SYSTEMS, 1988, Montreal. **Proceedings...** Montreal: Lecture notes in computer science, 1988, p. 109-115.

RÍOS-MERCADO, R. Z.; BARD, J. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. **IIE Transactions**, Norcross, vol. 31, n. 8, p. 721–731, ago. 1999a.

_____.; BARD, J. An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times. **Journal of Heuristics**, Amsterdam, vol. 5, n. 1, p. 53–70, abr. 1999b.

_____.; BARD, J. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. **Computers & Operations Research**, Oxford, vol. 25, n. 5, p. 351–366, mai. 1998a.

_____.; BARD, J. Heuristics for the flow line problem with setup costs. **European Journal of Operational Research**, Amsterdam, vol. 110, n. 1, p. 76-98, out. 1998b.

_____.; BARD, J .F. The flow shop scheduling polyhedron with setup times. **Journal of Combinatorial Optimization**, Amsterdam, vol. 7, n. 3, 291–318, set. 2003.

_____.; BARD, J .F. **The flowshop scheduling polyhedron with setup times.** Austin: University of Texas at Austin, 1996. (Technical Report ORP96-07)

RUIZ, R.; ALLAHVERDI, A. No-wait flowshop with separate setup times to minimize maximum lateness. **The International Journal of Advanced Manufacturing Technology**, London, vol. 35, n. 5-6, p. 551-565, dez. 2007b.

_____.; ALLAHVERDI, A. Some effective heuristics for no-wait flowshops with setup times to minimize total completion time. **Annals of Operations Research**, Amsterdam, vol. 156, n. 1, p. 143-171, dez. 2007a.

_____.; MAROTO, C.; ALCARAZ, J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. **European Journal of Operational Research**, Amsterdam, vol. 165, n. 1, p. 34-54, ago. 2005.

_____.; STÜTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop with makespan and weighted tardiness objectives. **European Journal of Operational Research**, Amsterdam, vol. 187, n. 3, p. 1143-1159, jun. 2008.

SCHALLER, J.; GUPTA, J. N. D.; VAKHARIA, A. J. Scheduling a flowline manufacturing cell with sequence dependent *family setup times.* ***European Journal of Operational Research****, Amsterdam, vol. 125, n. 2, p. 324–339, set. 2000.*

SETHANAN, K. **Scheduling Flexible** Flowshop**s with Sequence Dependent Setup Times.** 2001. 179f. Tese (Doctor of Philosophy in Decision Sciences and Production Systems) - *Department of Industrial and Management Systems Engineering,* West Virginia University, Morgantown, 2001.

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 & Industrial Engineering**, vol. 47, n. 2-3, p. 181-193, nov. 2004.

SIDNEY, J. B.; POTTS, C. N.; SRISKANDARAJAH, C. A heuristic for scheduling two-machine no-wait flow shops with anticipatory setups. **Operations Research Letters**, vol. 26, n. 4, p. 165-173, mai. 2000.

SILVA, E. L.; MENEZES, E. M. **Metodologia de pesquisa e elaboração de dissertação.** 3. Ed. Florianópolis: Laboratório de ensino a distância da UFSC, 2001. 121p.

SIMONS, J. V. Heuristics in flow shop scheduling with sequence dependent setup times. **Omega, The International Journal of Management Science**, Oxford, vol. 20, n. 2, p. 215-225, mar. 1992.

SLACK, N. et al. **Administração da produção**. São Paulo: Atlas, 1996. 726p.

SONMEZ, A. I.; BAYKASOGLU, A. New dynamic programming formulation of (n x m) flowshop sequencing problems with due dates. **International Journal of Production Research**, London, vol. 36, n. 8, p. 2269–2283, ago. 1998.

SRIKAR, B. N.; GHOSH, S. A MILP model for the n-job, m-stage flowshop, with sequence dependent setup times. **International Journal of Production Research**, London, vol. 24, n. 6, p. 1459-72, nov. 1986.

SRISKANDARAJAH, C. Performance of scheduling algorithms for no-wait flowshops with parallel machines. **European Journal of Operational Research**, Amsterdam, vol. 70, n. 3, p. 365-378, nov. 1993,

STAFFORD, E. F. **A Mixed-Integer Linear Programming Model for the Classical** Flowshop **Sequencing Problem.** Columbia: University of South Carolina, 1983. (Working Paper).

_____. On the development of a mixed-integer linear programming model for the standard flowshop. **Journal of the Operational Research Society**, Oxford, vol. 39, n. 12, p. 1163-1174, dez. 1988.

_____.; TSENG, F.T. Two models for a family of flowshop sequencing problems. **European Journal of Operational Research**, Amsterdam, vol. 142, n. 2, p. 282–293, out. 2002.

SUN, J.U.; HWANG, H. Scheduling problem in a two machine flow line with the n-step prior-job-dependent set-up times. **International Journal of Systems Science**, Abington, vol. 32, n. 3, p. 375–385, mar. 2001.

SZWARC, W. A Note on the Flowshop Problem Without Interruptions in Job Processing. **Naval Research Logistics Quarterly,** Hoboken, vol. 28, n. 4, p. 665–669, dez. 1981.

_____.; GUPTA, J.N.D. A flow-shop problem with sequence-dependent additive setup times. **Naval Research Logistics**, Hoboken, vol. 34, n. 5, p. 619-627, out. 1987.

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

TSENG, F. T.; GUPTA, J. N. D.; STAFFORD, E. F. A penalty-based heuristic algorithm for the permutation flowshop scheduling problem with sequence-dependent set-up times. **Journal of the Operational Research Society**, Oxford, vol. 57, n. 5, p. 541–551, jul. 2006.

_____.; STAFFORD JR, E. F. Two MILP models for the N × M SDST flowshop sequencing problem. **International Journal of Production Research**, London, vol. 39, n. 8, p. 1777–1809, mai. 2001.

USKUP G.; SMITH S. B. A branch and bound for two- stage production sequencing. **Operations Research**, Hannover, vol. 23, n. 1, p. 118-136, jan./fev. 1975.

VAN DER VEEN, J. A. A.; VAN DAL, R. Solvable Cases of the No-Wait Flow-Shop Scheduling Problem. ***The Journal of the Operational Research Society***, Oxford, vol. 42, n. 11, p. 971-980, nov. 1991.

WAGNER, H. M. An Integer Linear Programming Model for Machine Scheduling. **Naval Research Logistics Quarterly***,* Hoboken, vol. 6, n. 2, p. 131–140, jun. 1959.

WENG, M. X.; LU, J.; REN, H. Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. **International Journal of Production Economics**, Amsterdam, vol. 70, n. 3, p. 215-226, abr. 2001.

WIDMER, M.; HERTZ, A. A new heuristic method for the flow shop sequencing problem. **European Journal of Operational Research**, Amsterdam, vol. 41, n. 2, p. 186–193, jul. 1989.

WILBRECHT J. K.; PRESCOTT, W. B. The influence of setup time on job shop performance. **Faculty of Management Science**, London, vol. 16, n. 4, p. B274–B280, 1969.

WISMER, D. A. Solution of the flowshop scheduling problem with no intermediate queues. **Operations Research**, Hannover, vol. 20, n. 3, p. 689-697, mai./jun. 1972.

WORTMAN, D. B. Managing capacity: getting the most from your firm's assets. **Industrial Engineering**, vol. 24, n. 2, p. 47–9, 1992.

YANG, W. H.; LIAO, C. J. Survey of scheduling research involving setup times. **International Journal of Systems Science**, Abington, v. 30, n. 2, p. 143-155, fev. 1999.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo